An Investigation of Several Document Classification Algorithms Leading to the Design of an Autonomous Software Agent for Locating Specific, Relevant Information on the World Wide Web

Thesis by

John Lindal

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2001

(Defended December 19, 2000)

Acknowledgements

Of the many people with whom I have associated during my time as a graduate student, my advisor, Dr. Rodney Goodman, and one of his early students, Dr. Padhraic Smyth, have had the most direct influence on this thesis. Dr. Goodman encouraged me to pick a topic that interested me and then allowed me complete freedom to work at my own pace and do as I pleased with the project. Dr. Smyth's seminal work on ITRule provided the foundation for a significant portion of my research. Lengthy discussions with Dr. Smyth during my early years as a graduate student helped me get up to speed on the theory and provided much needed guidance, and his feedback on several different drafts of this thesis near the end was invaluable. Chris Ulmer also deserves thanks for taking the time early on to get me up to speed on the practical issues and cheerfully allowing me to throw away his implementation of ITRule in C so I could write my own version in C++.

Additionally, special thanks must go to Dr. R. David Middlebrook. His influence provided me with both teaching experience and interesting topics in circuit theory with which to tinker, most notably the NEET. His project to develop software which could aid the process of analytically analyzing electrical circuits (Design-Oriented Analysis) provided me with the initial impetus which eventually lead to the development of the JX Application Framework and the founding of New Planet Software, Inc. Without JX, I would have been unable to write Poirot, the end product of my research efforts.

Without JX on which to work over the years, I would likely no longer be sane. Special thanks are due to Glenn Bach and Dustin Laurence for their help in developing JX, even though this often threatened to derail my attempts to graduate. Additional thanks for helping to keep me from cracking up are due to Eileen Lau for all those lunch hours, Sharon Laubach for all those Hawaiian pizza dinners and everything else from rollerblading to cookie baking, Dustin Laurence, Teresa Moore, Ruben Krasnopolsky, and Sarah Yost for role playing and other activities, and Tammy Moore for being herself.

Finally, of course, I would never have succeeded at anything if it were not for the unshakable support of my parents. They encouraged me to grow without forcing me down a particular path and showed me how to survive in the American educational system. Their foresight in 1982 into the future importance of computers provided me with the opportunity to discover that I was born to develop software. As a graduate student, had I not been able to seek refuge from the world at their home as often as necessary and thereby work for extended periods without interruption, this thesis would never have been written.

The research presented in this thesis was primarily supported by the Fannie and John Hertz Foundation.

Abstract

The goal of the research described in this thesis was to design an autonomous software agent that can locate specific, relevant information on the World Wide Web. The first chapter provides the motivation behind this project and a brief overview of the challenges associated with it. The next chapter presents the analysis which led to the development of a new, improved version of the computer program called ITRule. The improvements consist of a new algorithm for classifying documents that outperforms the previous one, significantly enhanced support for data exploration, i.e., the process of extracting information from raw data, and a new algorithm for quantizing numeric variables so they can be used by ITRule. The third part of this thesis compares the performances of three versions of ITRule, two versions of the Naive Bayes classifier, several neural networks, the decision tree algorithm called CART, and a linear support vector machine, in order to determine which one is best suited for selecting relevant web pages. An analysis of the test results shows that a new ITRule classification algorithm, based on cross validation combined with the J-measure, performs best. The fourth and final part of the thesis describes how some of these results were used in the design of a user friendly, autonomous software agent called Poirot that can help World Wide Web users stay up to date on new developments in topics of interest.

Table of Contents

List of Figures

CHAPTER 2

CHAPTER 4

List of Tables

Chapter 1

Introduction

The number of pages on the World Wide Web has grown tremendously over the past few years. As a result, it has become difficult to locate all the existing information on any given subject. Part of the reason is the relatively poor coverage of the World Wide Web provided by most search engines (Selberg and Etzioni, 1995; Lawrence and Giles, 1998). It is therefore usually necessary to consult several of them to ensure that nothing is missed. Furthermore, most engines will only accept a short list of keywords for which to search. Thus, the selectivity is likely to be quite low unless the desired information can be concisely and unambiguously specified.

An additional obstacle is the determination of the optimal keywords to use. The user may initially have only a vague idea of what is required and will therefore often be unable to provide more than a couple of general terms. The result is that the user must manually read through many irrelevant web pages in order to find the special words that are actually important. Consequently, the web searches may have to be repeated many times with both different keywords and engines. In addition, it may be desirable to periodically look for newly created web pages and check for additions and corrections to old pages.

Computers are well suited for processing large volumes of data and should therefore be the ideal tools for automating the above process. Since the World Wide Web can be explored via the existing search engines, the remaining problem is essentially one of classification, i.e., deciding whether or not a particular web page is relevant. Once the user has located a few relevant and irrelevant web pages, an algorithm for constructing a classifier determines what distinguishes the relevant pages from the irrelevant ones.

The associated classification algorithm can then use this information to decide whether or not other web pages are likely to be relevant.

Over the past couple of decades, many different algorithms have been developed in attempts to solve classification problems, but they all have weaknesses. Naive Bayes seems too simple and requires conditional independence (Duda and Hart, 1973), though it sometimes works quite well in spite of this drawback (Domingos and Pazzani, 1996). In the case of neural networks (Haykin, 1999) and support vector machines (Vapnik, 1995), it is difficult to explain how they arrive at decisions unless the transformations are particularly simple (Towell and Shavlik, 1992; Setiono, 1997). Rule based classifiers such as ITRule (Goodman et al., 1992) are designed to be more comprehensible, but continuous variables must be quantized to reduce the search space to a manageable size and obtain accurate probabilities. Instead of solving the problem, this only changes it to that of designing a suitable quantization algorithm. Decision trees (Breiman et al., 1984) avoid the need for quantization, but only at the cost of greedily partitioning the input space instead of fully exploring it and thus being very sensitive to noise in the training data (Dietterich, 1997).

When all the variables associated with a problem are inherently discrete, however, then the major objection to using a rule based classifier vanishes. Thus, by simply treating the presence or absence of words and phrases in a document as Boolean features, the rule based classification system becomes more attractive. The ability of such a classifier to explain how it arrives at its decisions is especially helpful when applied to the search for web pages because the words and phrases chosen by the classifier can be sent directly to

each search engine.

However, making decisions based only on the presence or absence of words and phrases does not utilize all the available information. The frequency of occurrence may also be important. Unfortunately, Boolean variables are not sufficient for representing this additional information. Instead, one must use numeric variables. For this representation, neural networks are suitable, but it is not easy to determine which words to send to a search engine because as mentioned earlier, it is difficult to explain how neural networks arrive at decisions.

Thus, since all of the above classifiers seem to have some disadvantages, this thesis compares the performances of what is believed to be a reasonably representative sample of them in order to determine which approach is likely to yield the simplest and best classification scheme.

Chapter 2 describes the analysis leading to the design of a new, improved version of ITRule. The improvements that are relevant to the task of classifying web pages consist of refinements to the rule based classifier and a faster algorithm for choosing the rules which also improves the classifier's accuracy. Improvements and additions to ITRule's data exploration capabilities are also presented, along with a new algorithm for quantizing continuous, numeric variables.

Experiments designed to compare the performances of ITRule, Naive Bayes, neural networks, support vector machines, and CART with regard to classifying web pages are discussed in Chapter 3. The results demonstrate that, on average, a new ITRule algorithm, which uses cross validation combined with the J-measure, produces a more accurate classifier than any of the

other methods.

The final chapter presents the design of a user friendly, autonomous software agent called Poirot that can search the World Wide Web for new, relevant pages and also report interesting changes to pages discovered previously. In order to illustrate how well this system performs, a detailed description of a demonstration session, complete with screen shots, is also included.

Chapter 2

Improvements to the ITRule algorithms

2.0. Chapter outline

The first version of the computer program called ITRule was developed in conjunction with Padhraic Smyth's thesis (Smyth, 1988). It was given the name ITRule because its design is based on rules and information theory.

There are two tasks to which ITRule can be applied, namely classification, i.e., the process of making decisions based on a finite set of training data, and data exploration, i.e., the process of discovering useful information in raw data. Section 2.1 discusses the issues associated with classification and compares the original algorithm developed by Goodman et al. (1992) with two new algorithms. These algorithms will later be considered for use in Poirot. Section 2.2 presents improvements to ITRule's data exploration algorithms. The process of generating the rules used by the classification and data exploration algorithms is discussed in Section 2.3. Finally, Section 2.4 presents a new algorithm for quantizing continuous variables so they can be used in the rule generation process.

2.1. Review of the classification algorithm

The idea of using rules to make decisions has been around for several decades in the form of expert systems. In such a system, each rule consists of a left-hand side (LHS) that specifies a condition and a right-hand side (RHS) that contains a statement which is true if the LHS is true. A simple example is "If the animal has wings, then it can fly." If only one rule's LHS is satisfied in a particular situation, the result is simply the corresponding RHS. If several rules are satisfied, however, the corresponding RHS's may disagree, thereby requiring a conflict resolution mechanism to make the final decision. As an

example, there may be another rule that states "If the animal is an ostrich, then it cannot fly." It is easy for a human to resolve the conflict in this example, but a computer cannot do so without a lot of background information and a powerful inference algorithm. When rules are generated from raw data, there is usually no background information available. In this case, all conflict resolution mechanisms must be ad hoc because the rules are assumed to be correct, and yet they lead to different conclusions. One common method of handling this problem is to order the rules in some way and then use the first one that applies (Quinlan, 1993; Segal, 1994; Cohen, 1995; Weiss and Indurkhya, 1995). In contrast, Padhraic Smyth decided to try a different approach in order to avoid the problem entirely. He therefore introduced the idea of attaching a probability to each rule (Smyth, 1988). With this approach, the above rule might for instance become "If the animal has wings, then it can fly with probability 0.995," since there are several species of flightless birds. The introduction of a probability, no matter how close to 1.0, eliminates the need for conflict resolution because the rule probabilities can be combined in a Bayesian fashion to calculate the probability of each possible result instead of forcing the expert system to pick a single outcome. This, incidentally, also makes the system more tolerant of missing values, noise, and errors in the data.

The general form of a probabilistic rule is "If y_j is true, then $X{=}x_i$ with probability $p(x_i \mid y_j)$." Let $\{j_1, \ldots, j_n\}$ represent the indices of the rules whose LHS's are true, N_X be the number of possible values of X, and $p(x_i)$ be the prior probability of the i^{th} possible value of X. Assuming that the y_j's are independent when the value of X is known, then the posterior probability of X,

$p(x_i \mid y_{j1} \ldots y_{jn})$, can be calculated in the manner presented by Goodman et al. (1992):

$$p(x_i \mid y_{j_1} \ldots y_{j_n}) = \frac{p(y_{j_1} \ldots y_{j_n} \mid x_i)\, p(x_i)}{p(y_{j_1} \ldots y_{j_n})} \qquad \text{(Bayes' Rule)}$$

$$= \frac{p(x_i) \prod_{k=1}^{n} p(y_{j_k} \mid x_i)}{p(y_{j_1} \ldots y_{j_n})} \qquad \text{(Conditional Independence Assumption)}$$

$$= \frac{p(x_i) \prod_{k=1}^{n} \frac{p(x_i \mid y_{j_k})\, p(y_{j_k})}{p(x_i)}}{p(y_{j_1} \ldots y_{j_n})} \qquad \text{(Bayes' Rule)}$$

$$= \left(\frac{\prod_{k=1}^{n} p(y_{j_k})}{p(y_{j_1} \ldots y_{j_n})} \right) \left(p(x_i) \prod_{k=1}^{n} \frac{p(x_i \mid y_{j_k})}{p(x_i)} \right)$$

In this result, the first term is independent of i, so it can be represented by a constant C:

$$C = \frac{\prod_{k=1}^{n} p(y_{j_k})}{p(y_{j_1} \ldots y_{j_n})}$$

For the posterior probability of X normalized to C, this yields:

$$q_i = \frac{p(x_i \mid y_{j_1} \ldots y_{j_n})}{C}$$

$$\ln(q_i) = \ln p(x_i) + \sum_{k=1}^{n} \ln \frac{p(x_i \mid y_{j_k})}{p(x_i)}$$

Since $\Sigma_i\ p(x_i \mid y_{j1} \ldots y_{jn}) = 1$, p can be calculated from q by simply normalizing:

$$p(x_i \mid y_{j_1} \ldots y_{j_n}) = \frac{q_i}{\sum_{k=1}^{N_X} q_k}$$

Figure 2.1 provides a graphical representation of these calculations. Based on Boolean value of the k^{th} rule's LHS, the rule either contributes zero or $\ln(p(x_i|y_k)/p(x_i))$ to each of the output units. Each output adds in the bias of $\ln(p(x_i))$ and exponentiates the result to get q_i. The normalizer then generates the desired probabilities, $p(x_i|y_{j1}...y_{jn})$. Beyond this, one can attach an extra layer to make a "hard" decision based on the probabilities. The two most common algorithms are to pick either the most probable class or a random class using the probabilities as weights (Goodman et al., 1992).

The most obvious difficulty with the above derivation is the assumption of conditional independence. Even with the results presented by Domingos and Pazzani (1996), it remains a highly dubious proposition that the system will work correctly when an arbitrary set of rules is used in a network of the form shown in Figure 2.1 (Goodman et al., 1992). The problem of picking rules that work well together is analyzed in the next section.

Another problem with this approach is that C is independent of i only if every rule contributes to every output of the network. However, the rules used by Goodman et al. (1992) are of the form "If y is true, then X=x_i with probability $p(x_i|y)$." Thus, each rule contributes to only a single output. In order to avoid this problem, the new implementation of ITRule instead uses rules of the form "If y is true, then X has probability distribution $p(X|y)$."

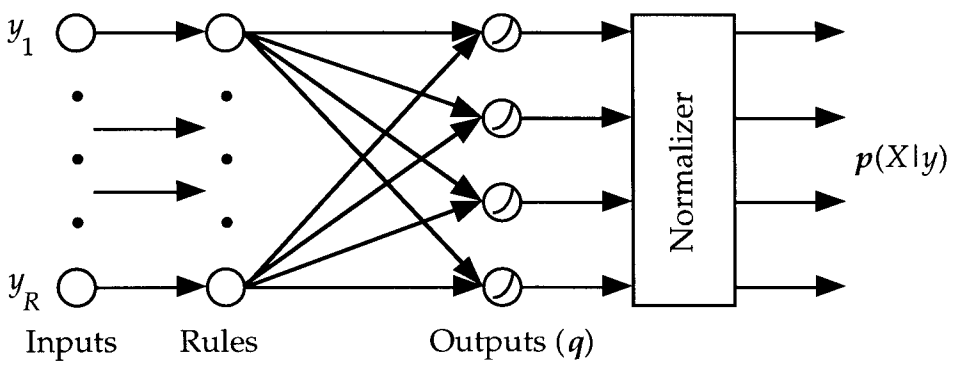


Figure 2.1: Rule-based classification network

2.1.1. Picking the rules to be used by the classification network

As mentioned in the previous section, the assumption of conditional independence is unlikely to be satisfied for an arbitrary set of rules. In an attempt to deal with this, Goodman et al. (1992) used the Minimum Description Length (MDL) principle to choose rules that empirically satisfy this assumption. MDL is based on the theory of data compression. The algorithm greedily picks rules to minimize the number of bits required to encode both the rules and the corrections to the errors that the rules make on the training data. The description length that Goodman et al. used was:

$$L_1 = \frac{N_R}{2} \log_2(N_E) + \sum_{i=1}^{N_E} -\log_2(p(x_{i,correct}))$$

where N_R is the number of rules, N_E is the number of training examples, and $p(x_{i,correct})$ is the predicted probability of the correct result for the i^{th} training example, which is known. The first term represents the number of bits required to transmit the rule probabilities. The terms of the form $-\log_2(p(x_{i,correct}))$ are the average number of bits required to transmit errors as measured by the Kullback-Liebler distance between the correct and predicted probability distributions (Smyth, 1988; Cover and Thomas, 1991).

The first term in L_1 increases as the algorithm adds more rules, while the second term decreases as the classification of the training examples becomes more accurate. The algorithm terminates when no more rules can be found that will decrease the value of L_1.

The results presented by Goodman et al. (1992) show that this approach works remarkably well on a variety of data sets. However, the first term in L_1 is only an ad hoc expression (Smyth, 1996). The theoretically correct descrip-

tion length formula that accounts for all the information required to transmit rules of the form "If Y_1=a and Y_2=b and ... then X has probability distribution $p(X|Y_1=a,Y_2=b,...)$" [1] is obtained as follows:

	Number of bits required to represent:
$L_2 = \frac{(N_{RHS}-1)}{2}\log_2(N_E)$	prior probabilities
$+ \log_2(N_R)$	number of rules
$+ \sum_{j=1}^{N_R}$	for each rule:
$\Big(\log_2(N_{LHS,max})$	number of variables on left-hand side (LHS)
$+ \sum_{i=1}^{n_{LHS,j}} \left(\log_2(N_V) + \log_2(n_{V,i})\right)$	index of each variable and the corresponding value
$+ \frac{(N_{RHS}-1)}{2}\log_2(N_E) \Big)$	probability distribution of right-hand side (RHS)
$+ \sum_{i=1}^{N_E} - \log_2(p(x_{i,correct}))$	corrections

As before, N_E is the number of training examples. Among the N_R rules, $N_{LHS,max}$ is the maximum number of variables that occur on the LHS, $n_{LHS,j}$ is the number of variables on the LHS of the j^{th} rule, N_V is the total number of variables, $n_{V,i}$ is the number of possible values of the i^{th} variable, and N_{RHS} is the number of possible values of the RHS variable.

[1] Even though one can, in principle, use any Boolean LHS, Goodman et al. (1992) chose to use this particular form. The reasons for this choice are discussed in Section 2.3.

The formula for L_2 accounts for sending the prior probabilities for the RHS variable, the number of rules, and the LHS conditions and the RHS probability distribution for each rule. Unfortunately, these terms produce such a large increase[2] in L_2 for every rule which is added that it completely overwhelms the decrease in the last term. Thus, when L_2 was tested on a variety of data sets in the collection available from the University of California at Irvine (UCI), the value of L_2 never decreased, so the algorithm always terminated without choosing any rules. Since the algorithm using L_2 never chooses any rules, in what follows, MDL refers to the original algorithm which uses L_1.

In addition to the fact that the first term in L_1 is ad hoc, MDL has two other undesirable properties. First, it does not provide any indication of how well the classifier will perform on an independent set of test data. Without this feature, one has no idea whether or not the classifier will actually work in practice. In addition, MDL does not allow the user to specify a different cost for each type of mistake.[3] The relative costs of various errors are typically encoded in the function, F_c, to be minimized[4] so it rises more steeply when an error is made whose consequences are more serious. MDL does not allow any such flexibility, however.

[2] Note that a large part of this increase results from attaching a complete probability distribution to each rule instead of a single probability, as discussed at the end of Section 2.1.

[3] A good example of the importance of specifying the relative costs of errors is the case of medical diagnosis when a human life is at stake. It is far more serious to make a mistake that leads to death than to make a mistake that causes the doctor to perform unnecessary but non-fatal procedures on the patient.

[4] The function F_c is often referred to as the cost function.

Cross validation, on the other hand, provides both of these features (Breiman et al., 1984). When applied to ITRule, the basic principle is that one can estimate the accuracy of any given algorithm for picking the rules generated from a data set with N_E examples by measuring the algorithm's accuracy on each data set created by using N_E–1 examples for training and the last example for testing. When repeated for different algorithms, the one with the best accuracy estimate is most likely to produce the best classifier when used on all N_E examples.[5] Thus, the desired accuracy estimate is automatically computed, and it is easy to allow the user to specify the relative costs of various errors as long as one uses algorithms that provide this option.

During the course of the work presented in this thesis, two new algorithms for picking rules have been developed: cross validation combined with steepest descent (CV-SD) and cross validation combined with the J-measure[6] (CV-J). CV-SD replaces the MDL criterion with a user specified cost function and then utilizes cross validation to determine the optimal number of rules to pick. In contrast, CV-J merely picks the N_R rules with the largest J-measures, with N_R determined via cross validation. The pseudocode descriptions of the original MDL algorithm and the new CV-SD and CV-J algorithms are presented in Sections 2.1.3 through 2.1.5.

[5] Provided, of course, that the models have the same Vapnik -Chervonenkis (VC) dimension (Vapnik, 1995).

[6] Refer to Section 2.2 for a discussion of the J-measure.

2.1.2. Speeding up the Minimum Description Length (MDL) and Cross Validation with Steepest Descent (CV-SD) algorithms

The MDL and CV-SD algorithms are both based on steepest descent. They are greedy because it is not feasible to consider all possible subsets of the available rules. In order to further reduce the execution time, a simple bound has been developed to reduce the number of rules that have to be considered at each step.

When using MDL, the first term in L_1 always increases by $\log_2(N_E)/2$ when a rule is added, while the second term cannot decrease by more than

$$\Delta_{max,MDL} = \sum_{i \in S_j} -\log_2(p(x_{i,correct}))$$

where S_j is the set of examples that satisfy the LHS of the j^{th} rule. If one precalculates a matrix specifying which examples satisfy each rule, and one stores an array of $-\log_2(p(x_{i,correct}))$ for the current set of rules, then $\Delta_{max,MDL}$ can be calculated very quickly, i.e., $O(N_R)$ additions, for each new candidate rule. If the result is less than $\log_2(N_E)/2$ or the actual decrease caused by some previously tried rule, then the rule can be ignored, thus avoiding the work of calculating the exact value of $p(x_{i,correct})$ for each example.

A similar bound can be derived for CV-SD. Given a set of candidate rules from which to choose and knowing the minimum value, $F_{c,min}$, that the cost function can attain, the maximum possible decrease that any particular rule can cause is given by

$$\Delta_{max,CV\text{-}SD} = \sum_{i \in S_j} (F_{c,i} - F_{c,min})$$

where $F_{c,i}$ is the current value of the cost function for the i^{th} example. If $\Delta_{max,CV\text{-}SD}$ is less than the actual decrease caused by some previously tried rule, then the rule in question is not worth trying. Again, this can be computed with $O(N_R)$ additions. This is much faster than calculating the actual decrease caused by the rule which requires using the full classification network on each example. Note that the idea behind this bound could be used to provide a second bound for use by the MDL algorithm, but the one discussed above is already sufficiently strong.

The experimental results confirming the effectiveness of these bounds are shown in Tables 2.1 and 2.2. The fraction of rules that are rejected by the bounds ranges from 5.5% to 99.4%. The lowest fractions occur when there are a large number of examples, which increases the value of Δ_{max} by including more terms in the summation, or when no single rule is able to significantly lower the value of the description length or the cost function, so that even a rule with a small value of Δ_{max} is worth trying. Conversely, the largest values occur when a few rules that dramatically decrease the value of the description length or the cost function are encountered early in the process, so that for the rest of the rules, a very large value of Δ_{max} is required in order for a rule to be worth trying.

The results shown in Tables 2.1 and 2.2 were generated by using only simple rules with a single condition on the LHS. In the general case when more specialized rules are also used, the bounds become even more effective. There are far more specialized rules than simple rules, and a specialized rule is more likely to be rejected than a simple rule. The reason is that a special-

ized rule usually applies to fewer examples, thereby reducing the number of terms in the summation for Δ_{max}.

Effectiveness of the bound used in the Minimum Description Length (MDL) algorithm

Name of topic	Training set size	% rules rejected	Name of topic	Training set size	% rules rejected
acq	80	5.5%	gold	20	28.8%
	120	11.6%		30	24.6%
bop	20	19.8%	grain	60	21.3%
	30	24.1%	interest	60	18.1%
coffee	20	15.3%	money-fx	40	14.8%
	30	21.1%		60	17.9%
corn	20	22.9%		80	13.4%
	30	23.2%		80	14.8%
cotton	20	23.7%		80	13.3%
	40	26.9%	money-supply	40	18.3%
cpi	20	29.1%	oilseed	40	17.8%
	30	22.9%	ship	40	18.5%
crude	60	26.7%	soybean	20	17.0%
dlr	20	25.0%		30	19.7%
	30	23.5%	sugar	20	21.5%
earn	80	30.2%		30	22.8%
	120	20.3%	trade	40	28.7%
gnp	20	20.9%		60	29.1%
	30	18.7%	wheat	20	25.5%
				30	32.5%

Table 2.1: The percentage of rules rejected by the bound, $\Delta_{max,MDL}$, presented in Section 2.1.2. The values range from 5.5% to 32.5%. The data sets were generated from the Reuters-21578 collection of news stories. The details of this procedure are explained in Chapter 3.

Effectiveness of the bound used in the Cross Validation with Steepest Descent (CV-SD) algorithm

Name of topic	Training set size	% rules rejected	Name of topic	Training set size	% rules rejected
acq	80	59.4%	gold	20	99.4%
	120	23.8%		30	99.4%
bop	20	98.0%	grain	60	67.5%
	30	97.6%	interest	60	38.8%
coffee	20	98.9%	money-fx	40	44.8%
	30	99.4%		60	86.5%
corn	20	94.5%		80	31.3%
	30	96.2%		80	49.7%
cotton	20	98.8%		80	45.0%
	40	99.4%	money-supply	40	90.0%
cpi	20	97.8%	oilseed	40	88.7%
	30	56.9%	ship	40	88.9%
crude	60	98.3%	soybean	20	99.4%
dlr	20	98.3%		30	93.3%
	30	93.7%	sugar	20	98.7%
earn	80	78.9%		30	97.9%
	120	41.8%	trade	40	41.3%
gnp	20	93.8%		60	23.1%
	30	88.5%	wheat	20	99.4%
				30	92.8%

Table 2.2: The percentage of rules rejected by the bound, $\Delta_{max,CV\text{-}SD}$, presented in Section 2.1.2. The values range from 23.1% to 99.4%. The data sets were generated from the Reuters-21578 collection of news stories. The details of this procedure are explained in Chapter 3.

2.1.3. Pseudocode description of the improved MDL algorithm

```
T        = set of training examples
Ntrain   = number of training examples
Generate rules from T.

C        = set of all candidate rules
           (sorted by decreasing J-measure)
R        = empty set
F        = matrix of which rules apply to each example in T
LIST     = value of -log2(p(x_{i,correct})) for each example in T
           using rule set R
CURRENT  = description length computed from LIST

Repeat until R stops growing
   {
   MIN    = CURRENT
   DELTA  = log2(Ntrain)/2

   For each RULE in C
      {
      Using F and LIST, calculate Δ_{max,MDL} for RULE
      If Δ_{max,MDL} > DELTA
         {
         L = LIST
         Using F, update L by re-classifying each example in T
                  to which RULE applies using R∪RULE.
         Using F, compute NEW_DELTA to be the sum of the changes
                  in the values of the elements of L.
         V = description length computed from L
         If V < MIN
            {
            MIN        = V
            DELTA      = NEW_DELTA
            BEST_RULE  = RULE
            BEST_LIST  = L
            }
         }
      }

   If BEST_RULE was assigned
      {
      Move BEST_RULE from C to R.
      CURRENT  = MIN
      LIST     = BEST_LIST
      }
   }

  R now contains the final set of rules.
```

2.1.4. Pseudocode description of the new CV-SD algorithm

```
NMAX  = maximum number of rules to consider
CV    = list of NMAX zeros
T     = set of training examples
For each example E in T
   {
   T1 = T with E removed
   Call DESCENT with N=NMAX and S=T1 to get contents of CV1.
   Add each element of CV1 to the corresponding element of CV.
   }
BEST = index of minimum element in CV
Call DESCENT with N=BEST and S=T to get final set of rules
(contents of R).
```

```
Subroutine DESCENT
   {
   S = set of training examples
   E = validation example
   Generate rules from S.

   C         = set of all candidate rules
               (sorted by decreasing J-measure)
   R         = empty set
   F         = matrix of which rules apply to each example in S
   CV1       = empty list
   LIST      = cost of classifying each example in S using rule set R
   CURRENT   = sum of values in LIST

   Repeat N times
      {
      MIN         = CURRENT
      DELTA       = 0
      BEST_RULE   = first rule in C

      For each RULE in C
         {
         Using F and LIST, calculate Δ_max,CV-SD for RULE
         If Δ_max,CV-SD > DELTA
            {
            L = LIST
            Using F, update L by re-classifying each example in S
                     to which RULE applies using R∪RULE.
            V = sum of values in L
            If V < MIN
               {
               MIN         = V
               DELTA       = CURRENT - MIN
               BEST_RULE   = RULE
               }
            }
         }

      Move BEST_RULE from C to R.
      Update CURRENT and LIST to match new R.
      Append the result of classifying E using R to CV1.
      }
   }
```

2.1.5. Pseudocode description of the new CV-J algorithm

```
NMAX  = maximum number of rules to consider
CV    = list of NMAX zeros
T     = set of training examples

For each example E in T
   {
   T1 = T with E removed
   Generate rules from T1.

   C    = set of all candidate rules
   R    = empty set
   CV1  = empty list

   Repeat NMAX times
      {
      Move rule with highest J-measure in C to R.
      Append the cost of classifying E using R to CV1.
      }

   Add each element of CV1 to the corresponding element of CV.
   }

N = index of minimum element in CV
Generate rules from T.
Keep N rules with highest J-measure.
```

2.1.6. Upper bounds on the computational complexities of the MDL, CV-SD, and CV-J algorithms

Upper bounds[7] on the amount of time required for each algorithm to finish are given by the following expressions:

$$T_{MDL} = T_{gen} + N_{MDL}\, N_{gen}\, N_{train}\, (T_C + T_{DL})$$

$$T_{CV\text{-}SD} = (N_{train} + 1)\, (T_{gen} + N_{max}\, N_{gen}\, N_{train}\, (T_C + T_{cost}))$$

$$T_{CV\text{-}J} = N_{train}\, (T_{gen} + N_{max}\, (T_C + T_{cost})) + T_{gen}$$

where N_{train} is the number of training examples, T_{gen} is the time required to generate rules from the training data (see Section 2.3), N_{gen} is the number of rules that the user wants generated, N_{MDL} represents the number of times the outer loop of the MDL pseudocode executes, N_{max} is the parameter from the CV-SD and CV-J pseudocode whose value is specified by the user, T_C is an upper bound on the time required to classify an example, T_{DL} is an upper bound on the time required to compute the description length once an example has been classified, and T_{cost} is an upper bound on the time required to evaluate the cost function, F_c, once an example has been classified. Typically, one sets $N_{max} \sim 10^2$ and $N_{gen} \sim 10^3$ to help keep the run time reasonable. It is not worth the effort to even attempt to write down an expression for N_{MDL} since it depends on the statistics of the training data in such a complicated way. However, in all the experiments performed for this thesis, the value of N_{MDL} was always significantly less than 10^2.

Even with these simple approximations, it is clear from the above expressions that CV-J is significantly faster than CV-SD, because CV-J lacks the factor N_{gen} and only has one factor of N_{train} in front of (T_C+T_{cost}). MDL is sig-

[7] Appendices 2-A through 2-D provide the exact expressions.

nificantly faster than CV-J, however, because MDL lacks the term $N_{train}\, T_{gen}$.

2.1.7. Experimental comparison of the run times of the MDL, CV-SD, and CV-J algorithms

Figure 2.2 depicts measured run times of the MDL algorithm. Also shown is a solid line with the slope of the quadratic, N_{train}^2. The MDL run time has approximately the slope of N_{train}^2, rather than the slope of N_{train}, because N_{MDL} tends to increase with the number of training examples. There is clearly no simple relationship between N_{MDL} and N_{train}, however. This was demontrated by one experiment where three different sets of 80 training examples were used, and the resulting values of N_{MDL} were 4, 6, and 8.

The data plotted in Figures 2.3 and 2.4 confirm that the run time of the CV-SD algorithm increases quadratically with the number of training examples, while the run time of the CV-J algorithm is a linear function of N_{train}.

The results presented in Tables 2.3 through 2.6 illustrate the relative execution speeds of the three algorithms and demonstrate the effectiveness of the optimizations discussed in Appendices 2-A through 2-D. The data sets that were used were generated from the Reuters-21578 collection of news stories. The details are discussed in Chapter 3. Here, the only relevant facts are that $N_{gen} = 200$ for all three algorithms[8], $N_{max} = 50$ for CV-SD, and $N_{max} = 200$ for CV-J. The value of T_{DL} was negligible when compared with T_C because modern CPU's have hard-wired circuitry for computing logarithms. The value of T_{cost} was also negligible because the cost function was simply a ma-

[8] There were 100 Boolean input variables, and only rules of the form "if x is true, then..." and "if x is false, then..." were considered, where x represents a single input variable. Thus, setting $N_{gen} = 200$ did not exclude any rules at all.

trix of constants specifying the cost of each possible mistake.

It is important to note that in all the above results, the time required to generate the rules, represented by T_{gen} in Section 2.1.6, was excluded because that analysis was presented in Randy Spangler's thesis (Spangler, 1999).

2.1.8. Experimental comparison of the accuracies of the MDL, CV-SD, and CV-J algorithms

Detailed results comparing the performances of the MDL, CV-SD, and CV-J algorithms are presented in Chapter 3. Here it is only important to note that CV-J was significantly more accurate than both MDL and CV-SD on the data sets which were tested, and that the MDL and CV-SD algorithms had approximately the same performance.

2.1.9. The sensitivities of the MDL, CV-SD, and CV-J accuracies to the choice of adjustable parameters

The only adjustable parameter in the MDL algorithm is N_{gen}. The CV-SD and CV-J algorithms depend on both N_{gen} and N_{max}.

All three algorithms operate on the N_{gen} rules with the highest J-measure (see Section 2.2). The algorithms are only sensitive to the value of N_{gen} when it is small so that potentially useful rules are discarded. If the value of N_{gen} is set large enough to discard only rules that are certain to be useless, then the exact value will not affect the accuracy of the resulting classifier. As mentioned in Section 2.1.6, N_{gen} is typically of the order of 10^3. In all the experiments where N_{gen} caused rules to be discarded, none of the algorithms picked rules near the bottom of the list of remaining rules. This indi-

cates that keeping the top few thousand rules is sufficient. Using significantly more would only slow down the computations.

The same argument is applicable to N_{max} for the CV-SD and CV-J algorithms. When using the values of N_{max} presented in Section 2.1.7, both CV-SD and CV-J always chose significantly fewer than N_{max} rules. Since these algorithms search for the minimum misclassification cost, this shows that the minimum is achieved by using only a few rules from the top of the list of input rules, so appending more rules to the end of the list would have no effect.

2.1.10. Extrapolating from the training data

As with all other classification results obtained from a limited number of training examples, one should not expect ITRule to provide accurate answers when it has to extrapolate, i.e., when the value of a variable falls outside the range of the training data. In order to guard against this situation, the latest implementation of ITRule warns the user about each variable whose value is out of range. In such cases, the user must make the final decision whether or not to trust the result.

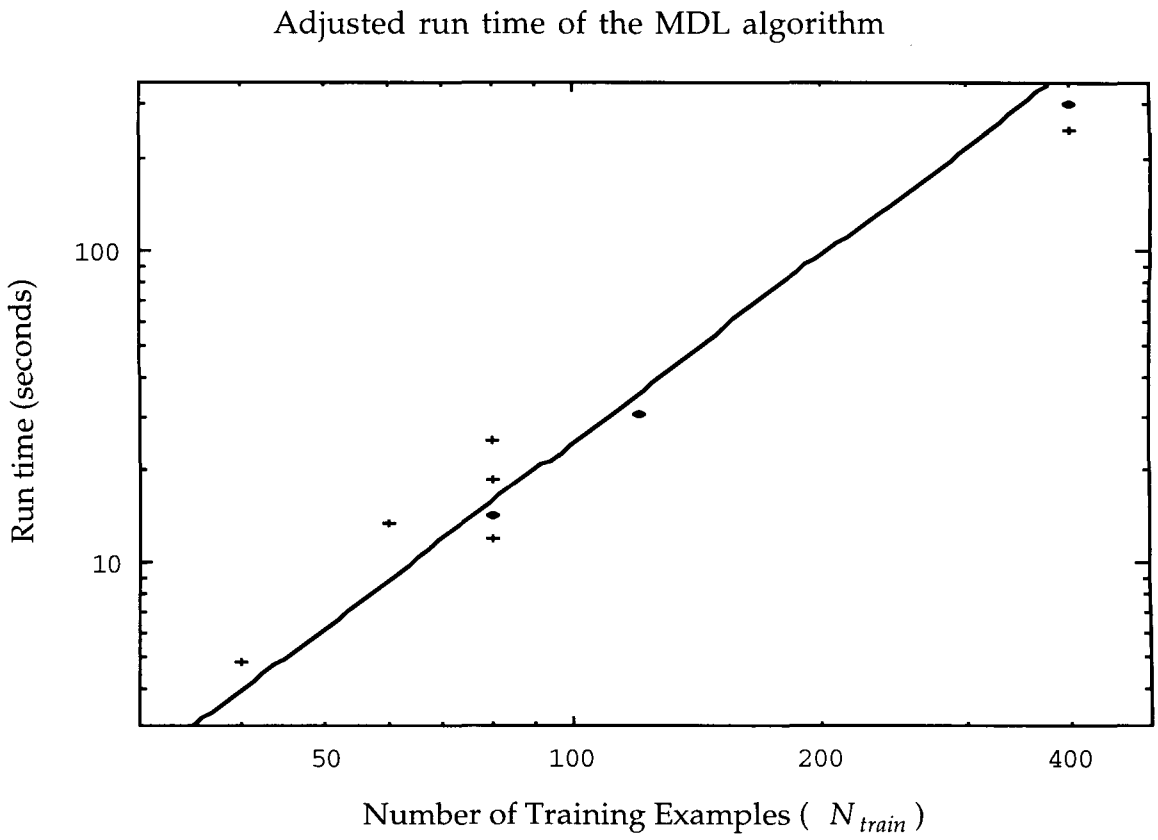


Figure 2.2: The discrete points show the adjusted run times of the MDL algorithm as a function of the number of training examples. The solid line has the slope of N_{train}^2. As discussed in Section 2.1.7, the data appear to fit a quadratic instead of a linear function of N_{train} because N_{MDL} tends to increase along with N_{train}. The run times were adjusted to remove the effect of $\Delta_{max,MDL}$, since the percentage of rules that this bound rejects varies, as shown in Table 2.1.

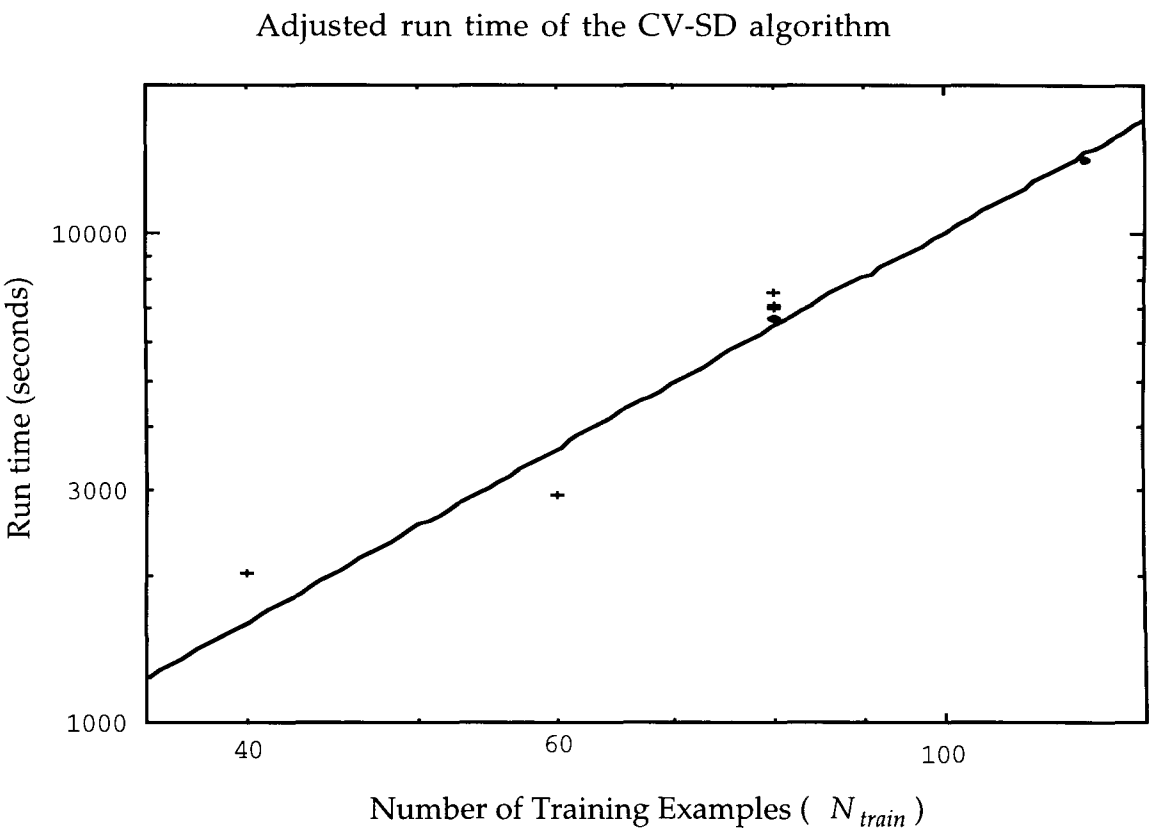


Figure 2.3: The discrete points show the adjusted run times of the CV-SD algorithm as a function of the number of training examples. The solid line has the slope of N_{train}^2. By comparing the data points and the line, it is evident that the CV-SD run time is approximately proportional to N_{train}^2. The run times were adjusted to remove the effect of $\Delta_{max,CV\text{-}SD}$, since the percentage of rules that this bound rejects varies dramatically, as shown in Table 2.2.

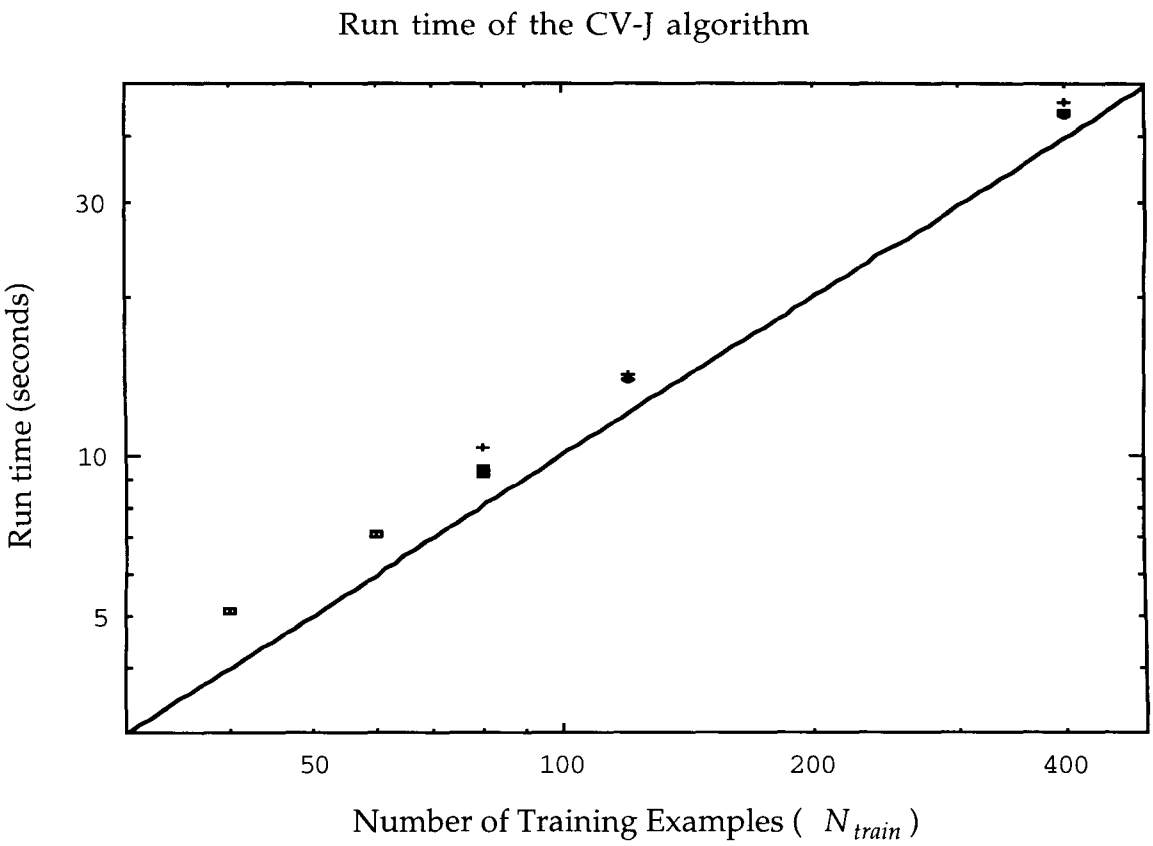


Figure 2.4: The discrete points show the measured run times of the CV-J algorithm as a function of the number of training examples. The solid line has the slope of N_{train}. By comparing the data points and the line, it is evident that the CV-J run time is approximately proportional to N_{train}.

	Run time (seconds)	Measured speed relative to CV-J	Upper bound on speed relative to CV-J	Speed increase from optimizations
MDL	1.49	0.58	2	3.45
CV-SD	6.92	2.68	1000	373.13
CV-J	2.58			

Table 2.3: A comparison of run times on a data set generated from articles dealing with the international balance of payments (referred to as "bop" in Reuters-21578). For this data set, N_{train}=20, and the computer run yielded N_{MDL} = 2. The increase in speed due to optimizations includes the effect of the bounds discussed in Section 2.1.2. The other optimizations are discussed in Appendices 2-A and 2-B.

	Run time (seconds)	Measured speed relative to CV-J	Upper bound on speed relative to CV-J	Speed increase from optimizations
MDL	4.68	0.66	2	3.03
CV-SD	2981.67	418.19	3000	7.17
CV-J	7.13			

Table 2.4: A comparison of run times on a data set generated from articles dealing with trade (referred to as "trade" in Reuters-21578). For this data set, N_{train}=60, and the computer run yielded N_{MDL} = 2. The increase in speed due to optimizations includes the effect of the bounds discussed in Section 2.1.2. The other optimizations are discussed in Appendices 2-A and 2-B.

	Run time (seconds)	Measured speed relative to CV-J	Upper bound on speed relative to CV-J	Speed increase from optimizations
MDL	13.5	1.44	5	3.47
CV-SD	2703.29	288.81	4000	13.85
CV-J	9.36			

Table 2.5: A comparison of run times on a data set generated from articles dealing with corporate acquisitions and mergers (referred to as "acq" in Reuters-21578). For this data set, N_{train}=80, and the computer run yielded N_{MDL} = 5. The increase in speed due to optimizations includes the effect of the bounds discussed in Section 2.1.2. The other optimizations are discussed in Appendices 2-A and 2-B.

	Run time (seconds)	Measured speed relative to CV-J	Upper bound on speed relative to CV-J	Speed increase from optimizations
MDL	21.85	1.54	5	3.25
CV-SD	11126.55	785.22	6000	7.64
CV-J	14.17			

Table 2.6: A comparison of run times on a data set generated from articles dealing with corporate earnings (referred to as "earn" in Reuters-21578). For this data set, N_{train}=120, and the computer run yielded N_{MDL} = 5. The increase in speed due to optimizations includes the effect of the bounds discussed in Section 2.1.2. The other optimizations are discussed in Appendices 2-A and 2-B.

2.2. Data exploration

The idea of using rules to represent information has been around for a long time. When the goal is to extract information from raw data, the problem is to decide which rules to keep out of the very large set of all rules generated from the data (see Section 2.3). In his thesis, Smyth suggested picking the rules with the largest J-measures (Smyth, 1988). For the probabilistic rule, "If y is true, then x is true with probability $p(x|y)$," the J-measure calculates how much useful information the rule provides about the raw data. This is done by combining the probability of the LHS (y) and the Kullback-Liebler distance (Cover and Thomas, 1991) between the rule probability and the prior probability of the RHS (x) in the following way:

$$J = p(y)\, D(p(x|y)\,||\,p(x)) = p(y)\left(p(x|y)\log\left(\frac{p(x|y)}{p(x)}\right) + (1-p(x|y))\log\left(\frac{1-p(x|y)}{1-p(x)}\right)\right)$$

The J-measure ranks a rule highly if its LHS is satisfied relatively often, i.e., when $p(y)$ is large, and the rule provides new information about the RHS, i.e., when $p(x|y)$ is different from $p(x)$. This gives a low rank to rules that are unlikely to apply or that predict what is already known.

The original ITRule algorithm only printed a list of rules sorted by their J-measures. There is, however, quite a bit more that can be done to help the user make sense of raw data. In particular, the new version of ITRule provides additional information about the data, helps the user clean the data by locating missing values and unnecessary variables, and provides additional methods of organizing the final set of rules beyond merely sorting them by their J-measures.

2.2.1. Correlations between variables

Let x_i and y_j represent particular values of the variables X and Y, respectively. Correlations between the mutually exclusive values of X and Y can then be presented in the form "$X=x_i \Leftrightarrow Y=y_j$ with probability p." This approach exposes the simplest relationships hidden in the data and can be useful during the first stage of data exploration when the user is searching for an initial understanding of the problem.[9] If one defines S_1 and S_2 to be the subsets of examples where $X=x_i$ and $Y=y_j$, respectively, then the probability is obtained from:

$$p = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

Correlations are usually only interesting when p is close to one. The new version of ITRule allows the user to set a threshold on p so correlations below the threshold are not reported. One can visualize the set of all correlations as an undirected graph where each node represents a statement, e.g., $X=x_i$, and each edge represents an interesting correlation. As illustrated in Figure 2.5, by introducing a third dimension, the strength of each correlation can be indicated by the height of the edge above the base plane. The threshold is then a horizontal plane at a particular height. All edges below this plane are ignored.

Unlike in Boolean logic, probabilistic correlations are not transitive. This can be seen in the example in Figure 2.5 where $X=x_i$ is correlated with $Y=y_j$, and $Y=y_j$ with $Z=z_k$, but $X=x_i$ is not considered to be correlated with $Z=z_k$ (as indicated by the dashed edge) because it is below the threshold.

[9] Correlations can also be used to clean the data, as discussed in Section 2.2.2.3.

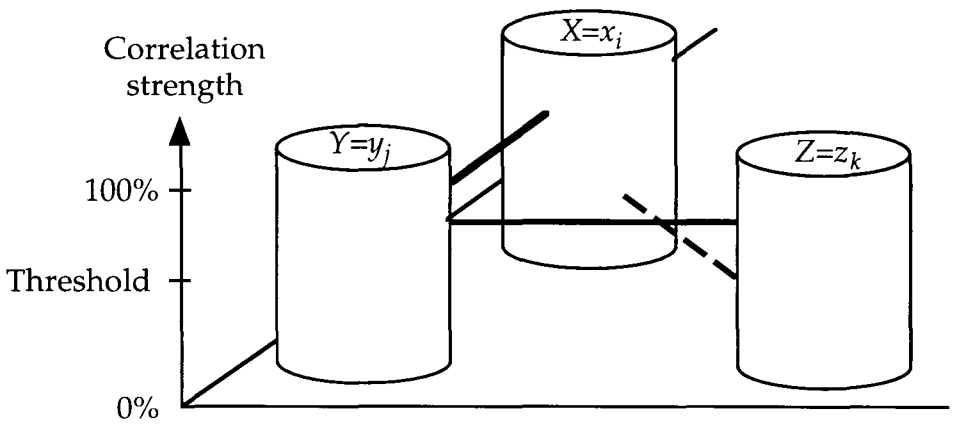


Figure 2.5: Example illustrating correlation strengths

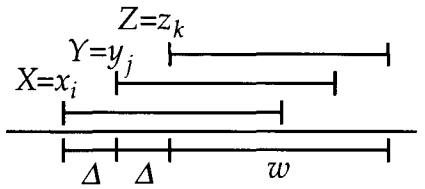


Figure 2.6: Venn diagram with three equal intervals displaced laterally

A lower bound on the strength of "$X=x_i \Leftrightarrow Z=z_k$," represented by p_{XZ}, can be derived from the strengths of "$X=x_i \Leftrightarrow Y=y_j$" and "$Y=y_j \Leftrightarrow Z=z_k$," represented by p_{XY} and p_{YZ}, respectively. As an example, if one assumes that the last two strengths are equal, and that all three statements have the same number of supporting examples, then the minimum possible overlap between the examples satisfying $X=x_i$ and $Z=z_k$ is illustrated by the one-dimensional Venn diagram shown in Figure 2.6. Since all three intervals have width w, and the overlaps between $X=x_i$ and $Y=y_j$ and between $Y=y_j$ and $Z=z_k$ are both $w-\Delta$, one has:

$$p \equiv p_{XY} = p_{YZ} = \frac{w-\Delta}{w+\Delta}$$

which yields:

$$\frac{w}{\Delta} = \frac{1+p}{1-p}$$

Since the overlap between $X=x_i$ and $Z=z_k$ is $w-2\Delta$, the lower bound on p_{XZ} is therefore

$$p_{XZ} = \frac{w-2\Delta}{w+2\Delta} = \frac{\frac{1+p}{1-p}-2}{\frac{1+p}{1-p}+2} = \frac{3p-1}{3-p}$$

The upper bound on p_{XZ} is obviously 1. This occurs when the three sets depicted in Figure 2.6 are perfectly aligned, i.e., when $\Delta = 0$.

2.2.2. Cleaning the data

There are many ways that data can be cleaned. The particular tasks with which the new ITRule can help are detecting noise in the form of missing values or irrelevant variables and detecting redundant variables that can be eliminated.

2.2.2.1. Detecting missing values in the data

Missing information can easily arise in practice due to human errors in data entry, e.g., in medical records, insurance claims, etc. These missing values can always be treated simply as noise. However, the new version of ITRule also provides the option to generate rules specifically about the missing data. This feature may help the analyst identify the problems that are causing the values to be absent in the first place so that these data acquisition problems can be corrected.

2.2.2.2. Identifying irrelevant variables

A variable is defined to be noise if there is no significant correlation between it and any of the other variables. In terms of probabilistic rules, this translates into the statement that if the variable is used on the RHS of a rule, then the rule's probability is the same as the variable's prior probability regardless of the rule's LHS. Since the Kullback-Liebler distance is the appropriate measure of the difference between probabilities, a variable is noise if all rules with that variable on the RHS have a low J-measure. Thus, by using the J-measure, ITRule can automatically detect irrelevant variables as long as sufficient data are available to calculate accurate probabilities.

Unfortunately, one must sometimes work with data sets that have relatively few examples. When data points are scarce, the statistics become less accurate, and it is therefore more likely that ITRule will compute a large J-measure for a rule that does not actually provide any information.

As an example, consider the situation where two variables, x and y, are independent. Let the prior probability distribution of x be $p(x)$. When x is conditioned on a particular value of y, the true distribution is still $p(x)$, but the distribution computed from the data is likely to be at least slightly different. One can think of the examples that support $p(x \mid y=y_1)$ as having been drawn without replacement from the pool of all examples. Since x and y are independent, the probability distribution that controls the sampling is $p(x)$. This situation is illustrated by the simple example in Figure 2.7.

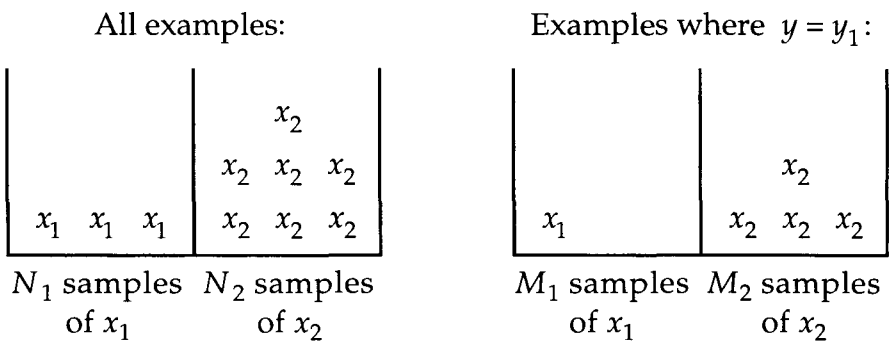


Figure 2.7: The two bins on the left contain all the available samples of x. Thus, the prior probabilities of x, i.e., $p(x=x_1)$ and $p(x=x_2)$, are $N_1/(N_1+N_2)$ and $N_2/(N_1+N_2)$, respectively. The two bins on the right-hand side of the figure contain the samples of x where $y=y_1$. The corresponding conditional probabilities are

$$p(x=x_1 \mid y=y_1) = \frac{M_1}{M_1 + M_2} \qquad p(x=x_2 \mid y=y_1) = \frac{M_2}{M_1 + M_2}$$

When the probability distribution associated with the rule "if $y=y_1$, then x" is significantly different from $p(x)$, then ITRule will report that the rule is interesting. In principle, one could provide an indication to the user of which such rules are likely to be spurious by estimating the probability of obtaining a Kullback-Liebler distance, D, greater than the observed value, D_{obs}, under the hypothesis that x and y are independent. If this probability is small enough (e.g., 5%), then one could assume that the rule is not spurious.

For the case depicted in Figure 2.7, let $N=N_1+N_2$ and $M=M_1+M_2$. If one assumes that N is large enough so that one never runs out of examples in either bin, i.e., that $M<\min(N_1,N_2)$, then one can treat the sampling as a sequence of Bernoulli trials. The probability of drawing x_1 is therefore $p=N_1/N$, while the probability of drawing x_2 is $1-p=N_2/N$. The probability of drawing M_1 samples of x_1 and M_2 samples of x_2 is given by the Binomial distribution:

$$P(M_1) = \binom{M}{M_1} p^{M_1} (1-p)^{M-M_1}$$

In the case when $M \geq \min(N_1,N_2)$, one can run out of examples in one of the bins. This complicates the calculation significantly, so it is best done numerically on a computer rather than analytically. Regardless of how one computes $P(M_1)$, however, one obtains the desired probability for a given value of M from

$$\text{Prob}[D > D_{obs}] = \sum_{M_1:\ D(M_1) > D_{obs}} P(M_1)$$

where

$$D(M_1) = \frac{M_1}{M} \log \frac{\left(\frac{M_1}{M}\right)}{p} + \frac{M-M_1}{M} \log \frac{\left(\frac{M-M_1}{M}\right)}{1-p}$$

Generalizing this derivation to the case where x has more than two possible values is straightforward. Note that the maximum value of $P(M_1)$ occurs when $D(M_1)$ is zero. The situation is similar to testing hypotheses with a Gaussian distribution. In fact, in the limit when N is large, the exact Binomial distribution can be approximated by a Gaussian distribution with mean Mp and variance $Mp(1-p)$. Of course, in this limit, one has enough examples that the above calculations are unnecessary. However, the Gaussian approximation may be useful for obtaining estimates of the probabilities even when the value of N is only moderately large.

There is one theoretical and one practical problem with the above approach. The theoretical complication is that the more often one repeats a statistical test, the more likely it becomes that the test will give the wrong answer at least once. One solution to this difficulty is the Bonferroni method presented by Bay and Pazzani (1999). Unfortunately, the practical problem appears to be insurmountable. The time required to calculate $P(M_1)$ when $M \geq \min(N_1, N_2)$ is exponential in M since one has to consider separately each way of obtaining M_1 samples of x_1 and M_2 samples of x_2. This makes the entire approach unfeasible.

2.2.2.3. Using correlations to discard redundant variables

The new correlation feature discussed in Section 2.2.1 can also be used to clean the data. If one finds that every value of one variable is correlated with a different value of another variable, then one should consider discarding

one of them.[10] This will reduce the time required for ITRule to consider all possible rules because the execution time is an exponential function of the number of variables (see Section 2.3). It is not appropriate for ITRule to automatically search for redundant variables, however, because there might be values that are not present in the data or there might not be enough examples to justify some of the correlations. It is a judgment that the user must make. As a guideline, if X and Y have the same number of possible values, and there is a one-to-one correspondence between them, then either X or Y can be discarded. If X has N_X possible values, Y has N_Y possible values, $N_X > N_Y$, and N_Y of the possible values of X each has a one-to-one correspondence with one of the values of Y, then X should be discarded since it has values that are unused. (It is impossible for two values of X to both be correlated at a level above 50% with one value of Y since the values of X are mutually exclusive.) These cases are illustrated in Figure 2.8.

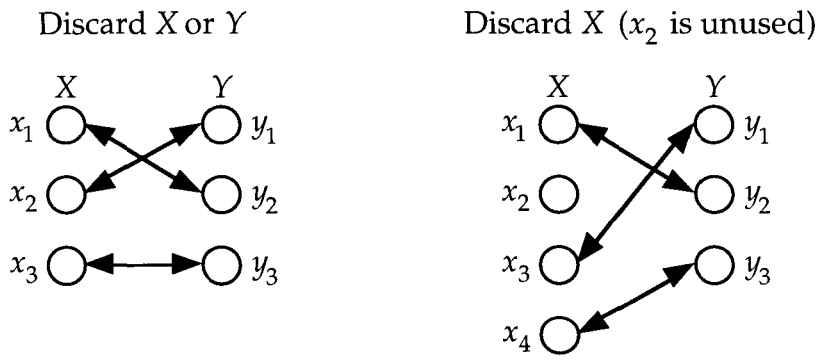


Figure 2.8: When two variables are fully correlated, one can be discarded

[10] One could simply calculate the mutual information between the variables (Ezawa and Schuermann, 1995), but this single value does not reveal the underlying structure of how the variables are related. The Markov blanket algorithm presented by Koller and Sahami (1996) also has this problem. John et al. (1994) provides a good overview of some of the other automated algorithms.

2.2.3. Identifying significant rules

One can generate a nearly unlimited number of different rules from a given set of data (see Section 2.3). Without a way to filter and organize these rules, however, this merely replaces the problem of studying a large quantity of raw data with the problem of studying a large number of rules. The new version of ITRule includes several features that can assist the user in dealing with this problem.

The original version of ITRule sorted the rules by their J-measures and discarded all rules with J-measures below a user specified threshold (Smyth, 1988). The actual value of the J-measure is probably meaningless to most users, however. The new version of ITRule therefore replaces the threshold on the J-measure with a threshold on the minimum number of examples to which a rule applies, i.e., the rule's support.

The new version of ITRule also displays the likelihood ratio, $p(x|y)/p(x)$, for each rule. While it may be difficult to decide whether or not a rule with probability 0.3 is important, it is clear that a rule with likelihood ratio 10 is quite interesting because conditioning on the LHS makes the RHS ten times as likely to be true.

2.2.3.1. Discarding subsumed rules

In addition to the simple rule selection tools discussed above, ITRule also has a filter that is based on the concept of subsumption. This is a generalization of the J-measure. The justification for discarding rules that have low J-measures is that these rules do not provide sufficient information beyond that of the prior probabilities, i.e., the simplest possible rule.

Subsumption extends this concept by using the Kullback-Liebler distance to discard rules that do not provide any information beyond that of all simpler rules. As an example, the analysis of a data set might yield the following four rules for the condition x:

R_1: For the entire data set, x is true with probability p_1

R_2: For the subset of examples where y is true,
x is true with probability p_2

R_3: For the subset of examples where z is true,
x is true with probability p_3

R_4: For the subset of examples where y and z are true,
x is true with probability p_4

Here, rule R_2 is discarded if the difference between p_1 and p_2 is not considered significant. The same applies to R_3. If R_2 and R_3 are discarded, then R_4 is discarded if p_4 is close to p_1. If R_2 is kept, however, R_4 is discarded if p_4 is close to p_2 but is set aside in a separate list of rules if p_4 is sufficiently different from p_2 but close to p_1. (The equivalent action is taken if R_3 is kept.) The reason for the separate list is that some users consider R_4 to be just as interesting as R_2 because it tells them that combining the two conditions y and z changes the probability of x, while others feel that R_4 is only of secondary interest since p_4 is close to p_1. By presenting two lists of rules in a situation like this one, ITRule allows the user to first grasp the main ideas by studying the first list and then look for subtleties by examining the second list.

Since the actual values of the Kullback-Liebler distance are meaningless to most users, the condition "close" is specified as a fraction of the maximum Kullback-Liebler distance between any two rules. Aggressive filtering might

use 0.5, while conservative filtering might use 0.1.

2.3. Generating rules

So far, the discussion has ignored the issue of how to generate the rules that are used by the algorithms for classification and data exploration because these algorithms are independent of how the rules are generated. Any set of probabilistic rules can in principle be used. In his thesis, Smyth presented an algorithm that, given a set of discrete valued variables, computes the probability for each possible conjunctive rule of the form "If Y_1=a and Y_2=b and ... then X=x_i" (Smyth, 1988; Smyth and Goodman, 1992). (The number of terms on the LHS is called the rule's "order.") Unfortunately, there are two serious problems with this approach. The first is the restriction that the variables cannot be continuous, numeric variables. This is unavoidable, as discussed in the next section. The second problem is the computational complexity of the algorithm. If there are N variables available for use on the LHS, and the i^{th} one has n_i possible values, then the number of possible rules is given by:

$$\left(\prod_{i=1}^{N}(n_i+1)\right)-1$$

The reason is that each of the N variables either takes one of its n_i values or is excluded, and the single case with all variables excluded is not relevant. Even in the case of only 20 Boolean variables, there are over 3 billion possibilities. The solution is to switch from a possibility driven algorithm to a data driven algorithm, i.e., only the rules that are actually supported by the data should be considered. This is the basis for the SpanRule algorithm (Spangler, 1999) which is used in the current implementation of ITRule.

However, this algorithm is unfortunately not fast enough either, when it comes to generating all the supported rules in a reasonable amount of time. The reason is that even though the algorithm is only $O(n \log_2(n))$ in the number of examples and does not depend significantly on the number of values for each variable, it is still exponential in the number of variables. ITRule therefore only generates rules from the first order up through some maximum rule order specified by the user, typically two or three. Since low order rules also have the largest support, this provides the additional benefit of generating only the rules that are most likely to have a large J-measure.[11] In addition, since the classification algorithm does not place any restrictions on the form of the LHS that can be used[12], ITRule allows the user to enter arbitrary Boolean expressions to be considered along with the rules that are automatically generated.

2.4. Quantizing numeric variables

As discussed in the previous section, ITRule can only generate rules for variables with a limited number of discrete values. Thus, numeric variables must be quantized before they can be used. This is a fundamental problem with all probabilistic approaches. Since each particular value of a continuous, numeric variable is by itself very unlikely to occur, one must consider ranges of values in order to get enough examples to generate accurate probabilities. Of course, it also makes intuitive sense that only ranges should matter, not exact values. Clearly, trying every possible quantization is far too time con-

[11] Note that cleaning the data by removing redundant variables also reduces the run time. This is discussed in Section 2.2.2.3.

[12] The subsumption algorithm (see Section 2.2.3.1) requires that every LHS *be a specialization* of some other, simpler LHS.

suming. This section therefore presents the theory behind Minimal Entropy Partitioning (MEP), a newly developed algorithm for obtaining a useful quantization.

The most general method for scalar quantization is first to transform the examples by passing a subset of the available variables through a function $f: \Re^{N_V} \rightarrow \Re$ and then to determine an ordered set of breakpoints, $b_2 < b_3 < ... < b_{N_I}$, that separate the real number line into intervals $[b_i, b_{i+1})$, with $b_1 = -\infty$ and $b_{N_I+1} = +\infty$. The index of the interval containing an example becomes the value of the new discrete variable for that example.

Ideally, the examples in an interval will all belong to the same class, as illustrated in Figure 2.9. In practice, however, this is usually not possible. Instead, one can only attempt to maximize the homogeneity of each interval by adjusting its breakpoints to minimize the entropy of the examples that it contains, as illustrated in Figure 2.10. Since decreasing the entropy of one bin may increase the entropy of the adjacent bins, it is necessary to minimize the weighted sum of the entropy of all the bins, H_{total}, which is given by:

$$H_{total} = \sum_{i=1}^{N_I} \frac{N_i}{N_E} H_i$$

$$H_i = -\sum_{c=1}^{N_C} p_{i,c} \log(p_{i,c})$$

$$p_{i,c} = \frac{N_{i,c}}{N_i}$$

Here, N_I is the number of intervals, N_c is the number of classes, N_E is the total number of examples, N_i is the number of examples in the i^{th} interval, and $N_{i,c}$ is the number of examples in the i^{th} interval that belong to class

c. Adjusting all the breakpoints between intervals to minimize the value of H_{total} yields the optimal quantization for a given N_I. The factor N_i/N_E ensures that nearly empty, mixed bins do not overwhelm the contribution of large, nearly homogeneous bins.

An existing algorithm called Recursive Minimal Entropy Partitioning (RMEP) (Dougherty et al., 1995) uses this idea in a way that is similar to growing a decision tree. It begins by placing a single breakpoint ($N_I = 2$) and then recursively applies itself to each newly created interval. The Minimum Description Length (MDL) principle is used to stop the process before it creates a separate interval for each example. As with decision trees, the process does not backtrack, so if the first breakpoint is poorly chosen, the effort is doomed. The new MEP algorithm avoids this problem by simultaneously adjusting all the breakpoints (via any minimization algorithm) when $N_I > 2$. While this is guaranteed to always be able to do at least as well as RMEP, since the search space for MEP is larger, MEP has the disadvantage that the number of intervals is not determined automatically. In order to avoid the problem of choosing a single quantization, ITRule has been modified to allow multiple quantizations of the same variable to be specified and to automatically ensure that only one quantization of a variable is used in a particular rule. This allows all the quantizations that appear promising to be tried simultaneously.[13]

In addition, MEP has the advantage that it can optimize the transformation f at the same time that it adjusts the breakpoints. It simply asks the minimization algorithm to treat the parameters of f as additional values to

[13] This also allows one to try entirely different algorithms for quantization. Dougherty et al. (1995) and Kohavi et al. (1997) have, for instance, shown that using ten equally wide intervals often performs surprisingly well.

be optimized. This expands the search space for MEP even further, again ensuring that MEP will be able to do at least as well as RMEP as long as MEP does not get stuck in a local minimum.

In the simplest case, f is the scalar identity transform, producing the value of a single, numeric variable. In multi-dimensional problems, however, the optimal decision boundaries are usually not parallel to any of the axes and are often not even linear. Thus, f typically combines several or even all of the variables. If one wishes to enforce linear decision boundaries, one must use a linear combination of the variables. Otherwise, one could use a two layer neural network to provide an arbitrary, non-linear transformation.

It should be noted that if all the data variables are continuous, then MEP provides a complete classifier for a discrete class variable if one uses an algorithm such as cross validation to determine the optimal number of breakpoints. On the other hand, if one has a mixture of discrete and continuous data variables, then ITRule should be used to take into account the additional information provided by the discrete data variables. In addition, it may not be appropriate to combine all the numeric data variables into a single transformation. The final result will be much easier to understand if one uses several transformations, each combining only related variables, instead of a single transformation that produces values to which even the experts cannot assign any meaning, as is sometimes the case when using a single neural network.

2.4.1. A comparison of the Minimal Entropy Partitioning (MEP) and Recursive Minimal Entropy Partitioning (RMEP) algorithms

Two simple experiments were performed to compare the performance of MEP and RMEP. First, a fully separable, one-dimensional data set was created as shown in Figure 2.9. The letters o and x represent data points belonging to two different classes, and the carets (^) indicate the optimal breakpoint locations. The RMEP algorithm stopped after placing only the left breakpoint. When instructed to use two breakpoints, MEP correctly placed both of them. When only one breakpoint was requested, MEP correctly stated that the left and right positions were the two best choices.

The second experiment used the one-dimensional data set shown in Figure 2.10 to test the case where the data is not fully separable. In this case, the carets indicate the Bayes-optimal breakpoint locations. This time, RMEP also stopped after placing only one breakpoint. In this case, however, it did not even place it in an optimal location. Instead, it placed it between the third and fourth data points from the left. MEP, on the other hand, correctly placed breakpoints at the Bayes-optimal locations when instructed to use two breakpoints. Moreover, when told to use only one breakpoint, MEP correctly stated that the positions of the two carets were the best choices.

Even though these two examples are quite simple, they clearly demonstrate that MEP can produce significantly better results than RMEP.

```
o o o o o x x x x x x x x x x o o o o o
      ^                     ^
```

Figure 2.9: Separable data set for comparing MEP and RMEP. The carets indicate the locations of the optimal breakpoints.

```
o o o x o x o x x x x x x o x o x o o o
      ^                     ^
```

Figure 2.10: Non-separable data set for comparing MEP and RMEP. The carets indicate the locations of the Bayes-optimal breakpoints.

2.4.2. The Minimal Entropy Partitioning (MEP) algorithm

This section presents the details of the MEP algorithm. For computational reasons discussed in the next section, MEP does not use the actual Shannon entropy function (Cover and Thomas, 1991). Instead, MEP minimizes the function F, defined as follows:

$$F = \sum_{i=1}^{N_I} \frac{M_i}{N_E} G_i$$

$$G_i = \frac{1}{N_C - 1} \sum_{c=1}^{N_C - 1} \begin{cases} 4\, q_{i,c}^2 & 0 \le q_{i,c}^2 \le \frac{1}{2} \\ 4\,(1 - q_{i,c})^2 & \frac{1}{2} < q_{i,c}^2 \le 1 \end{cases}$$

$$q_{i,c} = \frac{m_{i,c}}{M_i}$$

$$M_i = \sum_{c=1}^{N_C} m_{i,c}$$

$$m_{i,c} = \sum_{y_j \in \text{ class } c} m_i(y_j)$$

Here, N_C is the number of classes, N_E is the number of examples, and N_I is the number of intervals. As discussed in Section 2.4, y_j can be either the value of a single data variable or a transformation of several data variables. $m_i(y)$ is called the interval membership function. It has the shape shown in Figure 2.11. Any function with this shape can be used as long as it is differentiable at all points. One simple example is:

$$m_i(y) = \begin{cases} t\left(\dfrac{y-(b_i-\delta)}{2\delta}\right) & b_i-\delta < y < b_i+\delta \\ 1 & b_i+\delta \le y \le b_{i+1}-\delta \\ t\left(\dfrac{(b_{i+1}+\delta)-y}{2\delta}\right) & b_{i+1}-\delta < y < b_{i+1}+\delta \\ 0 & \text{everywhere else} \end{cases}$$

$$t(y) = 3y^2 - 2y^3$$

Here, δ is an adjustable parameter that determines the width of the transition regions in Figure 2.11. The choice of an appropriate value for δ is discussed in the next section. RMEP implicitly uses this function with δ=0 when it computes the entropy of the examples in each interval. One can apply MEP in this case, but only if one uses a minimization algorithm that does not require continuity, e.g., simulated annealing (Corana et al., 1987) or a simplex algorithm (Nelder and Mead, 1965). In this case, F will not be continuous because it remains constant while a breakpoint moves within the interval between two adjacent examples and changes abruptly when the breakpoint passes an example. By setting δ>0, F becomes differentiable, thereby allowing it to be minimized via gradient based algorithms.

2.4.3. Analysis of the function, *F*, minimized by the Minimal Entropy Partitioning (MEP) algorithm

In the definition of F presented in the previous section, q and G are used instead of p and H, respectively. This was done to indicate that $q_{i,c}$ and G_i do not represent the real probabilities and entropies. However, q becomes a

probability in the limit when δ=0. (It should be noted that even when δ>0, $0 \leq q_{i,c} \leq 1$ and $\Sigma_c\, q_{i,c} = 1$.)

The entropy, H, is not used in the definition of F because it causes F to be concave down, as illustrated by the example in Figure 2.13 for the case of one breakpoint. This shape places the minima in narrow valleys which can dramatically increase the time required for gradient based algorithms to find a minimum. In addition, the slope of F becomes infinite as the value of F approaches zero, again due to the shape of the entropy function. (The data set used to generate this surface is marked above the graph using "o" and "x" to represent the two different classes.) G is therefore concave up as shown in Figure 2.12. This places the minima in wide basins that are much easier to find, as shown in Figure 2.14. The cusp in G can be smoothed out, as demonstrated by the graphs of G_s in Figure 2.12 and F in Figure 2.15, but experiments have shown that this is not necessary because gradient descent always moves away from any sharp edges caused by the cusp, so the lack of differentiability does not cause problems. In addition, the cusp does not introduce any infinite slopes. Briefly stated, G causes problems in regions that are avoided by minimization algorithms, while H presents difficulties in the region directly surrounding the minimum.

It is important to note that the graphs in Figures 2.13 through 2.15 were generated by choosing δ to be wider than the distance between adjacent examples. If this is not done, F will have the shape of a staircase function as illustrated in Figure 2.16. Even though this surface is differentiable at every point, it is perfectly flat between examples, so gradient descent will not work.

In the typical case of overlapping class distributions, F also has many local minima when δ is of the order of the distance between adjacent examples. This is illustrated in Figure 2.17. Setting δ much larger than the distance between adjacent examples can eliminate these spurious minima without changing the position of the global minimum, as shown in Figure 2.18. This case also demonstrates another problem with using the entropy, H, instead of G. As Figure 2.19 shows, the minimum of F is not at the Bayes-optimal location because $H(y)$ rises so rapidly near y=0 that when the breakpoint starts at b=25 in Figure 2.19 and moves to the right, the decrease in the contribution to F from the bin $(b, 100]$ is swamped by the increase in the contribution from the bin $[0, b)$. (G_s does not have this problem because, like G, it has zero slope at y=0 and y=1.)

As can be seen by comparing Figures 2.14 and 2.18, the less separable the classes are, the shallower the valleys will be. In all cases, F attains its maximum value of one when the breakpoints are far from all the data points. When the classes are fully separable, F achieves the value of zero when the breakpoints are positioned optimally. If there is no correlation between the data variables and the class variable, then the data points from each class will be uniformly distributed, and F will be flat.

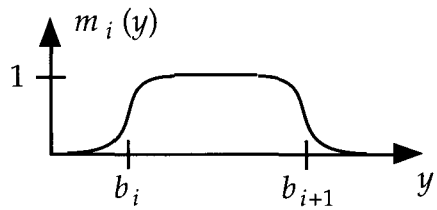


Figure 2.11: The shape of the interval membership function

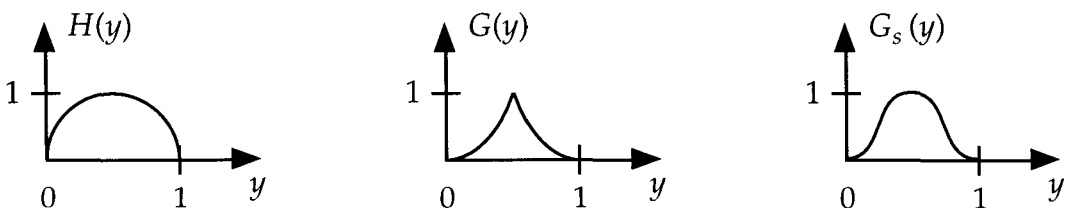


Figure 2.12: Entropy (H) and its concave up replacements (G, G_s) for the case of two classes ($y = q_{i,1}$, $q_{i,2} = 1 - q_{i,1}$)

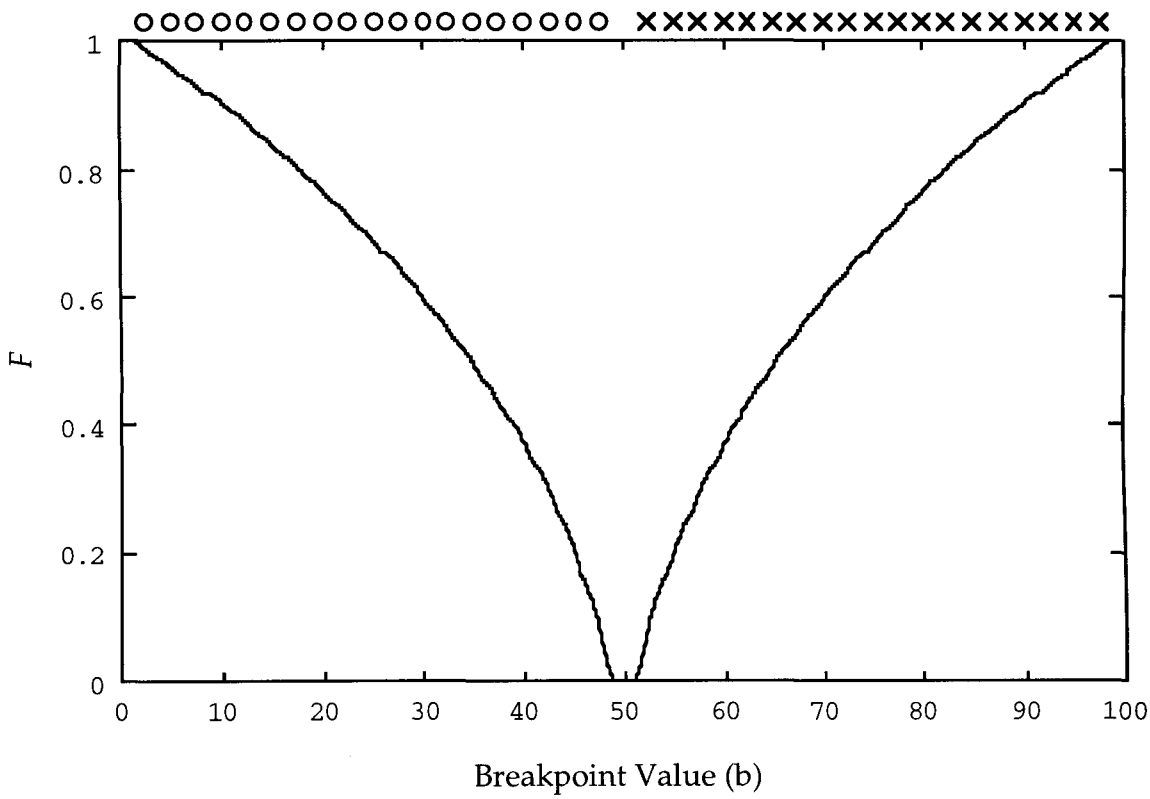


Figure 2.13: Sample MEP surface using H

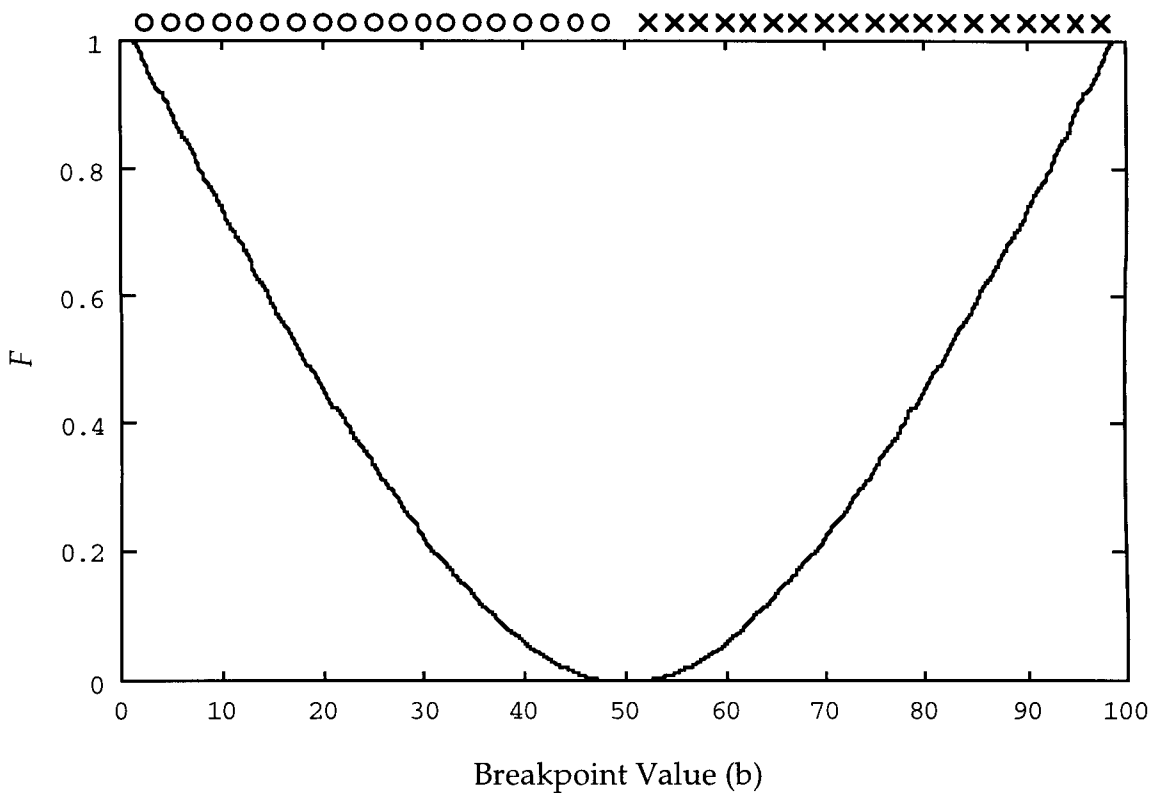


Figure 2.14: Sample MEP surface using G

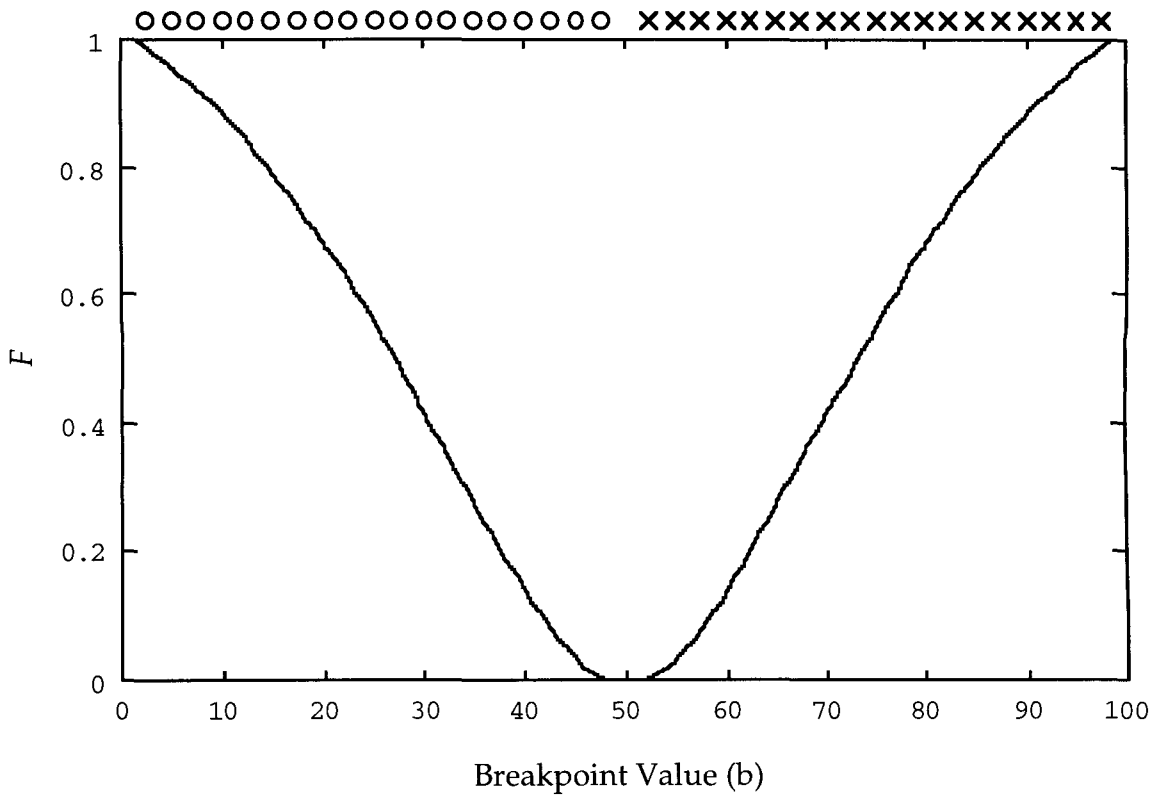


Figure 2.15: Sample MEP surface using G_s

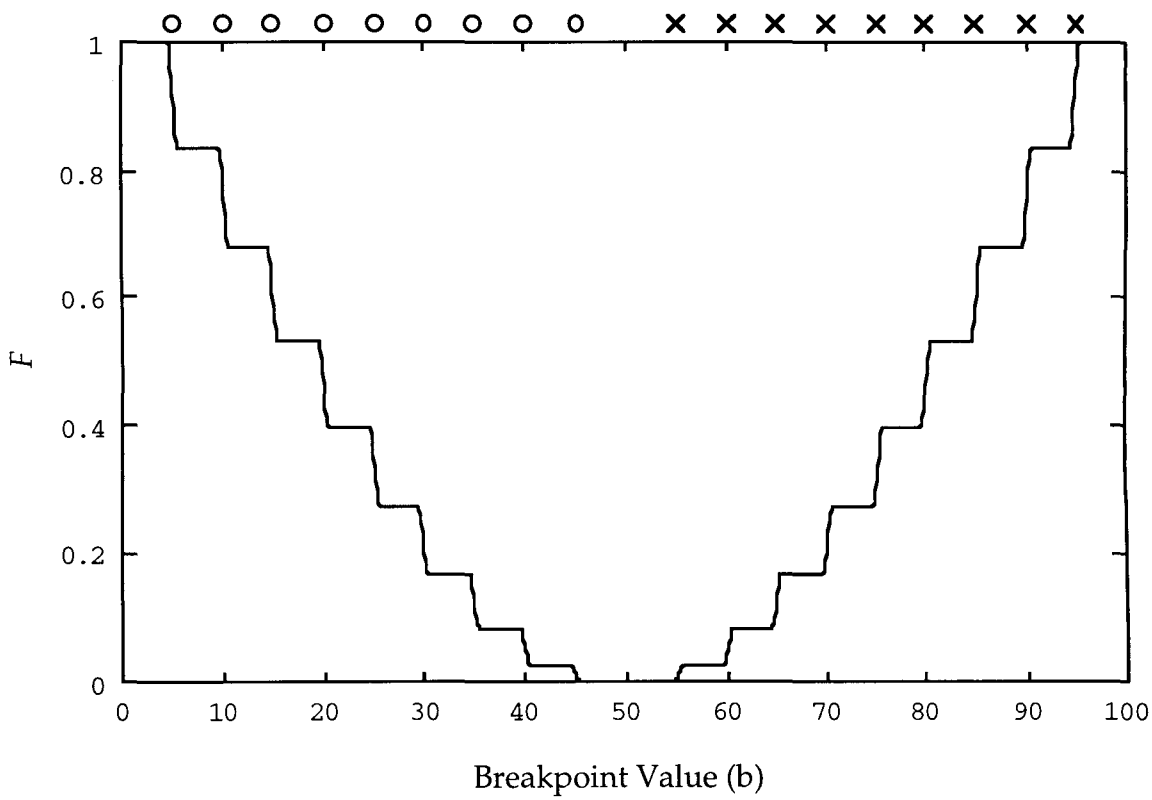


Figure 2.16: Sample MEP surface using G with δ much smaller than the separation between adjacent examples

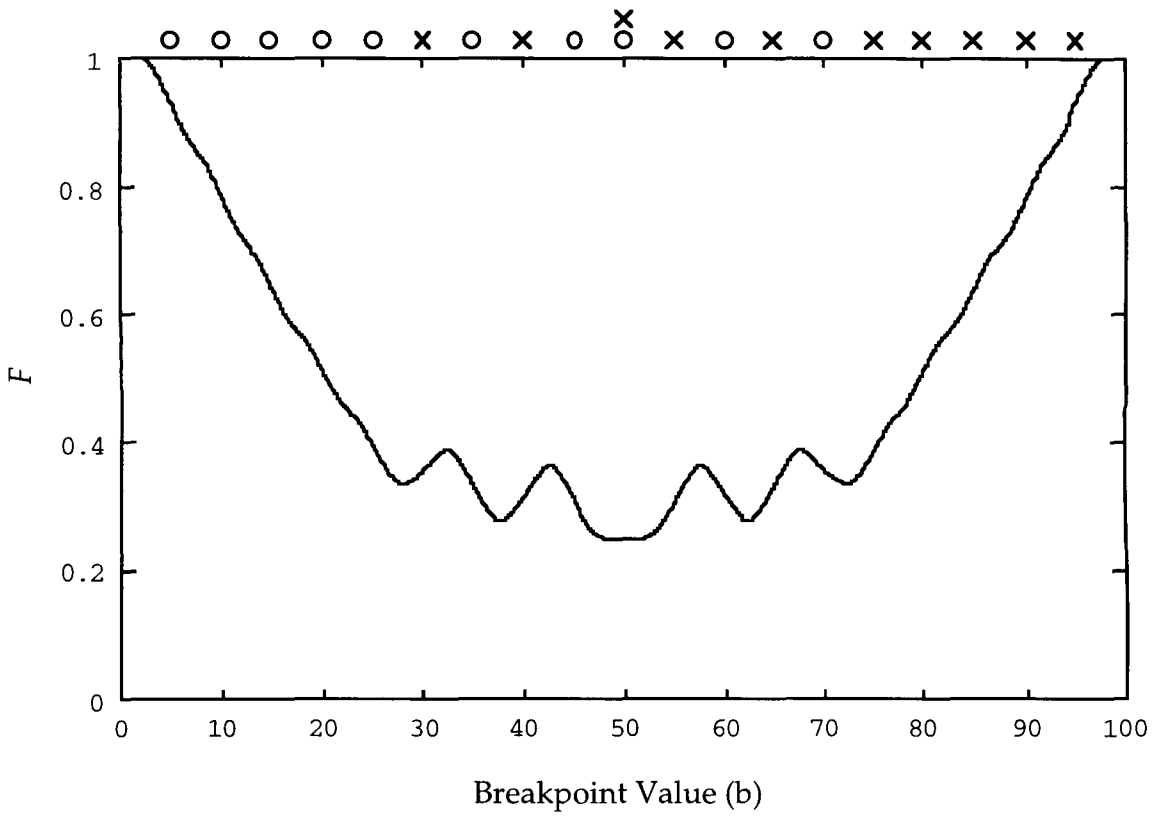


Figure 2.17: Sample MEP surface using G with δ of the order of the separation between adjacent examples when perfect separation is not possible

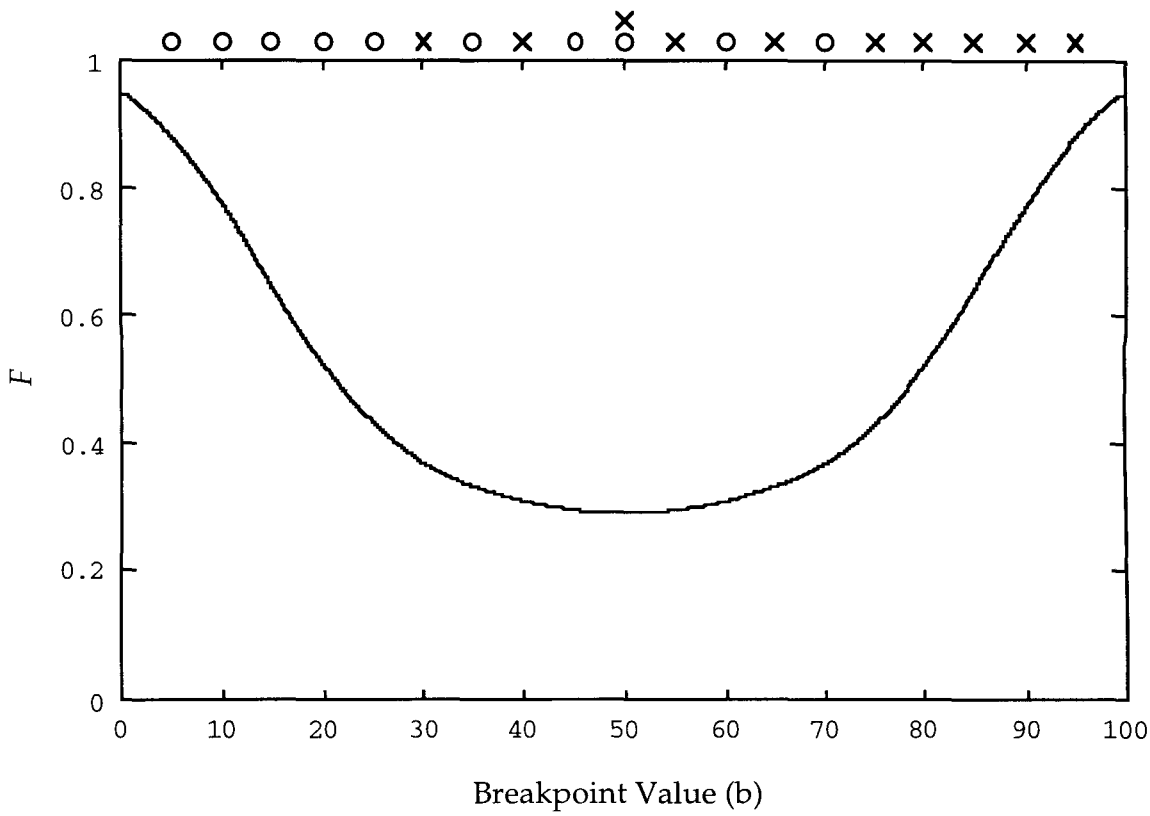


Figure 2.18: Sample MEP surface using G with δ much larger than the separation between adjacent examples when perfect separation is not possible

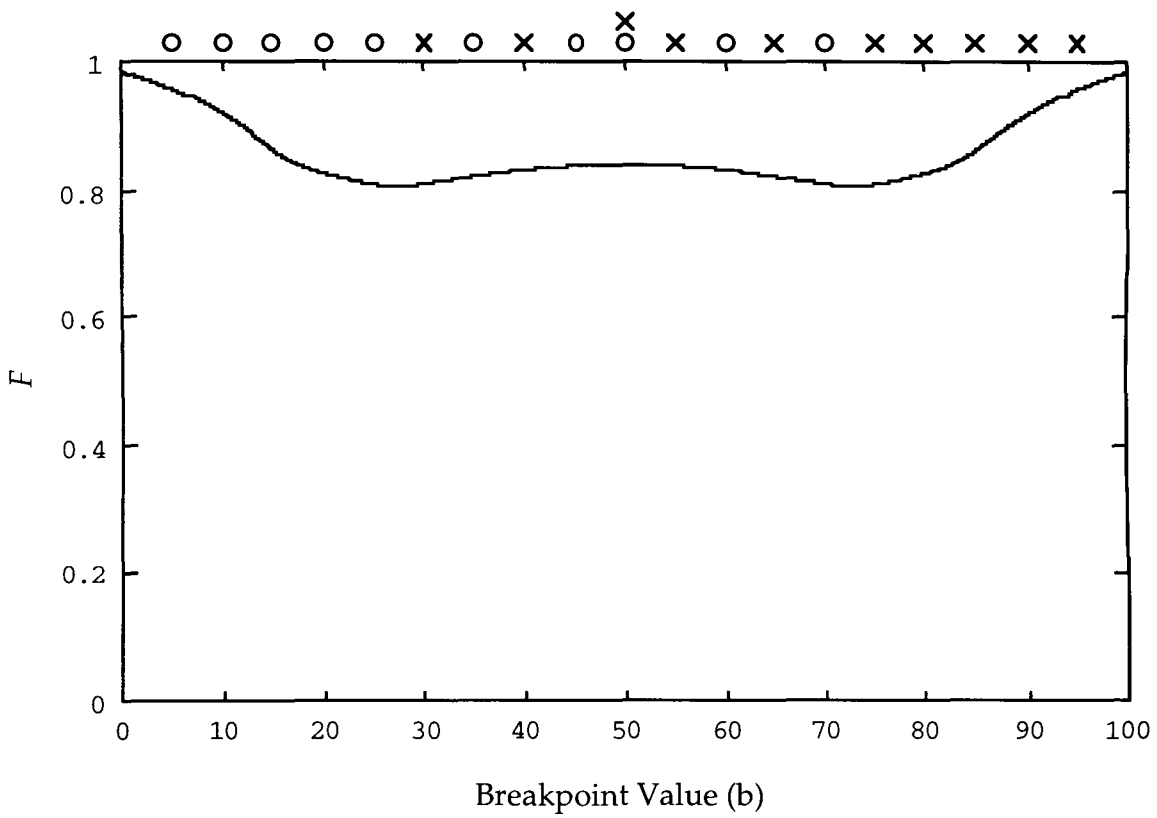


Figure 2.19: Sample MEP surface using H with δ much larger than the separation between adjacent examples when perfect separation is not possible

Appendix 2-A: The analytical expression for the computational complexity of the improved MDL algorithm

The amount of time required for the MDL algorithm to finish can be derived directly from the pseudocode presented in Section 2.1.3:

T_{MDL} =	Time required to:
T_{gen}	generate initial set of rules
$+ N_{gen} N_{train} T_{LHS}$	evaluate every LHS on every example
$+ N_{train} T_{DL}$	initialize LIST
$+ \sum_{i=1}^{N_{MDL}}$	repeat until description length stops decreasing:
$\sum_{j=1}^{N_{gen}-i+1}$	for each rule that has not already been added to the final set of rules:
$[f_{R,j} N_{train} T_{add}$	compute $\Delta_{max,MDL}$
$+ f_{\Delta,j} ($	if the value is large enough:
$f_{R,j} N_{train} (T_C(i) + T_{DL})$	re-classify examples and compute $-\log_2(p(x_{i,correct}))$
$+ N_{train} T_{add})]$	compute description length

Most of these symbols are defined in Section 2.1.6 where the formula for the upper bound on T_{MDL} is presented. The parameter $f_{R,j}$ is the fraction of the examples that are processed as a result of optimizing to operate only on the examples to which the candidate rule applies, while $f_{\Delta,j}$ is the fraction of the rules that are processed as a result of using the bound $\Delta_{max,MDL}$ presented in Section 2.1.2. Both of these parameters lie in the interval (0,1]. The function $T_C(n)$ computes the amount of time required to classify *one training* ex-

ample when there are n rules in the classification network of Figure 2.1. This function is analyzed in Appendix 2-D. The parameters T_{add} and T_{DL} represent the amounts of time required to perform one addition and compute one term in the summation in the formula for L_1 presented in Section 2.1.1, respectively. T_{LHS} represents the average amount of time required to evaluate a rule's LHS. Since the LHS can in principle be arbitrarily complex, a simple expression for T_{LHS} cannot be given. However, if the LHS is a conjunction of simple conditions, as discussed in Section 2.3, then T_{LHS} is merely the time required to check the values of one or more data variables.

Appendix 2-B: The analytical expression for the computational complexity of the new CV-SD algorithm

The amount of time required for the CV-SD algorithm to finish can be derived directly from the pseudocode presented in Section 2.1.4:

$T_{CV\text{-}SD}$ =	Time required to:
$(N_{train} + 1)\Big\{$	for each call to DESCENT:
T_{gen}	generate initial set of rules
$+ N_{gen}\,(N_{train} - 1)\,T_{LHS}$	evaluate every LHS on every example
$+ (N_{train} - 1)\,T_{cost}$	initialize LIST
$+ \sum_{i=1}^{N_{max}}$	repeat N_{max} times:
$\sum_{j=1}^{N_{gen}-i+1}$	for each rule that has not already been added to the final set of rules:
$[\,f_{R,j}\,N_{train}\,T_{add}$	compute $\Delta_{max,CV\text{-}SD}$
$+\ f_{\Delta,j}\,($	if value is large enough:
$f_{R,j}\,N_{train}\,(T_C(i) + T_{cost})$	re-classify examples and evaluate cost function
$+\ N_{train}\,T_{add})]\Big\}$	compute total cost
$+ N_{train}\,N_{max}\,T_{add}$	add list CV1 to list CV (inside outer loop)

Most of these symbols are defined in Section 2.1.6 where the formula for the upper bound on $T_{CV\text{-}SD}$ is presented. The parameter $f_{R,j}$ is the fraction of the examples that are processed as a result of optimizing to operate only on the examples to which the candidate rule applies, while $f_{\Delta,j}$ is the fraction of

the rules that are processed as a result of using the bound $\Delta_{max,CV\text{-}SD}$ presented in Section 2.1.2. Both of these parameters lie in the interval (0,1]. The function $T_C(n)$ computes the amount of time required to classify one training example when there are n rules in the classification network of Figure 2.1. This function is analyzed in Appendix 2-D. The parameters T_{add} and T_{cost} represent the amounts of time required to perform one addition and evaluate the cost function introduced in Section 2.1.1, respectively. Since the cost function is specified by the user and may therefore be arbitrarily complex, an expression for T_{cost} cannot be given. However, if the cost function is simply a matrix of constants specifying the cost of each possible mistake, as it was for the experiments presented in Chapter 3, then T_{cost} is the time required to look up a value in memory. T_{LHS} represents the average amount of time required to evaluate a rule's LHS. Since the LHS can in principle be arbitrarily complex, a simple expression for T_{LHS} cannot be given. However, if the LHS is a conjunction of simple conditions, as discussed in Section 2.3, then T_{LHS} is simply the time required to check the values of one or more data variables.

Appendix 2-C: The analytical expression for the computational complexity of the new CV-J algorithm

The amount of time required for the CV-J algorithm to finish can be derived directly from the pseudocode presented in Section 2.1.5:

$T_{CV\text{-}J}$ =	Time required to:
N_{train} (	for each training example:
T_{gen}	generate initial set of rules
$+ \sum_{i=1}^{N_{max}} (T_C(i) + T_{cost})$	repeat N_{max} times: add next rule and classify the example
$+ N_{max}\, T_{add})$	add list CV1 to list CV
$+ T_{gen}$	generate the final list of rules

Most of these symbols are defined in Section 2.1.6 where the formula for the upper bound on $T_{CV\text{-}J}$ is presented. The function $T_C(n)$ computes the amount of time required to classify one training example when there are n rules in the classification network of Figure 2.1. This function is analyzed in Appendix 2-D. The parameters T_{add} and T_{cost} represent the amounts of time required to perform one addition and evaluate the cost function introduced in Section 2.1.1, respectively. Since the cost function is specified by the user and may therefore be arbitrarily complex, an expression for T_{cost} cannot be given. However, if the cost function is simply a matrix of constants specifying the cost of each possible mistake, as it was for the experiments presented in Chapter 3, then T_{cost} is the time required to look up a value in memory.

Appendix 2-D: The analytical expression for the time required to classify one example using the rule-based classification network

The amount of time required for the rule-based classification network to classify one example can be derived from the procedure in Section 2.1:

	Time required to:
$T_C(n) = n\, T_{LHS}$	evaluate the LHS of each rule
$+ N_{RHS}$ (	for each output:
$f_R\, n\, T_{add}$	add the values from the rules that were satisfied
$+ T_{exp}$)	exponentiate the result
$+ N_{RHS}\, (T_{div} + T_{add})$	normalize the results

Here, n is the number of rules in the network, N_{RHS} is the number of outputs from the network, and T_{LHS} represents the average amount of time required to evaluate a rule's LHS. Since the LHS can in principle be arbitrarily complex, a simple expression for T_{LHS} cannot be given. However, if the LHS is a conjunction of simple conditions, as discussed in Section 2.3, then T_{LHS} is simply the time required to check the values of one or more data variables. It is important to note that the pseudocode presented in Sections 2.1.3 and 2.1.4 assumes that the matrix of Boolean values resulting from evaluating every LHS for every example has been pre-calculated. In this case, T_{LHS} is zero. The parameter f_R is the fraction of rules whose LHS's are satisifed. All that can be said about this value is that it lies in the interval [0,1]. The symbols T_{exp}, T_{div}, and T_{add} represent the amounts of time required to exponentiate, divide, and add two numbers, respectively.

Chapter 3

Experimental comparison of the classification algorithms considered for use in Poirot

3.0. The data used to evaluate the classifiers

This chapter presents the results of experiments designed to compare the performances of the three different versions of ITRule presented in Chapter 2, two variants of the Naive Bayes classifier (Duda and Hart, 1973), several neural networks (Haykin, 1999), the decision tree algorithm, CART (Breiman et al., 1984), and a support vector machine (Vapnik, 1995), in order to determine which classifier is best suited for use in Poirot. The data sets utilized in these experiments were generated from a well known collection of 21,578 articles published by the Reuters News Service during 1987 (Reuters-21578, 1987). Each article has been manually classified as belonging to one or more of 120 different topics. This collection of articles is large and diverse. Although they are not actual web pages, they provide good substitutes.

In the experiments described here, each training set was generated by first picking a topic and then randomly choosing N articles about the chosen topic and N articles about other topics. The corresponding test set consisted of the remaining articles about the chosen topic and an equal number of randomly chosen articles concerning other topics in order to ensure that the average accuracy one could expect to get from pure guessing was 50%. For each training set and its corresponding test set, sampling was done without replacement so no article was used more than once. When generating data sets, only topics containing a reasonably large number of articles were used, thereby ensuring that each test set contained at least 100 examples so that the performance of each classifier was evaluated accurately.

3.1. The method used to calculate the probabilities used by the classifiers

Since there is only a finite number of training examples, only approximate statistics can be calculated. ITRule and Naive Bayes therefore use the Laplacian formula for estimating probabilities (Kohavi et al., 1997). With this method, the probability that a relevant article will contain a particular word is given by:

$$p(w \mid r) = \frac{N_{w \cap r} + 1}{N_r + 2}$$

Here, $N_{w \cap r}$ is the number of relevant documents that contain the word of interest, and N_r is the total number of relevant documents. When N_r is zero, $N_{w \cap r}$ must also be zero, so $p(w \mid r) = 1/2$. This corresponds to the maximum entropy assumption which is considered to be the best guess when no other information is available. As N_r increases, the information about the frequency of occurrence of the word becomes more and more accurate. This eventually overwhelms the contribution of the constants, so $p(w \mid r)$ asymptotically approaches $N_{w \cap r}/N_r$. The same principle is used for computing all the other probabilities that are required.

3.2. Misclassification costs

Since most users probably prefer to skim and reject some irrelevant articles rather than risk missing an interesting article, the cost[1] of misclassifying an article as irrelevant was set to twice the cost of misclassifying an article as relevant in all experiments except with the neural networks, where the usual

[1] The term "cost" is used because it is analogous to situations such as finance or medicine where all decisions can be measured in monetary units. When dealing with web pages, the cost is not related to anything concrete, but is simply a subjective measure of the relative severity of the different types of errors.

mean squared error was used. As demonstrated by the results presented in Section 3.5.3, this turned out to be primarily an issue of principle because varying the cost did not significantly affect the performance of any of the algorithms.

3.3. Reducing the run time by prefiltering the list of words obtained from the training articles

The complete list of unique words for a set of training articles usually numbers in the thousands. When phrases are included, this may rise to tens of thousands. Many of these potential input features, such as articles and conjunctions, provide no information. By discarding these useless words and phrases before the classifier construction algorithm (see Section 3.4) is run, one can dramatically increase the speed of the computations without a noticeable loss of accuracy (see Section 3.5.4).

In order to decide which words and phrases to discard, a ranking function is used. This function is evaluated separately for each word and phrase. After sorting them in descending order of the value of the ranking function, all but the top N words and phrases are discarded.

The most commonly used ranking function is the mutual information, $I(W;A)$, between the Boolean occurrence of each word, represented by the random variable W, and the Boolean relevance of each training article, represented by the random variable A. As illustrated in Figure 3.1, the value of this function is large for words that occur in many relevant articles but in only a few irrelevant articles, and also visa versa.

Using the value of $I(W;A)$ to rank the words does not produce the correct result in all cases. For instance, if there are an equal number of relevant and irrelevant training articles, and the word only occurs once in each relevant article but many times in each irrelevant article, then $I(W;A)$ is zero. This will cause the word to be ignored since zero is the lowest value that the mutual information can attain. However, even though $I(W;A)$ is zero in this case, the distribution of the word clearly indicates that articles containing the word are likely to be irrelevant since the word occurs much more often in irrelevant articles than in relevant ones. In order to handle this case correctly, a new ranking function dubbed the word imbalance, F_{WI}, has been developed:

$$F_{WI} = \left| \frac{N_r - N_{\bar{r}}}{N_r + N_{\bar{r}}} \right| \max(p(w \mid r), p(w \mid \bar{r}))$$

Here, r denotes relevant and $\bar{r}$ not relevant. The symbols N_r and $N_{\bar{r}}$ represent the average numbers of occurrences of the word in relevant and irrelevant articles, respectively. Similarly, $p(w \mid r)$ and $p(w \mid \bar{r})$ are the probabilities that the word occurs in relevant and irrelevant articles. The first factor in the formula provides a measure of the imbalance in the distribution of the word between the sets of relevant and irrelevant training articles. The denominator of this factor scales the value so that it lies in the interval [0, 1]. If the word occurs equally often in both sets, the value of the function is zero. The second factor ensures that the value of the function is also small for words that are very rare in both sets.[2]

[2] Without the absolute value bars, F_{WI} would provide additional information. A positive value would indicate that an article containing the word is relevant, while a negative value would indicate that an article containing the word is irrelevant. This additional information is not needed by the classifier construction algorithm, however. It only requires an unbalanced probability distribution, not a probability distribution that specifically indicates "relevant."

Figures 3.1 through 3.4 provide examples of the difference between the mutual information and the word imbalance. The graphs were generated using one hundred relevant articles and one hundred irrelevant articles. Figure 3.1 shows $I(W;A)$ as a function of the fraction of relevant and irrelevant documents that each contain ten occurrences of the word under consideration. Figure 3.2 plots the word imbalance under the same conditions. Note that both functions have the same shape in this case. The only difference is in the curvature. This is demonstrated by the graph in Figure 3.3 which plots the difference between the two functions. The difference is zero down the middle and at all four corners.

The situation is very different if each relevant article that contains the word at all contains only one occurrence rather than ten. The graph of the mutual information does not change. The word imbalance, on the other hand, changes dramatically, as shown in Figure 3.4. The initial example discussed above occurs when all the articles contain the word, i.e., at (1, 1) in the relevant-irrelevant plane of the graph. The word imbalance gives a value of $|(1-10)/(1+10)| \approx 0.82$, in stark contrast to the mutual information which is zero. The word imbalance is zero along the line between (0, 0) and (1, 0.1) in the relevant-irrelevant plane, as opposed to the mutual information which is zero along the line between (0, 0) and (1, 1). The displacement of the word imbalance's "zero line" demonstrates this function's sensitivity to the number of occurrences of the word, rather than only the Boolean occurrence of the word, as is the case with the mutual information.

Considering the success of the J-measure used by ITRule, one might be tempted to use the Kullback-Liebler distance (Cover and Thomas, 1991) between $p(w\,|r)$ and $p(w\,|\bar{r})$, denoted by $D(p(w\,|r)\,||\,p(w\,|\bar{r}))$. This does not work, however, because this distance measure is unbounded. Even if the word occurs in only a single, relevant article, as illustrated in Table 3.1, $D(p(w\,|r)\,||\,p(w\,|\bar{r}))$ will be infinite. Both the mutual information and word imbalance give a very low score in this case. ITRule will also ignore the word because it considers the J-measure between each column and the prior probability distribution, not the distance between the rows. In Table 3.1, the left column produces a rule with very low J-measure because the probability of the rule's LHS is very low, and the right column does the same because the distribution is very close to the prior distribution.

In practice, picking the top 100 words with either mutual information or word imbalance yields mostly the same words, merely in a different order. As shown in Section 3.5.4, discarding all but these top 100 words does not degrade performance. The relevancy score which is common in the information retrieval literature was not tried because Wiener et al. (1995) state that it produces results very similar to those obtained when using mutual information.

article \ word	occurs	does not occur
relevant	$1/N$	$0.5 - 1/N$
not relevant	0	0.5

Table 3.1: Joint probabilities if a word occurs in only a single, relevant training article, and there are equally many relevant and irrelevant articles. There are a total of N training articles, so Pr(word occurs $\cap$ article is relevant) = $1/N$.

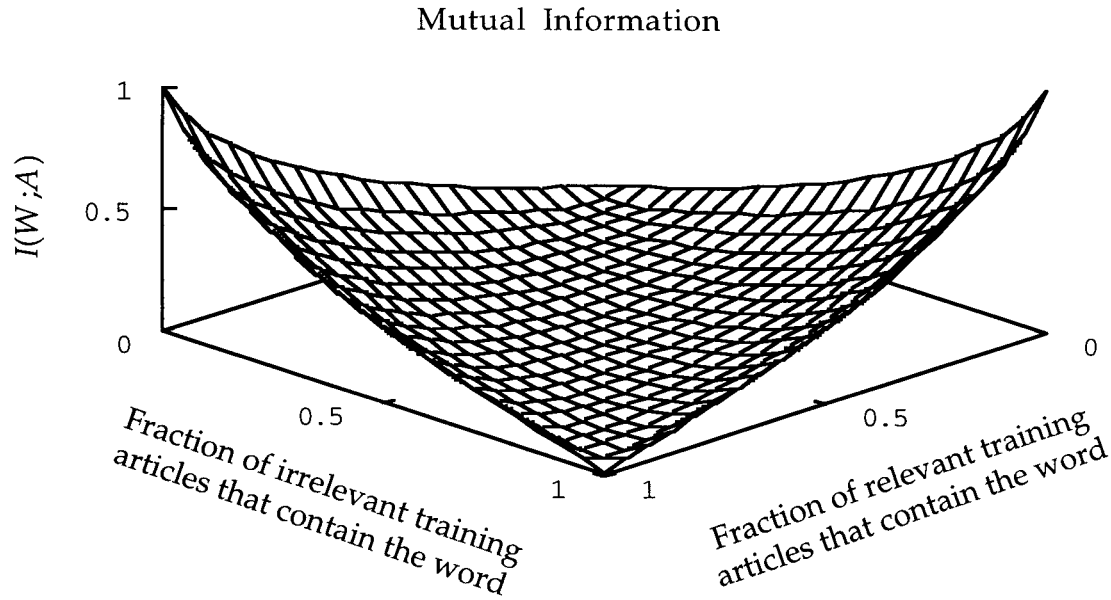


Figure 3.1: Plot of the mutual information, $I(W;A)$, as a function of the fraction of relevant and irrelevant training articles that contain a given word. The random variable W represents the Boolean occurrence of a word, and the random variable A represents the Boolean relevance of a training article.

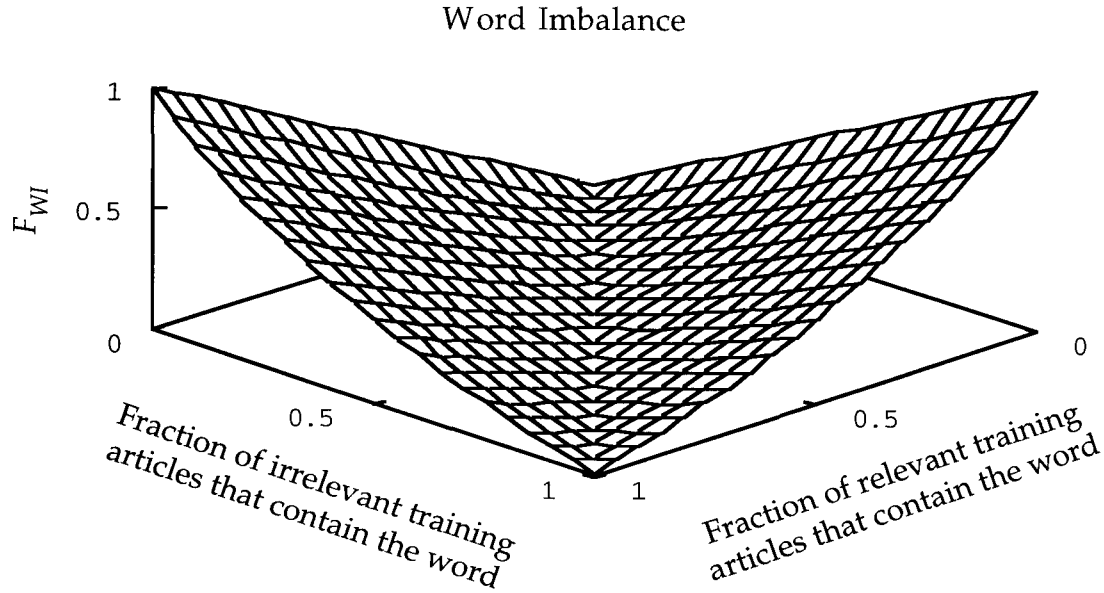


Figure 3.2: Plot of the word imbalance, F_{WI}, as a function of the fraction of relevant and irrelevant training articles that contain a given word. This plot was generated by using relevant and irrelevant articles that either did not contain the word or contained ten occurrences of the word.

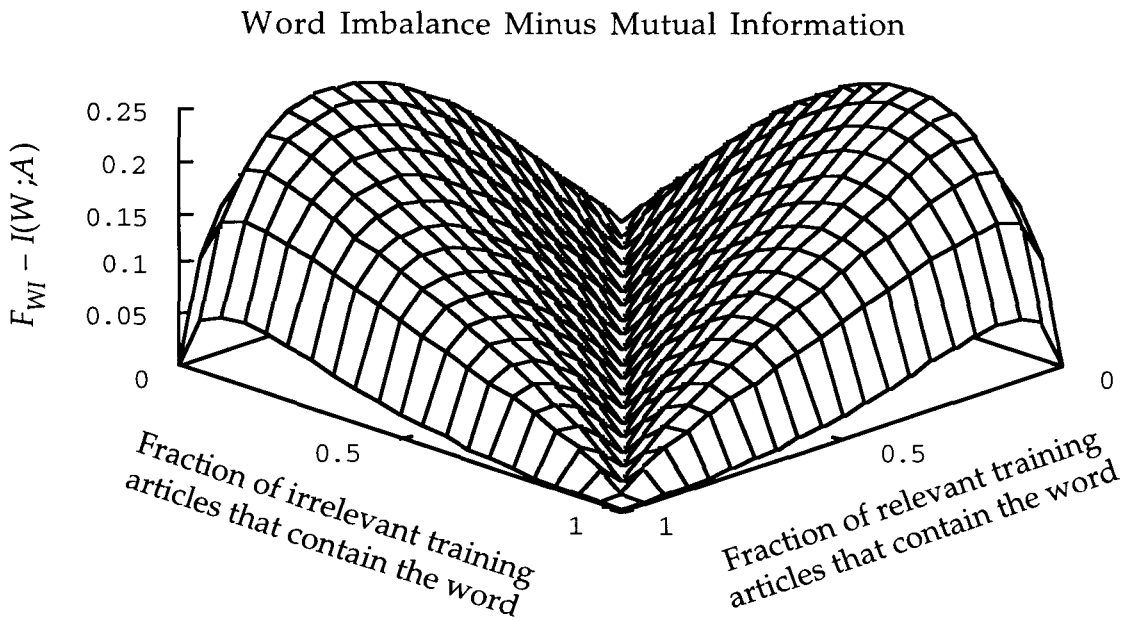


Figure 3.3: Plot of the difference between the word imbalance and the mutual information as a function of the fraction of relevant and irrelevant training articles that contain a given word. This plot was generated by using relevant and irrelevant articles that either did not contain the word or contained ten occurrences of the word.

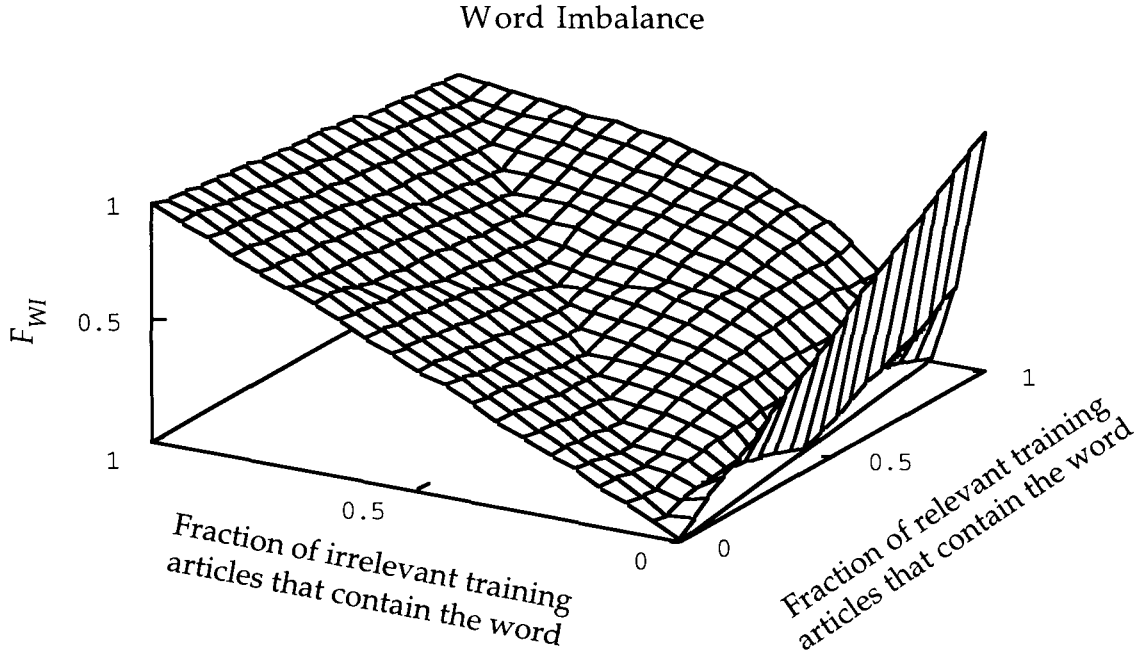


Figure 3.4: Plot of the word imbalance, F_{WI}, as a function of the fraction of relevant and irrelevant training articles that contain a given word. This plot was generated by using relevant articles that either did not contain the word or contained one occurrence of the word and irrelevant articles that, if they contained the word, contained ten occurrences of the word. Note that the "zero line" from (0, 0) to (1, 0.1) is not clearly visible because the sampling grid is too coarse. The graph is zero at the three points (0, 0), (0.5, 0.05), and (1, 0.1) because these lie on both the zero line and the sampling grid.

3.4. Description of the classification algorithms

3.4.1. ITRule

The three algorithms, MDL, CV-SD, and CV-J, described in Chapter 2 were all tested. The input variables were Boolean values denoting the presence or absence of words or phrases. Only first order rules were considered for MDL and CV-SD because the run time was excessive even in this case. For CV-J, inclusion of second order rules produced significantly lower accuracy, so only the results for first order rules are presented here.

In the case of CV-SD, the cost function was a simple matrix of constants. The cost of correctly classifying an example was zero. The cost of misclassifying an example as relevant was one, while the cost of misclassifying an example as irrelevant was two, as discussed in Section 3.2.

As the results below show, the CV-J algorithm works very well. However, it should be noted that when only one word is needed to distinguish between relevant and irrelevant articles, and the data set has an equal number of positive and negative examples, then the algorithm will fail. The reason is that every cross validation cycle will use M positive and M–1 negative examples or visa versa. Thus, when a single word is a perfect discriminator, ITRule only needs to pick a single rule that moves the predicted probability in the direction opposite the initial slant in the prior probability distribution. For instance, with M positive and M–1 negative examples, the default decision is "relevant," and one only needs the rule "if word is not present, then $p(not\ relevant)\approx 1$" to always get the correct result. In this case, this happens to be the rule with the highest J-measure since it is further from the

prior distribution, so the result of cross validation is "pick the single rule with highest J-measure." This does not work on the complete set of M positive and M negative examples, however, because it is perfectly balanced. Since the prior distribution is not slanted in either direction, one needs both rules "if word is present, then ..." and "if word is not present, then ..." in order to make the correct prediction in all cases. Thus, in order to work in this special case, the algorithm presented in Section 2.1.5 was modified so it cannot pick fewer than two rules.[3]

3.4.2. Naive Bayes

The Naive Bayes classifier produces probability estimates under the assumption that the input variables are conditionally independent when given the class (Domingos and Pazzani, 1996). This is identical to the case when ITRule is restricted to use only first order rules except that when Naive Bayes uses an input variable, it automatically uses both rules for that variable, i.e., "if x=T, then ..." and "if x=F, then ..." ITRule, on the other hand, can choose each one separately.

The input variables for this classifier were Boolean values denoting the presence or absence of words or phrases. Two different algorithms for picking the words to be used were tested, namely NB-96 and NB-CV. NB-96 is the algorithm used by Pazzani and Billsus (1997). It simply picks the 96 words and phrases with the highest mutual information between the Boolean occurrence of the word or phrase and the Boolean value indicating the relevance of the article to the topic under consideration. While many have studied the

[3] It should be noted that when the algorithm is run on a data set where the number of positive and negative examples is not equal, it correctly picks two rules even without this adjustment.

effects of varying the number of words that are used (Joachims, 1996; Mladenic, 1996; Pazzani and Billsus, 1997; McCallum and Nigam, 1998), nobody seems to have considered the problem of determining the optimal number of words. In the course of the research presented in this thesis, the new NB-CV algorithm was developed to do just this. It sorts the words and phrases in decreasing order of the word imbalance and uses cross validation to pick the top N. This is fundamentally the same algorithm as ITRule CV-J, except that the ranking function is different.

3.4.3. Neural networks

The experiments were performed with the basic, feed forward neural network architecture because it is the most widely used. Since even this simple configuration has a dizzying number of adjustable parameters, any attempt to optimize all these settings would alone take years of computer time. The network size chosen by Pazzani and Billsus (1997) was therefore used, namely 96 input units (the same features used by NB-96 above) and twelve hidden units. Both the hidden layer and the output layer used the hyperbolic tangent function whose range is [–1, +1]. In order to avoid forcing the units into saturation, the training data used +0.9 to indicate "relevant" and –0.9 to indicate "not relevant." (During testing, positive output values were considered to be "relevant," and negative output values were considered "not relevant.") The initial weights were randomly chosen in the interval [–0.001, +0.001], and plain gradient descent was used with a learning rate of 0.1 and a momentum of 0.9. Training was stopped after 5000 iterations. This compromise allowed the training error to decrease to within a few percent of the ab-

solute minimum without using excessive amounts of computer time. In order to compensate for the tendency to find only local minima instead of the global minimum, ten independent trials were conducted on each data set, and only the best performance was recorded.

Two different types of input features were tested. In the first case (NN-Boolean), the input variables were Boolean values denoting the presence or absence of words or phrases. In the second case (NN-Numeric), the input variables were numbers representing the frequency of occurrence of each word or phrase, i.e.,

$$\frac{\textit{number of occurrences of the word}}{\textit{number of words in the article}}$$

This approach often produces very small input values. However, these values are guaranteed to lie in the interval [0, 1], so the method does not suffer from normalization problems if a test article is longer than any of the training articles. Scaling these values separately for each article so they lay in the interval [–1, +1] was also tried, but the performance was significantly worse, so these results are not reported.

In addition, to compensate for the fixed number of hidden units, the effect of using a weight decay factor of 0.995 was investigated in both cases, i.e., NN-Boolean-Decay and NN-Numeric-Decay. The results of all these trials are presented in Section 3.5.

3.4.4. Classification and Regression Trees (CART)

CART (Breiman et al., 1984) builds a decision tree from the training data and then uses cross validation to prune the tree to combat the problems of overfitting and sensitivity to noise. The tests presented here used the implementation of CART provided by the "IND" package (Buntine, 1992). Since CART can handle both discrete and continuous variables, the same two types of input variables were tried that were used in the neural network. For consistency with the neural network results, these two cases are denoted by CART-Boolean and CART-Numeric. Since IND does not accept more than 250 input variables and pushing this limit is not recommended (Buntine, 1992), the top 100 words ranked by word imbalance were used.

3.4.5. Support Vector Machines (SVM)

On the advice of Dr. Joshua Alspector (Alspector, 2001), only the linear support vector machine (Vapnik, 1995) with Boolean inputs was tested. This algorithm simply searches for the hyperplane that best separates the relevant and irrelevant training articles. The training data used +1 to indicate both "relevant article" and "word or phrase is present" and –1 to indicate the opposite. Preliminary experiments showed that using all the available words and phrases as features resulted in very poor accuracy, e.g., 50%, so the same cross validation algorithm that was used in NB-CV was used to pick the optimal number of input variables. The tests presented here used the implementation of SVM provided by the SVM^{light} package (Joachims, 1999). Since this software does not directly support any form of feature selection[4], a separate

[4] As with the Naive Bayes algorithm, nobody appears to have considered the problem of determining the optimal number of words.

program was created to implement the cross validation algorithm and run SVM^{light} for each set of words. Unfortunately, this method is very slow because it requires writing a separate data file for each set of words and invoking SVM^{light} from scratch on each data file. In addition, SVM^{light} often got stuck in an infinite loop when there were fewer than ten features. The cross validation algorithm was therefore restricted to considering only sets of words of size 10, 15, 20, 25, ..., 100.

3.5. Experimental comparison of classifier performances

Before presenting the test results, it is necessary to describe how the comparisons of the classifiers were performed. Since the ultimate goal of this research was to develop a system that would assist users in locating relevant web pages, the performance goal was somewhat flexible. A system will be considered useful even if its performance is not perfect. The performance measure used in all the experiments was the test accuracy, which is defined as the ratio of the number of testing examples that were classified correctly to the total number of testing examples.

As illustrated in Figure 3.5, the performance of the best algorithm, namely NB-CV, rarely dropped below 80% accuracy. This may not appear particularly good when compared with the ideal accuracy of 100%, but when one takes into account the time that the user saves by not having to study each irrelevant page manually, it is an impressive result. Moreover, misclassifying a web page as "relevant" is not a serious problem as long as it does not occur too often. Misclassifying a web page as "not relevant" can be more serious but is not usually critical because the user will probably discover the page

while surfing from other relevant pages.

The performance criterion is also flexible in the sense that users are unlikely to notice a difference in accuracy of a couple of percent. Thus, rigorous statistical tests were not used as the primary method of comparison.[5] Instead, one classifier was considered to be better than another if its average performance was at least several percentage points higher across all the data sets that were generated, and the standard deviation was small when compared to the mean. This approach was chosen because there will always be variations due to the randomness in the sampling process.[6]

The comparison between mean and standard deviation can easily be performed by plotting the difference in absolute accuracy across all the data sets. If the mean (shown by a horizontal dashed line) does not appear to be significant relative to the standard deviation (indicated by the scatter of the points), then the user is unlikely to notice the difference between the algorithms.

As an example of how to read the graphs, Figure 3.8 shows the accuracy difference between NB-CV and NB-96. When the average "Difference in Percent Accuracy" is 10.18 as indicated by the dashed line, this means that if NB-96 achieved 79.82% accuracy on average, then NB-CV achieved 90% accuracy on average.

[5] The "Difference of Two Proportions Test" (Dietterich, 1998; Yang and Liu, 1999) was nevertheless applied in all cases. The result from a single test example was assumed to be a Bernoulli random variable. Under this assumption, the total number of mistakes is a binomial random variable. If one assumes that the results from different classifiers are independent and uses the Gaussian approximation to the binomial distribution, one can obtain a unit Normal random variable and use a two-sided test to check the null hypothesis that two classifiers have the same performance. As discussed in Section 3.0, the data sets used in this thesis were generated independently. Thus, the unit Normal random variables generated from different data sets should be independent. By applying the Central Limit Theorem, the results from all the data sets were combined to obtain a single unit Normal random variable. The corresponding Z value is given in the figure caption for each graph.

[6] This randomness is actually a good way to simulate the way in which users accumulate web pages.

3.5.1. Comparing the effects of using single words and phrases

When only the occurrences of single words were considered, NB-CV, ITRule CV-J, and SVM performed equally well on average as can be seen from the graphs in Figures 3.6 and 3.7.[7] As shown in Figures 3.8 through 3.16, these three algorithms also performed better on average than NB-96, the two other ITRule algorithms, the neural networks, and CART.

The NN-Boolean-Decay and CART-Boolean algorithms performed second best. While one could argue that the difference in accuracy between these two algorithms and NB-CV is too small to be noticeable by users, there are additional issues which make NB-CV preferable. When compared with the neural network algorithms, the training process for NB-CV runs at least two orders of magnitude faster. In addition, since NB-CV explicitly chooses particular words, it is also much easier to determine which words should be sent to search engines. In Figure 3.15, the average difference between NB-CV and CART-Boolean is quite small only because there are so many data sets on which they both performed equally well. On the data sets where their performance differed, NB-CV was almost always better.

It is also interesting to note that NN-Numeric and CART-Numeric performed worse than NN-Boolean and CART-Boolean, respectively. (crf. Figures 3.11 through 3.16) Apparently, neither algorithm was able to utilize the word frequency information. Because of these results, word frequencies were not tried with ITRule or SVM.

[7] The fact that NB-CV and ITRule CV-J performed equally well might have been expected since they use the same basic algorithm.

When the occurrences of two and three word phrases were also included, ITRule CV-J, NB-CV, SVM, NN-Boolean-Decay, and CART-Boolean did sometimes make use of them, but on average, the performance only improved slightly, as can be seen from the graphs in Figures 3.17 through 3.21. Even though there were no significant improvements on average, it is important to note that the inclusion of phrases did not degrade the average performance, either. In fact, the only reason the average accuracies of NB-CV and ITRule CV-J did not change considerably is that the performance difference was zero on a large fraction of the data sets. On the data sets where the performance was different, the use of phrases almost always improved the accuracy. Thus, it is worthwhile to include phrases as potential input features.

It is important to note that when phrases were included, "stop words" were automatically removed before the articles were passed to a classifier. Stop words are words such as articles and conjunctions that are not useful for discriminating between relevant and irrelevant articles. When only single words were used as features, all the classifiers automatically ignored stop words because they had no discriminatory power. Removing them ahead of time was therefore unnecessary. However, when phrases were included, stop words often degraded the performance. As an example, if "merger" was ranked highly by the word imbalance, then "the merger," "merger with," and "the merger with" were also likely to be ranked highly. However, such phrases clearly provide no additional information. Thus, by removing stop words like "the" and "with" before passing the articles to the classifier, this problem was avoided. The list of stop words that were removed is given in

Appendix 3-A.[8]

3.5.2. Improvements in performance when more training examples are added

In order to model the scenario where the user evaluates only a few web pages before asking Poirot to make suggestions, the experiments discussed in Section 3.5.1 were performed with a small number of training articles. An experiment was also conducted in which a larger number of training articles was used for each topic. This models how well Poirot might perform after the user has used it for a while and accumulated a larger collection of web pages. The results are depicted in Figure 3.22. There is a clear and consistent improvement in performance when more training data is available.

A second experiment was performed to demonstrate this result in more detail for two particular topics. For each training set size, 20 independent data sets were generated and tested. Sampling without replacement was used within each data set, but sampling with replacement was used between data sets. Figures 3.23 and 3.24 show that as the training set size increased, the average performance improved, and the standard deviation due to the variation in the training data decreased, i.e., the classifier both worked better and was more reliable.

[8] The source of this list has unfortunately been lost, but there are several other lists available from `http://www-a2k.is.tokushima-u.ac.jp/member/kita/NLP/lex.html`

3.5.3. The sensitivity of classifier accuracy to variations in the misclassification costs

As discussed in Section 3.2, the cost of misclassifying an article as irrelevant does not have to be equal to the cost of misclassifying an article as relevant. The NB-CV and ITRule CV-J algorithms were tested with the relative cost of misclassifying an article as irrelevant set to 1, 2, and 4. Figures 3.25 and 3.26 demonstrate that ITRule CV-J has the same average accuracy for all three values of the cost. As shown in Figures 3.27 and 3.28, NB-CV performs marginally better when the relative cost is two. This indicates that these two algorithms are not particularly sensitive to the value of the relative cost.

3.5.4. The effect on classifier accuracy of prefiltering the list of words obtained from the training articles

The NB-CV and ITRule CV-J algorithms were tested on both unfiltered and filtered sets of words and phrases to determine the effect of filtering out all but the top 100 words via the word imbalance ranking function (see Section 3.3). The results are shown in Figures 3.29 and 3.30. It is clear from these plots that filtering the list of words did not have a detrimental effect on the accuracy. The issue does not apply to NB-96 or any of the neural networks because they always pick the top 96 words. (As discussed in Section 3.4.3, optimizing the number of words for the neural networks was not attempted because it would require too much CPU time.) The issue does not apply to SVM either, because, as discussed in Section 3.4.5, this algorithm only considered the top 100 words in order to avoid using excessive run time. The test could not be performed directly on CART because it cannot accept

more than 250 features, but tests did show that using the top 250 words produces exactly the same performance as using the top 100 words.

3.5.5. The effect on classifier accuracy of using stemmed words

Stemming is the process of combining the statistics for words that have the same root, such as "fly" and "flying." This improves the support of each resulting stem, thereby producing more accurate statistics. However, this is done at the cost of reducing the predictive power of some features due to averaging over the words that are combined into a single stem. The stemming algorithm that was used for this thesis is the one developed by Porter (Frakes and Baeza-Yates, 1992). When NB-CV and ITRule CV-J were tested on both unstemmed and stemmed sets of words, there were no significant differences in the average performance, as shown in Figures 3.31 and 3.32.

An additional experiment was conducted to reduce the variance due to the sampling process used to generate the data sets. For each of 11 different topics, 20 data sets of the same size were created. Sampling without replacement was used within each data set, but sampling with replacement was used between data sets. NB-CV was run on each data set for both unstemmed and stemmed words. The difference between the average performances using unstemmed and stemmed words was not significant, as shown in Figure 3.33. The error bar for each topic shows the estimated standard deviation calculated from the variation in performance over the 20 data sets for that topic.

3.5.6. Experimental comparison of the run times of the various algorithms

Figures 3.34 through 3.36 plot the measured run times of the NB-CV, CART-Boolean, and SVM algorithms as functions of the number of training examples, N_{train}. The run time of the NB-CV algorithm is clearly a linear function of N_{train}, while the run time of the CART-Boolean algorithm appears to be proportional to the square root of N_{train}. For the SVM algorithm, since cross validation had to be performed by a separate program and was therefore very slow as explained in Section 3.4.5, the time required for this computation was not measured. Instead, the run time was measured for the case when the top 100 words and phrases as ranked by the word imbalance were used as input variables. In this case, the run time of the SVM algorithm is clearly a linear function of N_{train}, as illustrated by Figure 3.36. If all the computations for SVM were done in one program, cross validation would, in the worst case, produce an additional factor of N_{train} in the computational complexity.[9]

The run times of the ITRule algorithms presented in Section 2.1.7 are not directly comparable to the run times plotted in Figures 3.34 through 3.36 because the current implementation of ITRule performs significant amounts of disk access[10] while the NB-CV, CART, and SVM algorithms operate entirely in RAM. The run times of the neural network algorithms were not measured because they are known to be slow to train, and the factor of ten penalty in the run time resulting from the strategy used to avoid local minima (see Section 3.4.3) merely exacerbates the problem.

[9] It is possible that clever optimizations could reduce or even eliminate this additional factor.

[10] The software was originally designed to handle very large data sets that would not fit in the available RAM.

3.6. Experimental comparison of classifier performances on a second data set

In order to verify that the superior performance of NB-CV was not due merely to some quirk of the Reuters-21578 collection[11], the NB-CV, NB-96, SVM, and CART-Boolean algorithms were also tested on the WebKB data set (Craven et al., 1998). This is a collection of 8,282 web pages collected from the computer science departments of several large universities. The web pages are grouped into seven categories: course information, department information, faculty home pages, research project home pages, staff home pages, student home pages, and "other." Even though these categories are far broader than any topic likely to be of interest to a Poirot user, the data set nevertheless provides a good test of whether or not NB-CV might be useful for other classification problems. For each topic except "other," 20 independent data sets were generated and tested. Sampling without replacement was used within each data set, but sampling with replacement was used between data sets. Each data set had 100 training pages and over 1000 testing pages. Both the training and testing data was evenly split between relevant and irrelevant pages. Table 3.2 shows the average accuracy of each algorithm and the corresponding standard deviation due to the variation in the training data. The accuracies of NB-CV, NB-96, and SVM all fall within one standard deviation of each other, while CART-Boolean performed significantly worse.

The effect on the performance of NB-CV of using only single words or including phrases without stop words was also tested. The results are shown in Table 3.3 and demonstrate the same result as in Section 3.5.1, namely that using phrases provides, on average, a slight improvement in performance.

[11] This does seem rather unlikely considering the wide variety of topics for which data sets were generated.

Finally, the effect of adding more training examples was tested, as in the second half of Section 3.5.2, and the same result was obtained, namely that increasing the number of training examples increased the average accuracy and decreased the standard deviation due to the variation in the training data. The details are shown in Table 3.4.

	course	dept.	faculty	project	staff	student
NB-CV	87.7%±2.5	91.6%±2.9	85.9%±2.3	77.2%±3.8	70.8%±5.1	81.3%±3.4
NB-96	87.6%±2.6	91.4%±2.2	86.7%±1.8	78.1%±2.2	72.6%±3.9	81.7%±2.6
SVM	89.1%±2.1	91.2%±2.8	87.2%±1.5	79.0%±2.3	71.5%±3.9	79.4%±4.0
CART	84.5%±2.4	82.4%±5.7	84.6%±1.9	69.6%±7.3	65.9%±8.7	70.1%±6.1

Table 3.2: Average accuracies on each WebKB category. The averages were taken over 20 independently generated data sets, each with 100 training examples and at least 1000 testing examples. The standard deviations are due to the variations in the training data.

	course	dept.	faculty	project	staff	student
words	85.4%±4.0	90.6%±2.5	85.1%±2.6	77.3%±4.0	71.1%±5.6	75.9%±6.2
phrases	87.7%±2.5	91.6%±2.9	85.9%±2.3	77.2%±3.8	70.8%±5.1	81.3%±3.4

Table 3.3: Average accuracies achieved by NB-CV on each WebKB category when only single words were used and when phrases without stop words were included. The averages were taken over 20 independently generated data sets, each with 100 training examples and at least 1000 testing examples. The standard deviations are due to the variations in the training data.

	course	dept.	faculty	project	staff	student
30	80.4%±7.1	84.2%±5.8	78.2%±8.6	66.9%±6.6	60.5%±6.9	70.8%±5.3
100	87.7%±2.5	91.6%±2.9	85.9%±2.3	77.2%±3.8	70.8%±5.1	81.3%±3.4

Table 3.4: Average accuracies achieved by NB-CV on each WebKB category when data sets with 30 and 100 training examples were used. The averages were taken over 20 independently generated data sets. The standard deviations are due to the variations in the training data.

3.7. Comparison with other published results

Many papers have been published during the past few years comparing the performance of various classifiers. Of the ones that used the Reuters-21578 data set, none used accuracy as the measure of performance. Instead, they used recall, precision, the precision/recall breakeven point (PRBEP) (Raghavan et al., 1989), or some other function. As explained by Schapire et al. (1998), these measures are all unsuitable for comparing text classification algorithms. However, since it is desirable to allow comparisons between the newly developed NB-CV algorithm and algorithms developed previously by other researchers, the PRBEP was computed for each data set that was generated from the Reuters-21578 and WebKB collections.

In order to compute the PRBEP for a particular data set, the NB-CV classifier was first constructed from the training articles. Next, the test articles were sorted in descending order of the output values that they produced when used in the classifier. The misclassification cost was then varied so that the threshold for predicting "relevant" was placed successively between each pair of adjacent test articles, and the precision and recall were computed at each position of this threshold. The PRBEP is defined as the value of the precision and recall when they are equal. Following the method used by Joachims (1998), the Probability of Relevance (PRR) algorithm (Raghavan et al., 1989) was used to interpolate the precision and recall curves to find the breakeven point when it did not occur at a threshold position. The details of this interpolation algorithm are explained in Appendix 3-B. Multiple breakeven points on a single data set were never encountered, but the algorithm was designed to report the lowest one if this situation had occurred.

	acq	corn	crude	earn	grain	interest	money-fx	ship	trade	wheat
NB-CV	92.5	100	100	92.5	100	95.0	95.0	100	91.4	100
Joachims	95.4	85.7	88.9	98.5	93.1	76.2	76.3	86.5	77.8	85.9
McCallum	93.9	70.4	83.9	98.0	81.7	62.6	67.4	86.1	71.7	65.4

Table 3.5: The highest precision/recall breakeven points achieved by the new NB-CV algorithm when using phrases without stop words, the non-linear support vector machines tested by Joachims (1998), and the multinomial Naive Bayes classifier developed by McCallum and Nigam (1998) on ten topics chosen from the Reuters-21578 collection. NB-CV performs much better on eight of the ten topics, and is only slightly inferior on the other two. Joachims also tested the well known TF-IDF (Rocchio, 1971), C4.5 (Quinlan, 1993), and k-nearest neighbors (Duda and Hart, 1973) algorithms. All three of these algorithms had lower precision/recall breakeven points than the non-linear support vector machines that he tested.

category	minimum	maximum
course	86.5	96.0
department	90.0	98.0
faculty	84.0	100.0
project	78.8	100.0
staff	58.3	96.0
student	88.0	96.0

Table 3.6: The minimum and maximum precision/recall breakeven points achieved by the NB-CV algorithm on each of the six categories in the WebKB collection. These values were obtained from the data sets with 100 training examples that were discussed in Section 3.6.

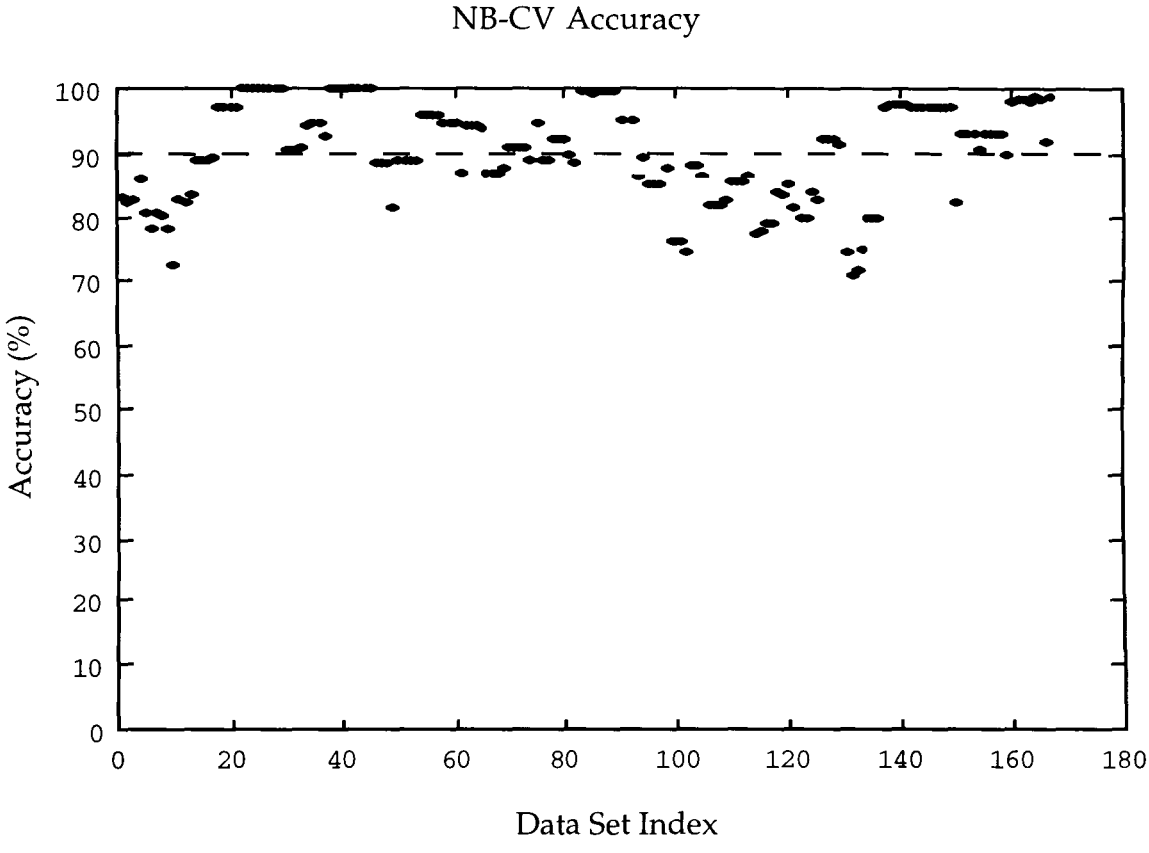


Figure 3.5: This graph shows the accuracy of the Naive Bayes using Cross Validation (NB-CV) algorithm. The experiments were performed with 167 data sets generated from the Reuters-21578 collection of news articles. Only single words were used to identify relevant articles. The accuracy is defined as the percentage of testing examples that were classified correctly. The average accuracy, indicated by the dashed line, was 90%, and the performance rarely dropped below 80%. Because of its excellent performance, the NB-CV algorithm was chosen for use in Poirot.

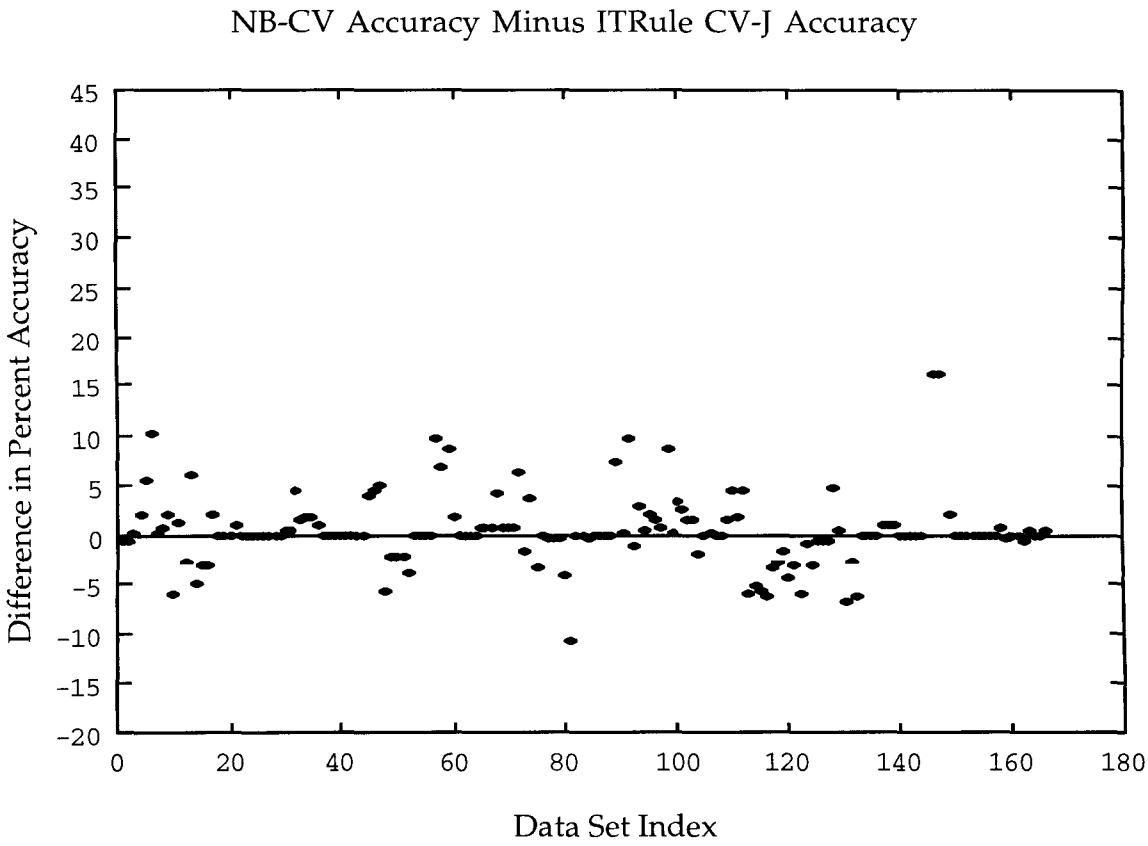


Figure 3.6: Difference in accuracy between Naive Bayes using Cross Validation (NB-CV) and Cross Validation using the J-measure (ITRule CV-J) when only single words were used to identify relevant articles. The average difference is 0.21%. This is not statistically significant (Z=1.1, see Section 3.5).

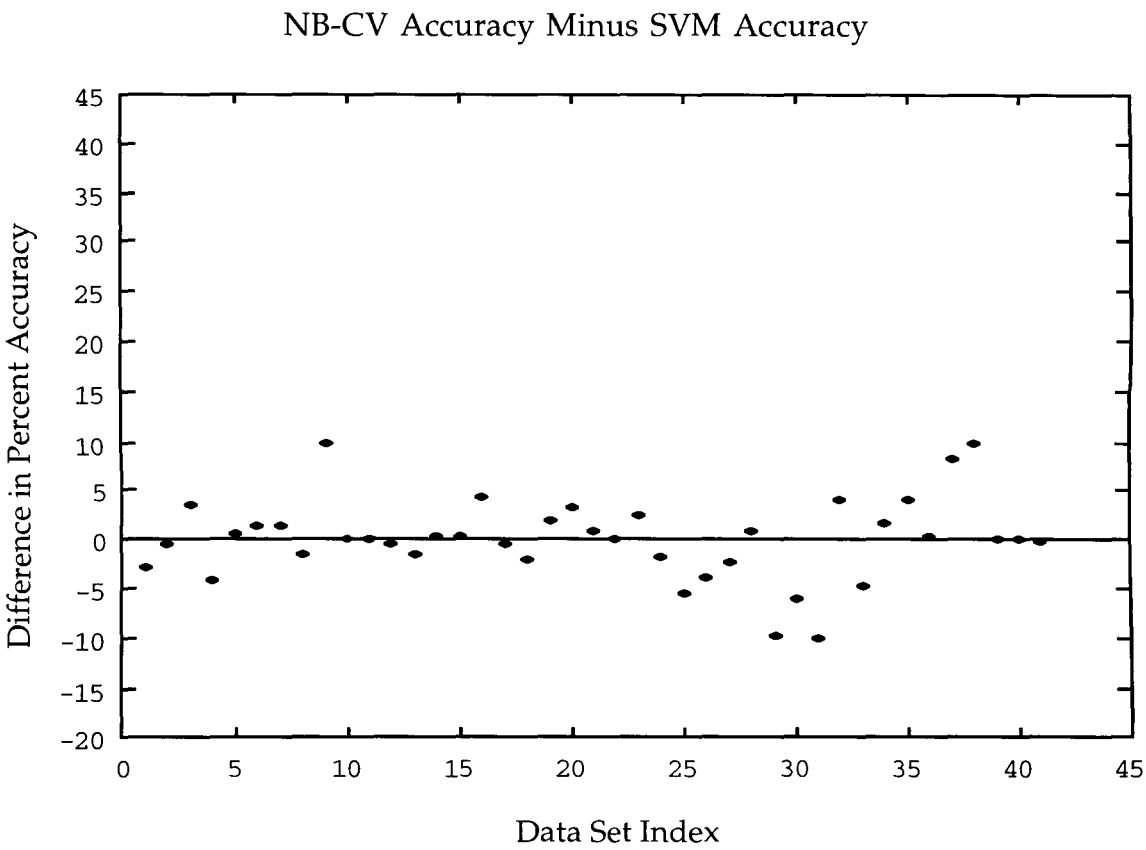


Figure 3.7: Difference in accuracy between Naive Bayes using Cross Validation (NB-CV) and the linear support vector machine (SVM) when only single words were used to identify relevant articles. The average difference is –0.05%. This is not statistically significant (Z=0.68, see Section 3.5).

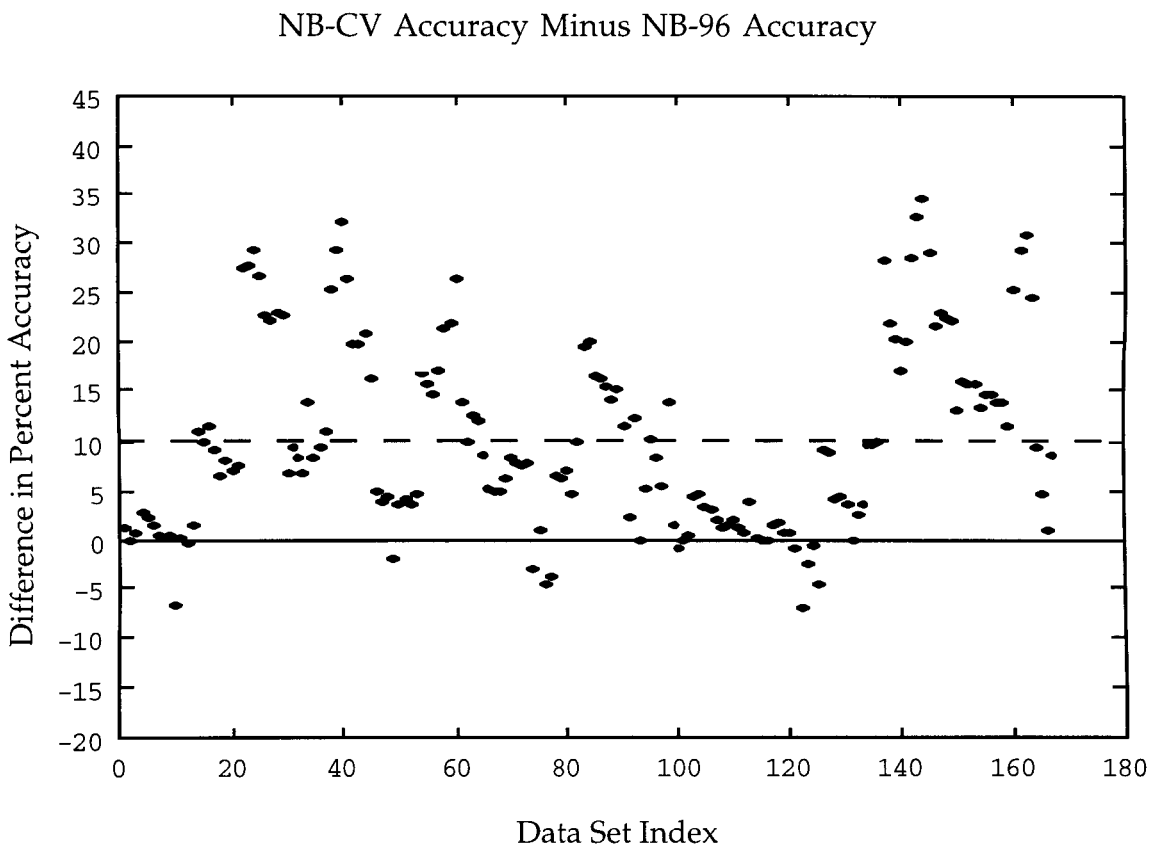


Figure 3.8: Difference in accuracy between Naive Bayes using Cross Validation (NB-CV) and Naive Bayes using 96 words (NB-96) when only single words were used to identify relevant articles. The average difference is 10.18%, as shown by the dashed line. This is statistically significant far beyond the 0.1% confidence level (Z=30, see Section 3.5).

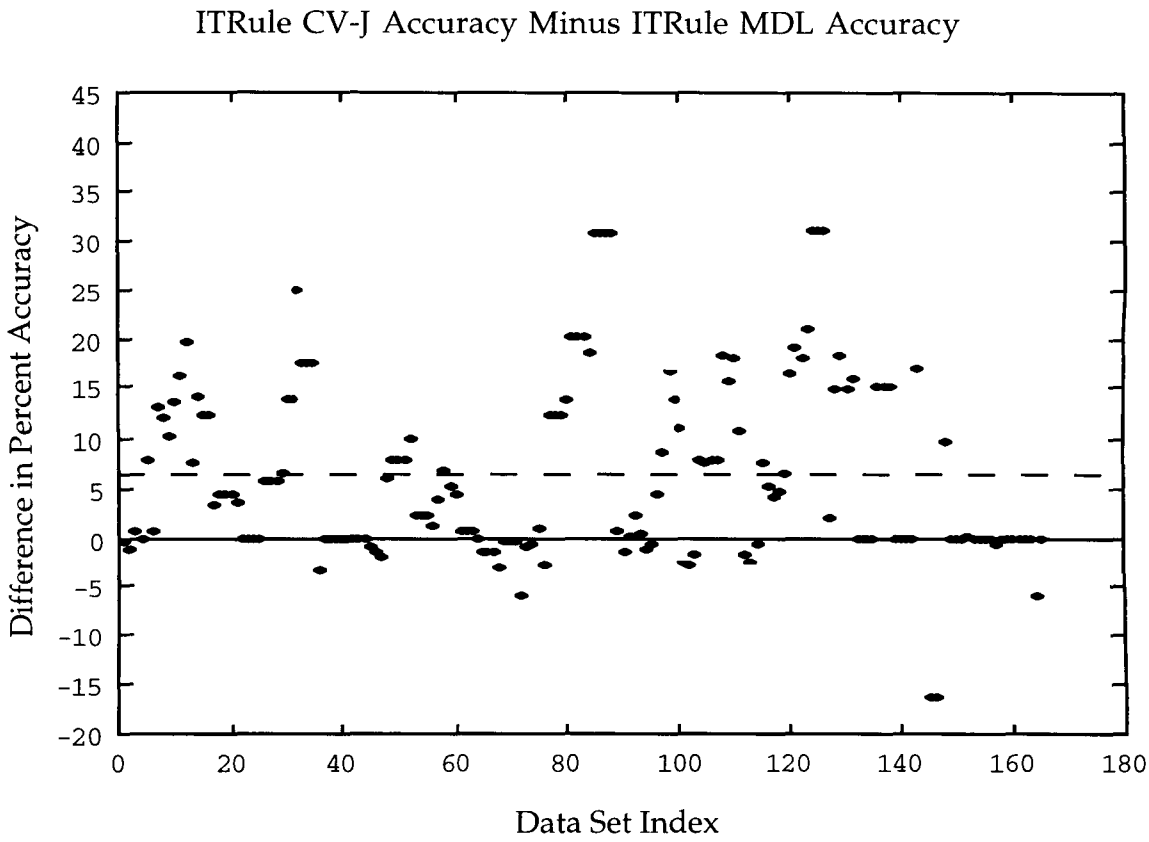


Figure 3.9: Difference in accuracy between Cross Validation using the J-measure (ITRule CV-J) and Minimum Description Length (ITRule MDL) when only single words were used to identify relevant articles. The average difference is 6.53%, as shown by the dashed line. This is statistically significant far beyond the 0.1% confidence level (Z=16, see Section 3.5).

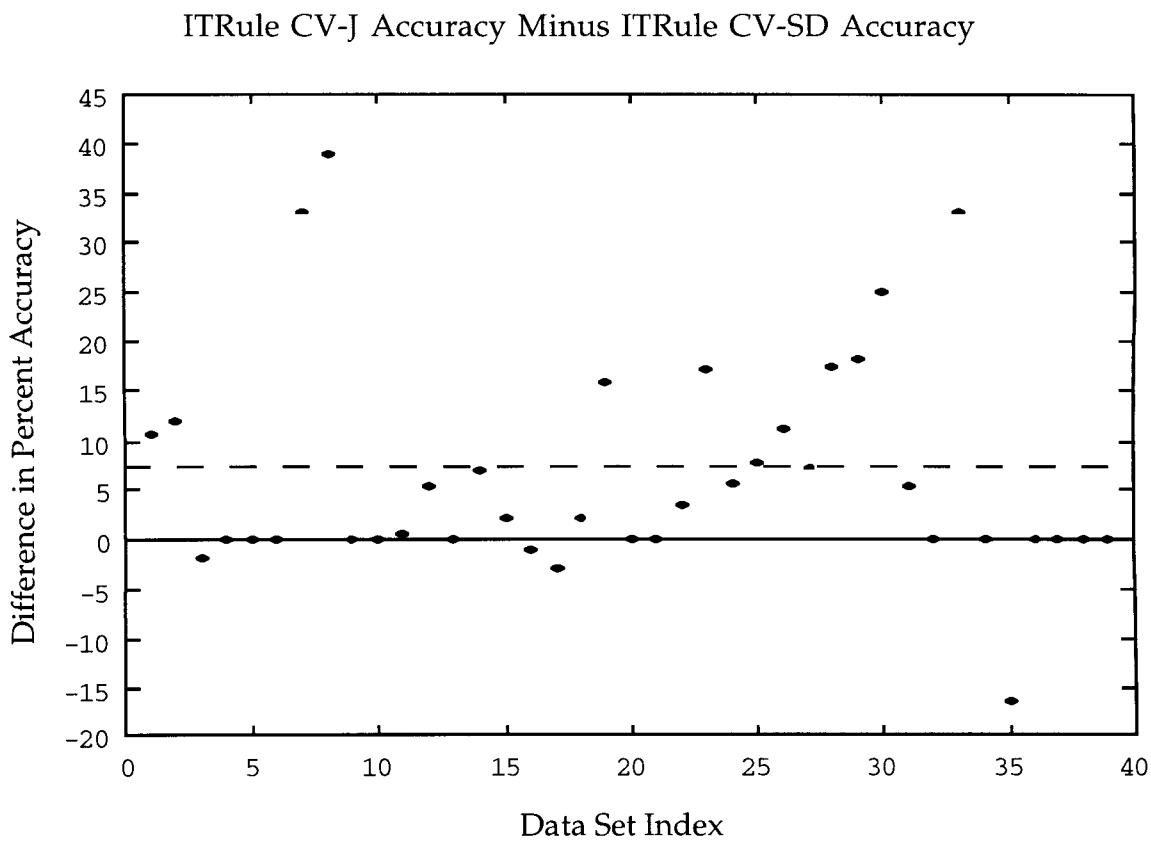


Figure 3.10: Difference in accuracy between Cross Validation using the J-measure (ITRule CV-J) and Cross Validation using Steepest Descent (ITRule CV-SD) when only single words were used to identify relevant articles. The average difference is 7.18%, as shown by the dashed line. This is statistically significant far beyond the 0.1% confidence level (Z=18, see Section 3.5).

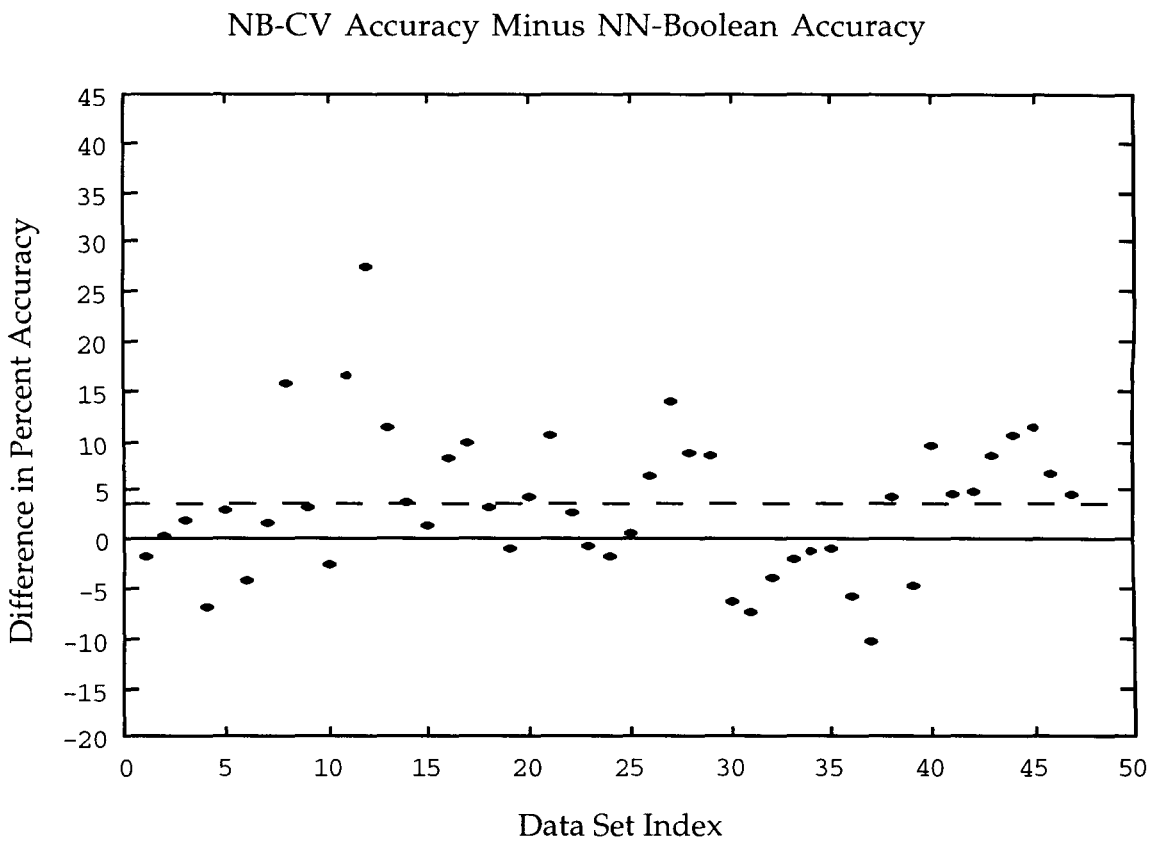


Figure 3.11: Difference in accuracy between Naive Bayes using Cross Validation (NB-CV) and Neural Network with Boolean inputs (NN-Boolean) when only single words were used to identify relevant articles. The average difference is 3.45%, as shown by the dashed line. This is statistically significant far beyond the 0.1% confidence level (Z=9.4, see Section 3.5).

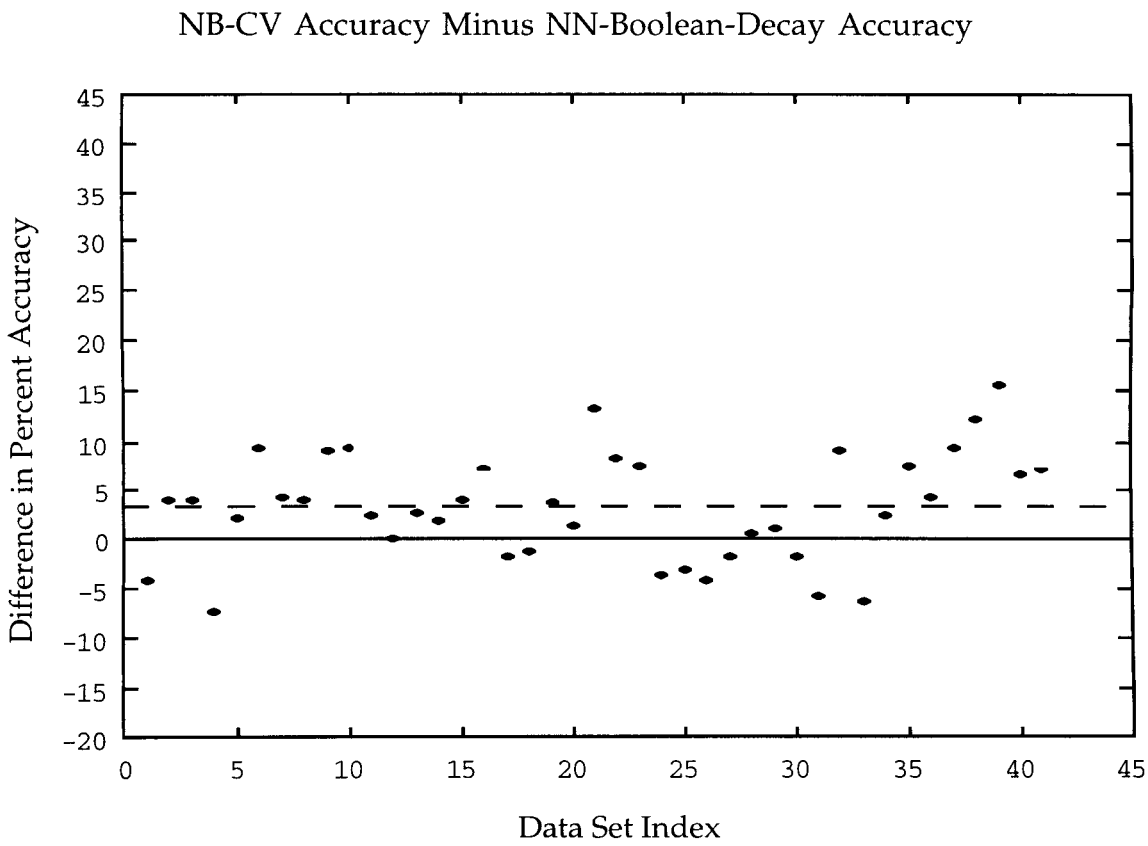


Figure 3.12: Difference in accuracy between Naive Bayes using Cross Validation (NB-CV) and Neural Network with Boolean inputs using weight decay (NN-Boolean-Decay) when only single words were used to identify relevant articles. The average difference is 3.16%, as shown by the dashed line. This is statistically significant far beyond the 0.1% confidence level (Z=11, see Section 3.5).

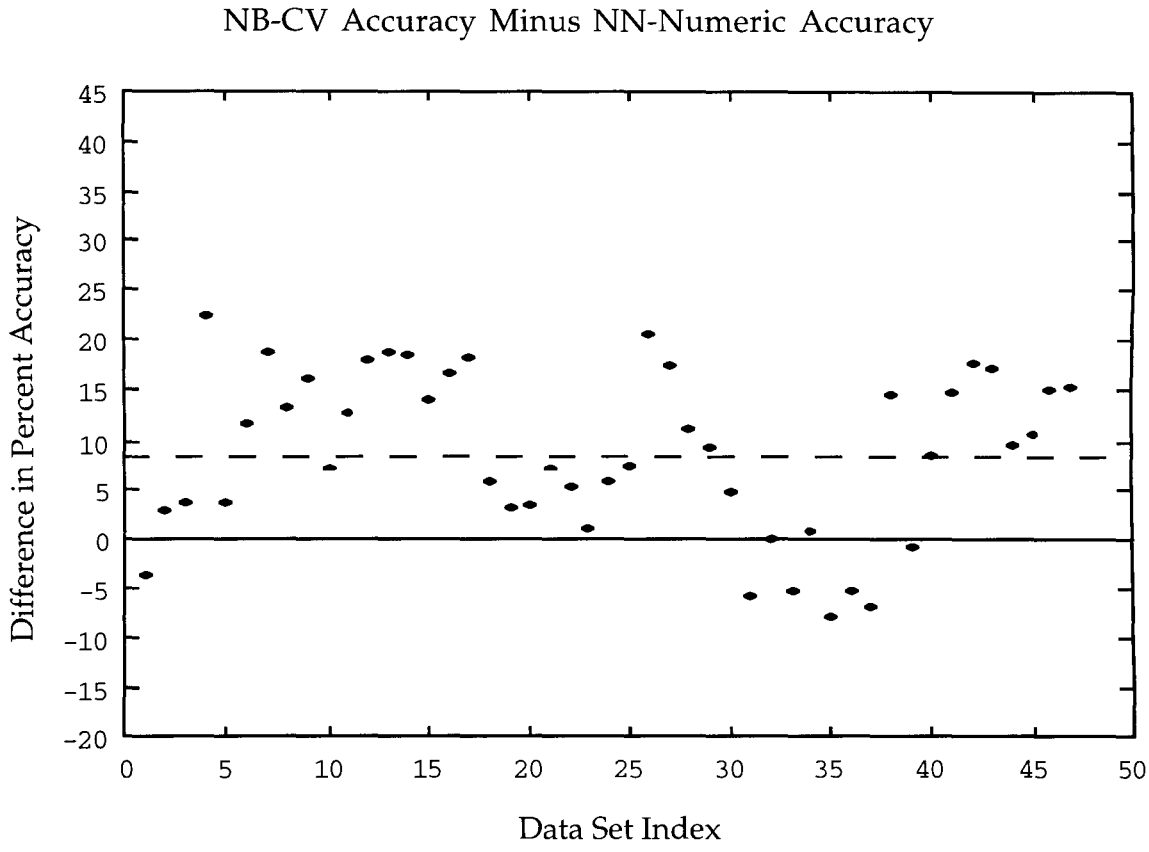


Figure 3.13: Difference in accuracy between Naive Bayes using Cross Validation (NB-CV) and Neural Networks with numeric inputs (NN-Numeric) when only single words were used to identify relevant articles. The average difference is 8.66%, as shown by the dashed line. This is statistically significant far beyond the 0.1% confidence level (Z=22, see Section 3.5).

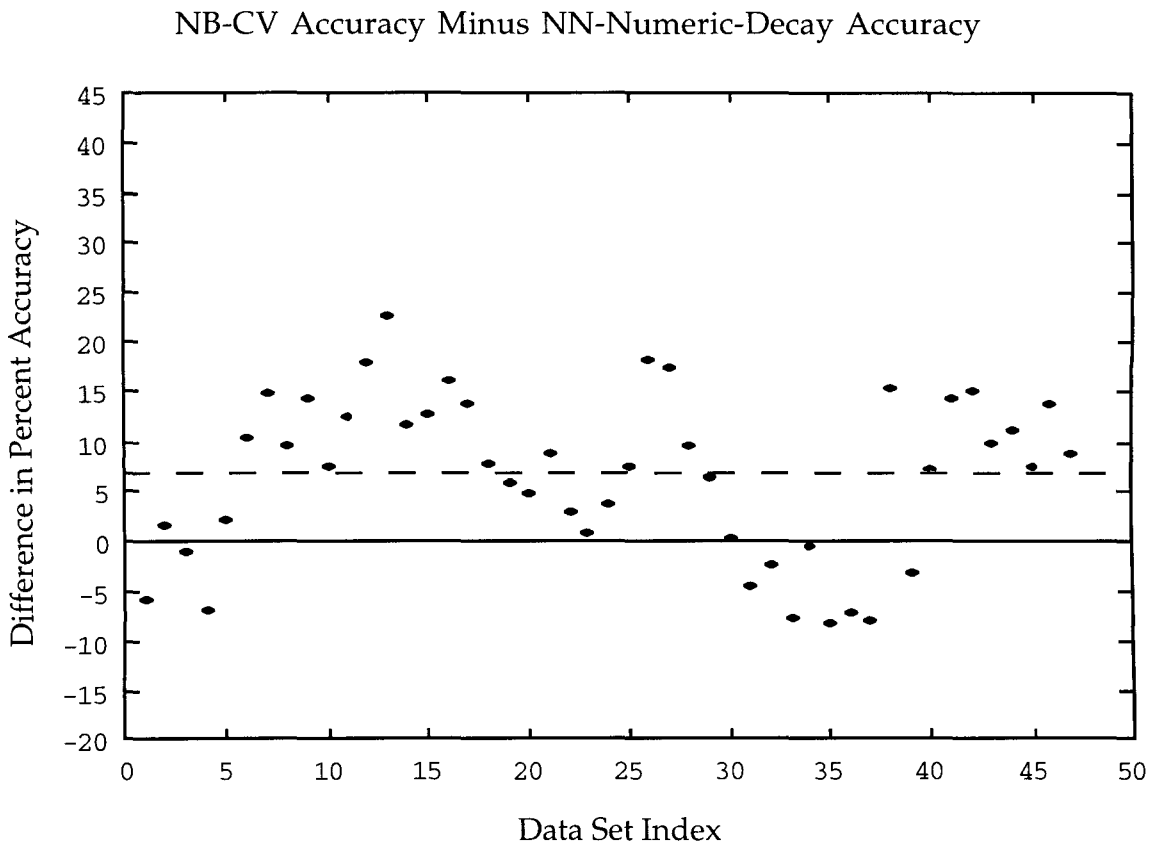


Figure 3.14: Difference in accuracy between Naive Bayes using Cross Validation (NB-CV) and Neural Networks with numeric inputs using weight decay (NN-Numeric-Decay) when only single words were used to identify relevant articles. The average difference is 6.56%, as shown by the dashed line. This is statistically significant far beyond the 0.1% confidence level (Z=17, see Section 3.5).

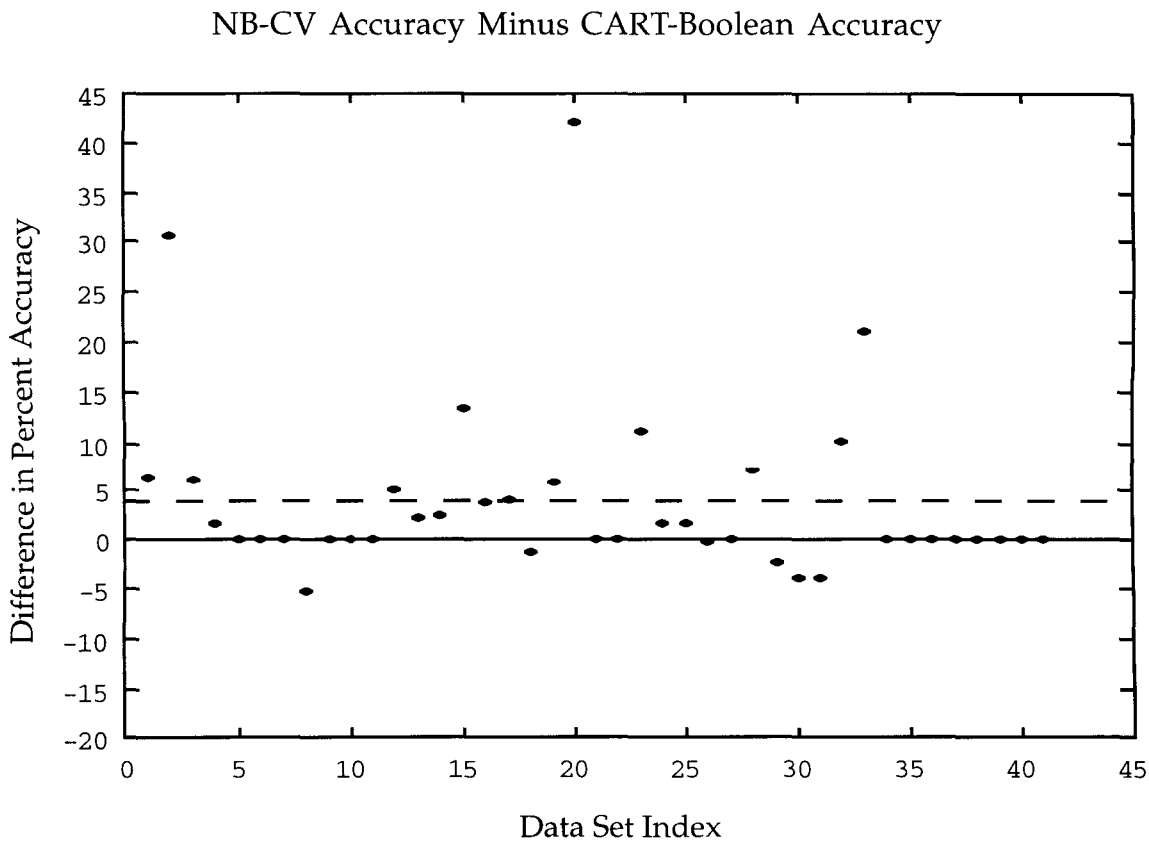


Figure 3.15: Difference in accuracy between Naive Bayes using Cross Validation (NB-CV) and CART with Boolean inputs (CART-Boolean) when only single words were used to identify relevant articles. The average difference is 3.82%, as shown by the dashed line. This is statistically significant far beyond the 0.1% confidence level (Z=15, see Section 3.5).

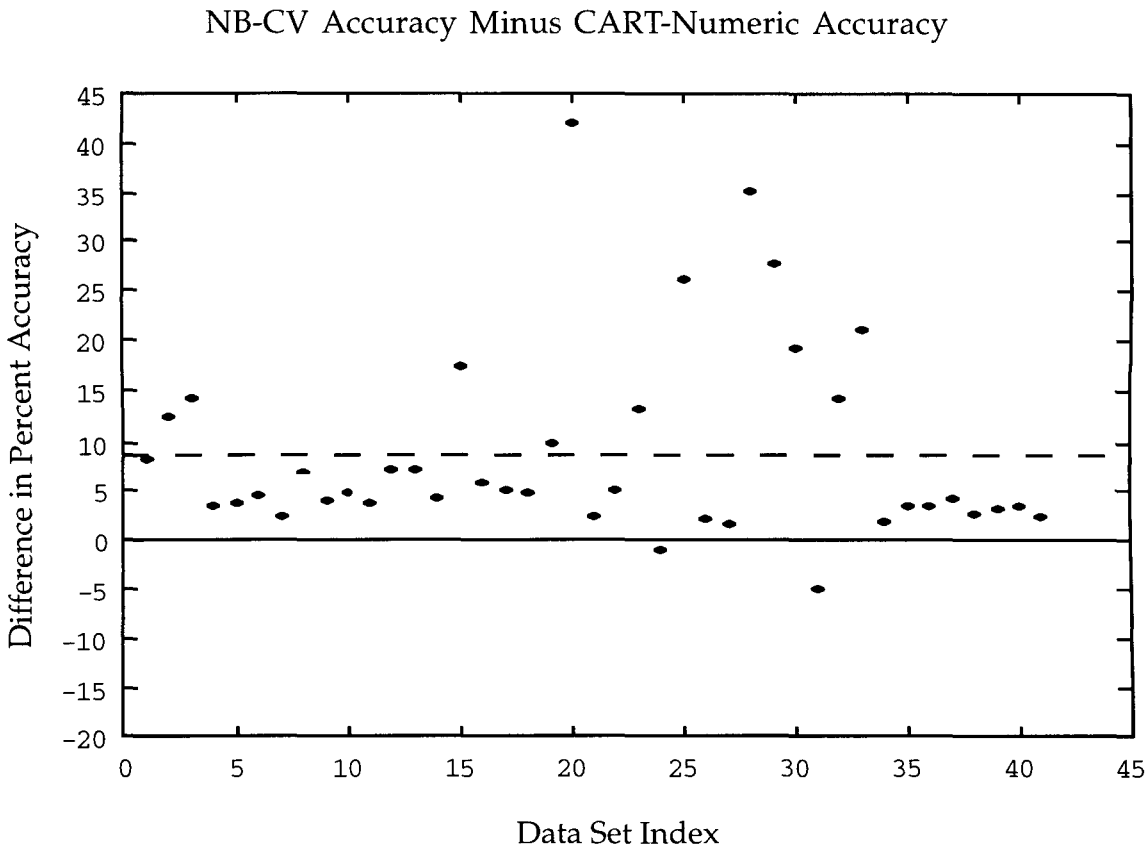


Figure 3.16: Difference in accuracy between Naive Bayes using Cross Validation (NB-CV) and CART with numeric inputs (CART-Numeric) when only single words were used to identify relevant articles. The average difference is 8.71%, as shown by the dashed line. This is statistically significant far beyond the 0.1% confidence level (Z=34, see Section 3.5).

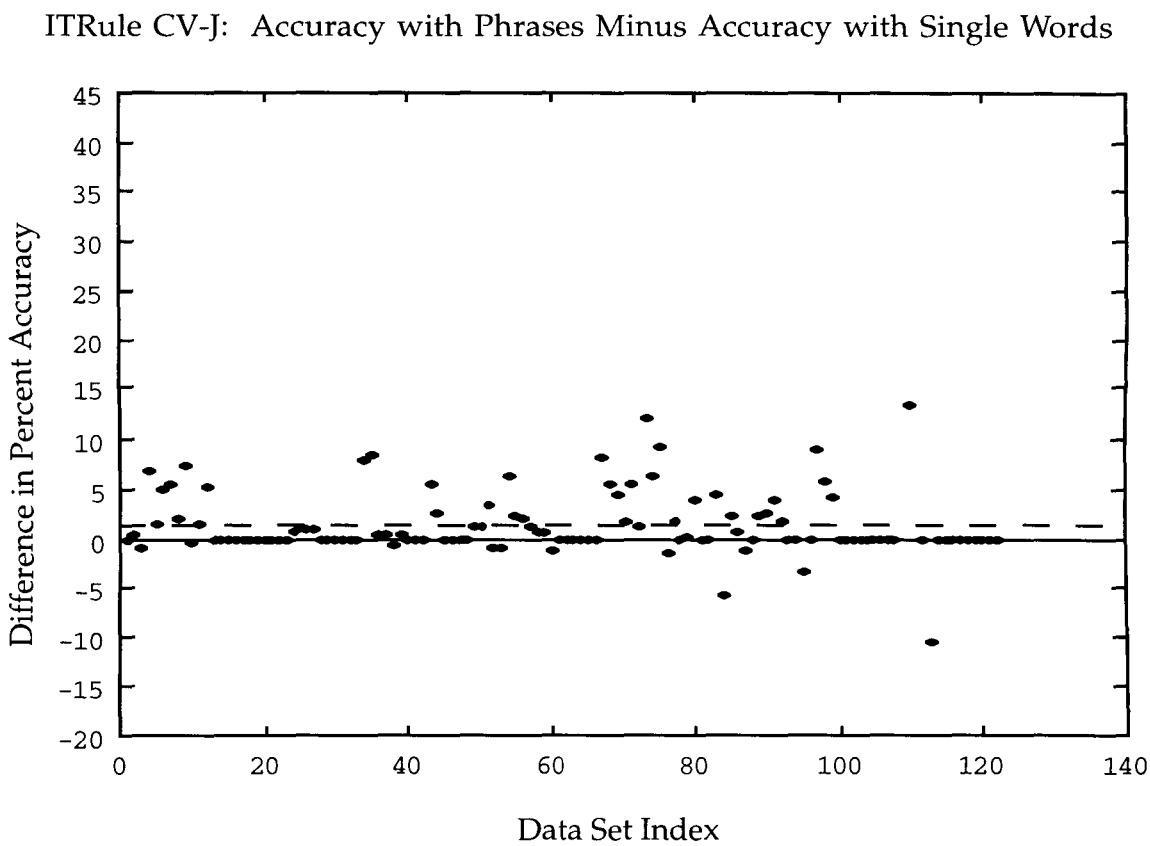


Figure 3.17: Difference in accuracy of Cross Validation using the J-measure (ITRule CV-J) between using phrases without stop words and using only single words to identify relevant articles. The average difference is 1.37%, as shown by the dashed line. This is statistically significant far beyond the 0.1% confidence level (Z=6.8, see Section 3.5).

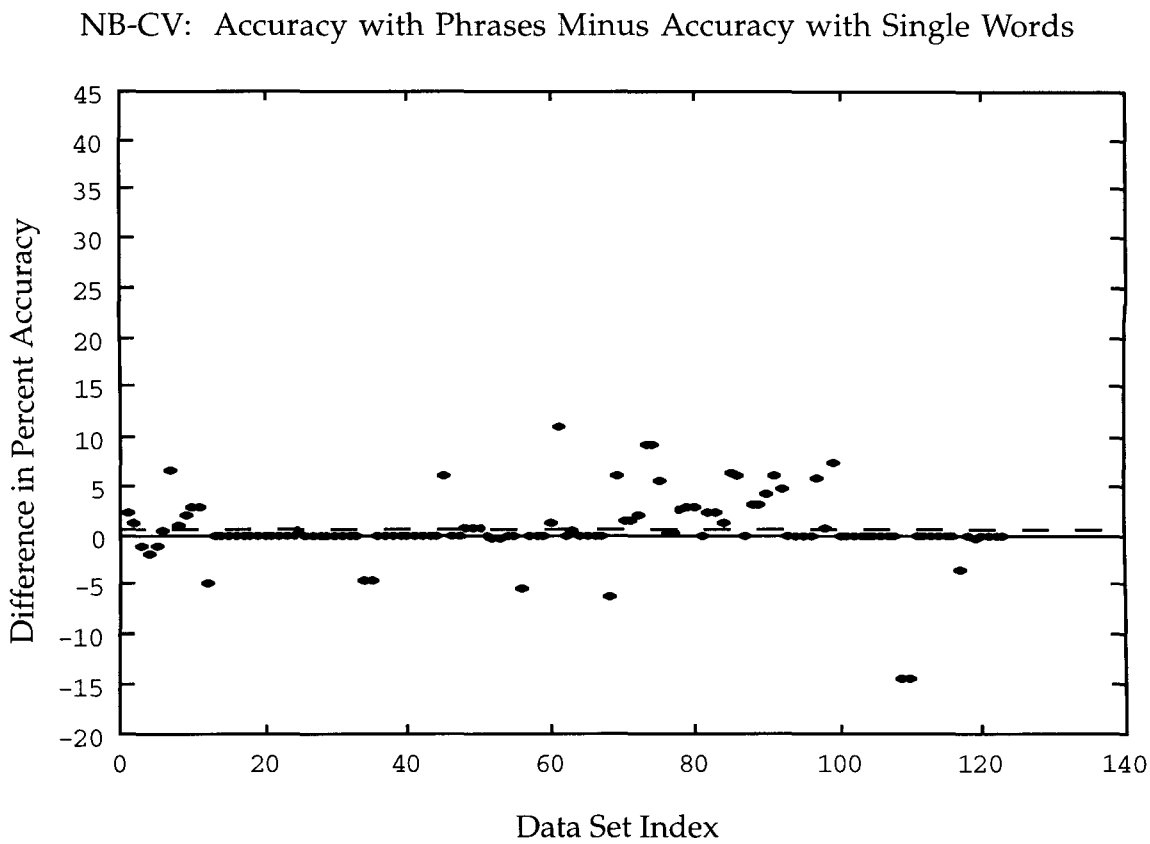


Figure 3.18: Difference in accuracy of Naive Bayes using Cross Validation (NB-CV) between using phrases without stop words and using only single words to identify relevant articles. The average difference is 0.58%, as shown by the dashed line. This is statistically significant far beyond the 0.1% confidence level (Z=4.7, see Section 3.5).

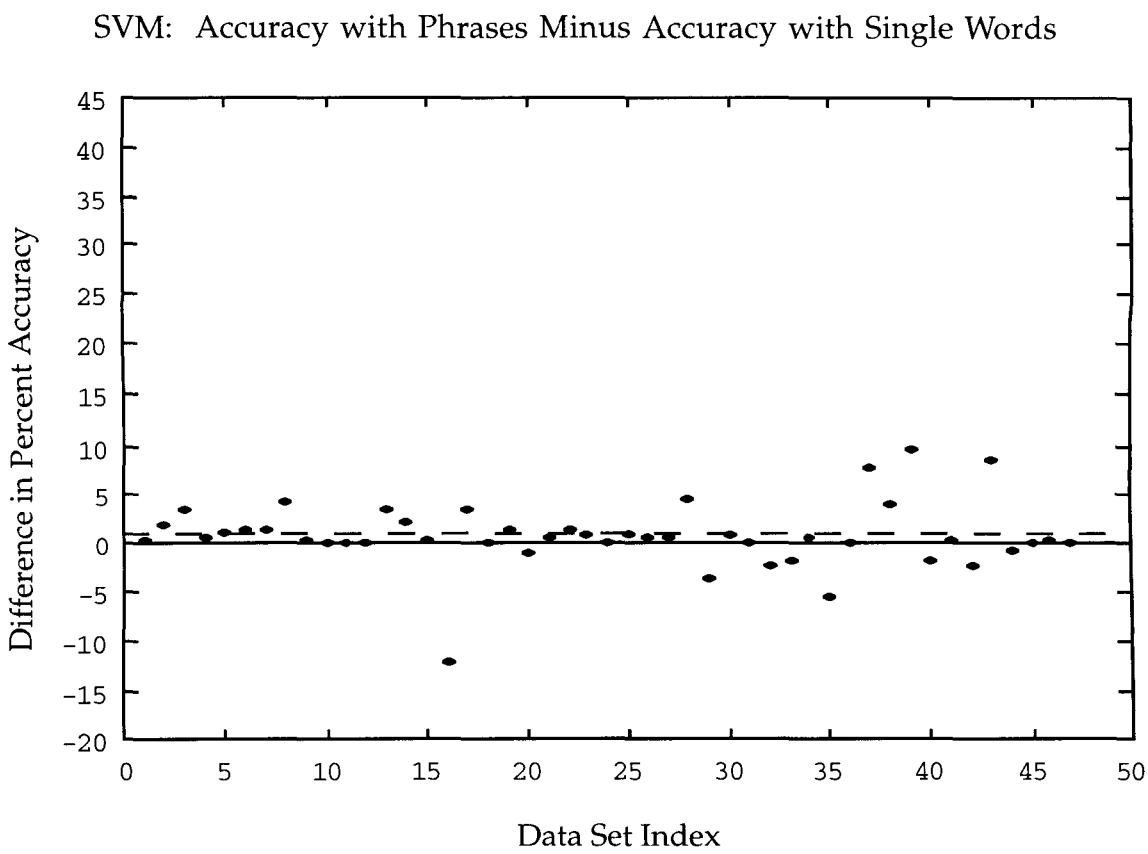


Figure 3.19: Difference in accuracy of the linear support vector machine (SVM) between using phrases without stop words and using only single words to identify relevant articles. The average difference is 0.66%, as shown by the dashed line. This is statistically significant at the 0.2% confidence level (Z=3.2, see Section 3.5).

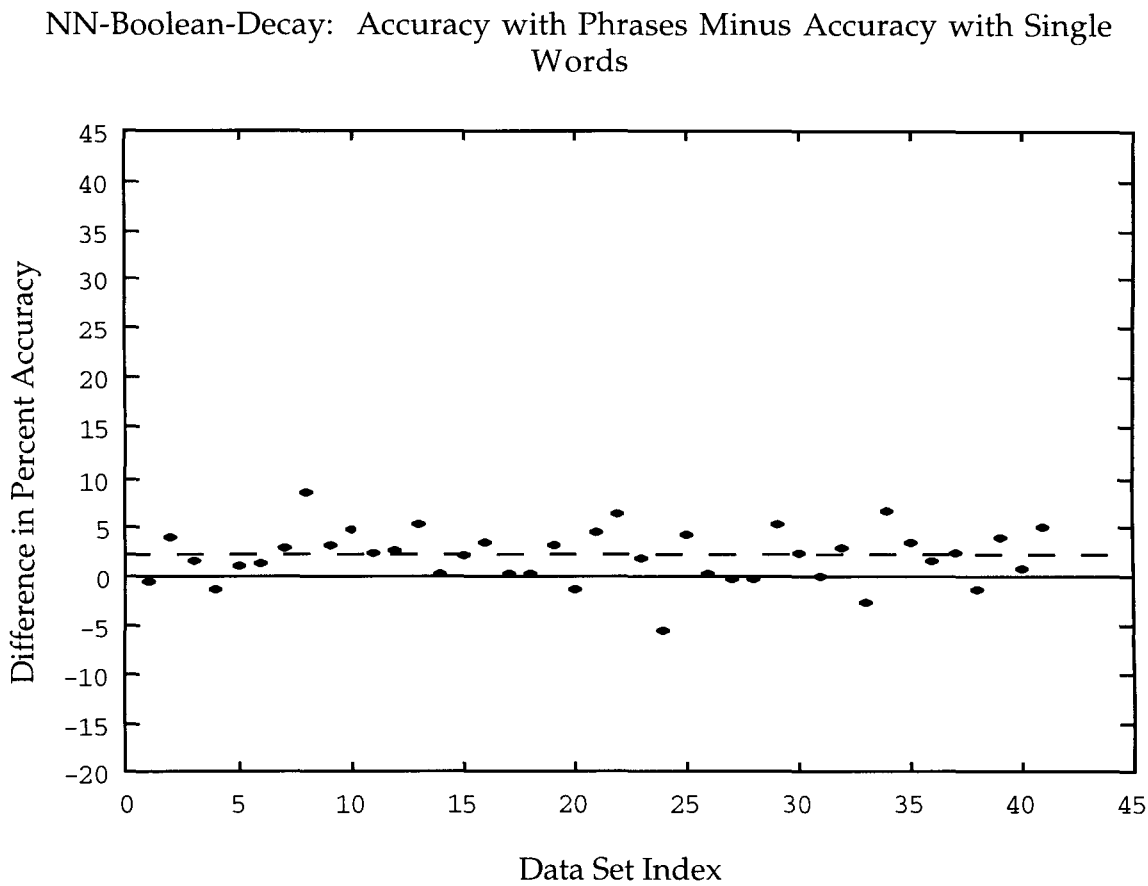


Figure 3.20: Difference in accuracy of Neural Networks with Boolean inputs using weight decay (NN-Boolean-Decay) between using phrases without stop words and using only single words to identify relevant articles. The average difference is 1.98%, as shown by the dashed line. This is statistically significant far beyond the 0.1% confidence level (Z=6.8, see Section 3.5).

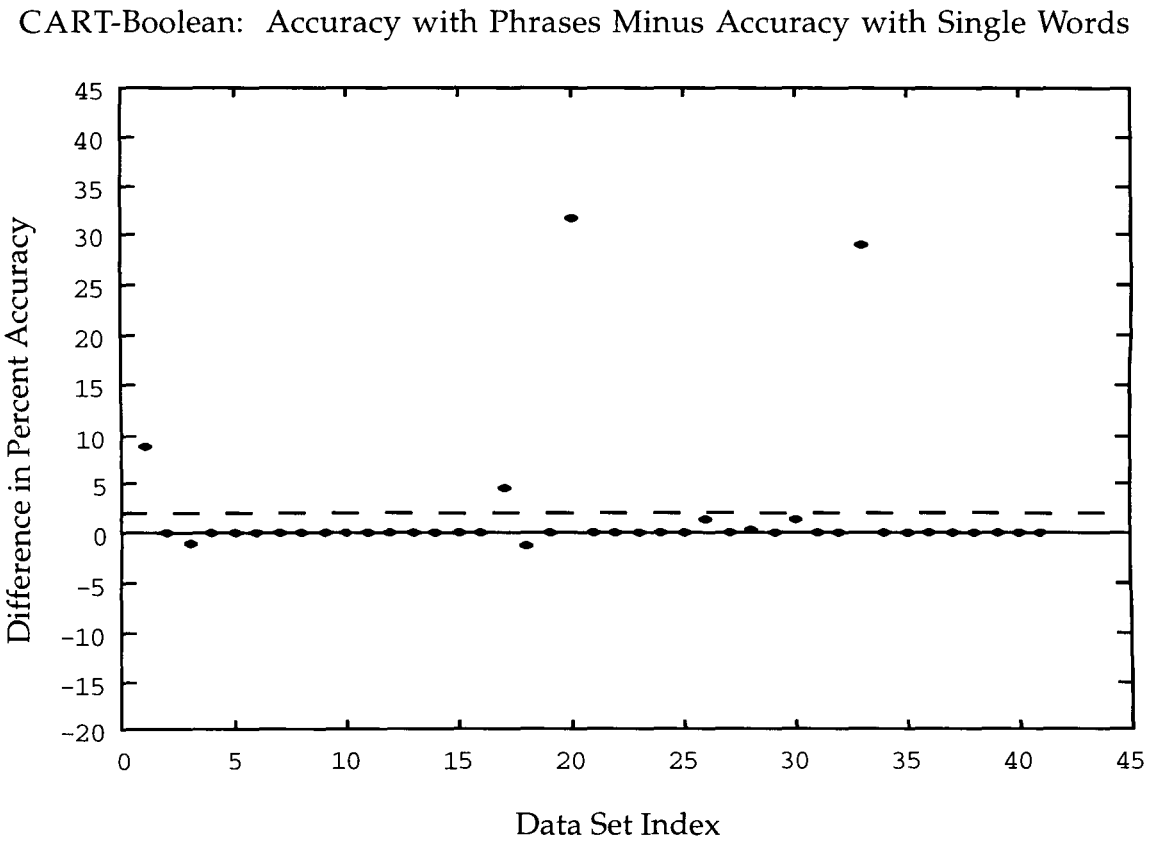


Figure 3.21: Difference in accuracy of CART with Boolean inputs (CART-Boolean) between using phrases without stop words and using only single words to identify relevant articles. The average difference is 1.81%, as shown by the dashed line. This is statistically significant far beyond the 0.1% confidence level (Z=5.4, see Section 3.5).

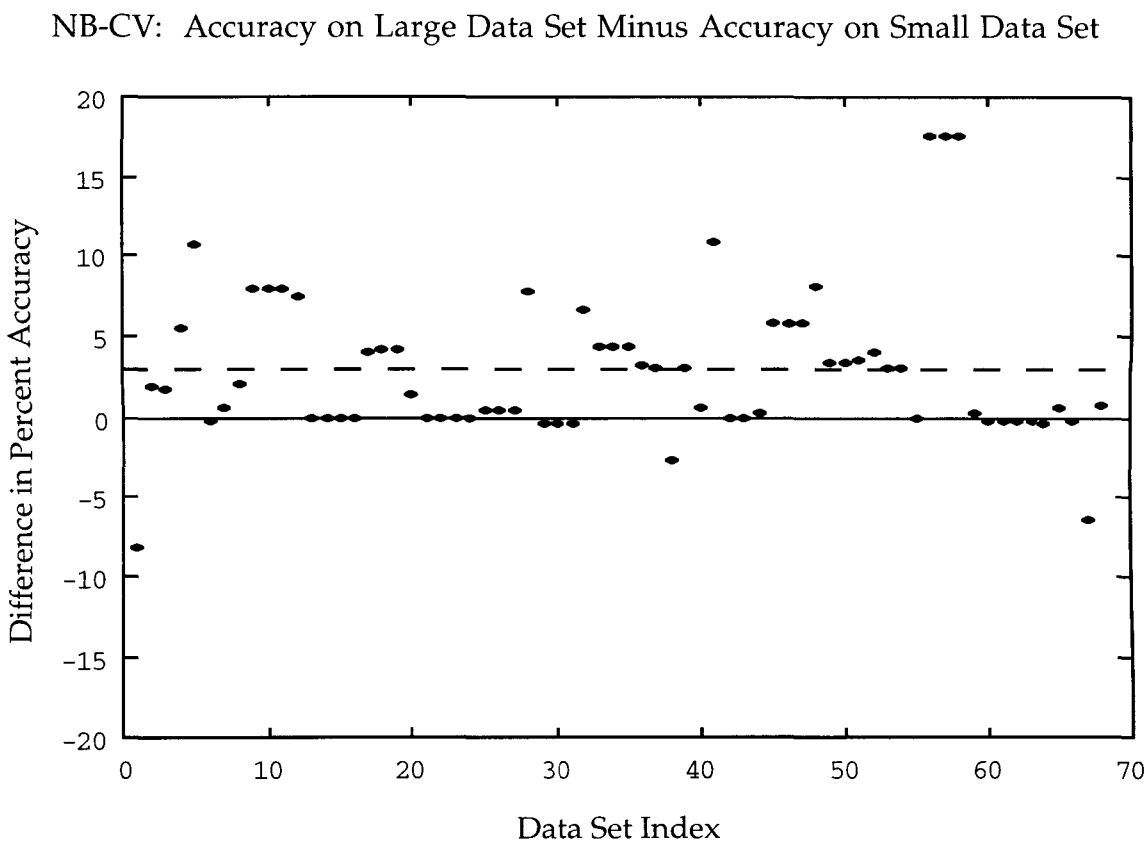


Figure 3.22: Improvement in accuracy of Naive Bayes using Cross Validation (NB-CV) when adding more training examples. The average difference is 2.92%, as shown by the dashed line. As explained in the Section 4.2, NB-CV was chosen for use in Poirot.

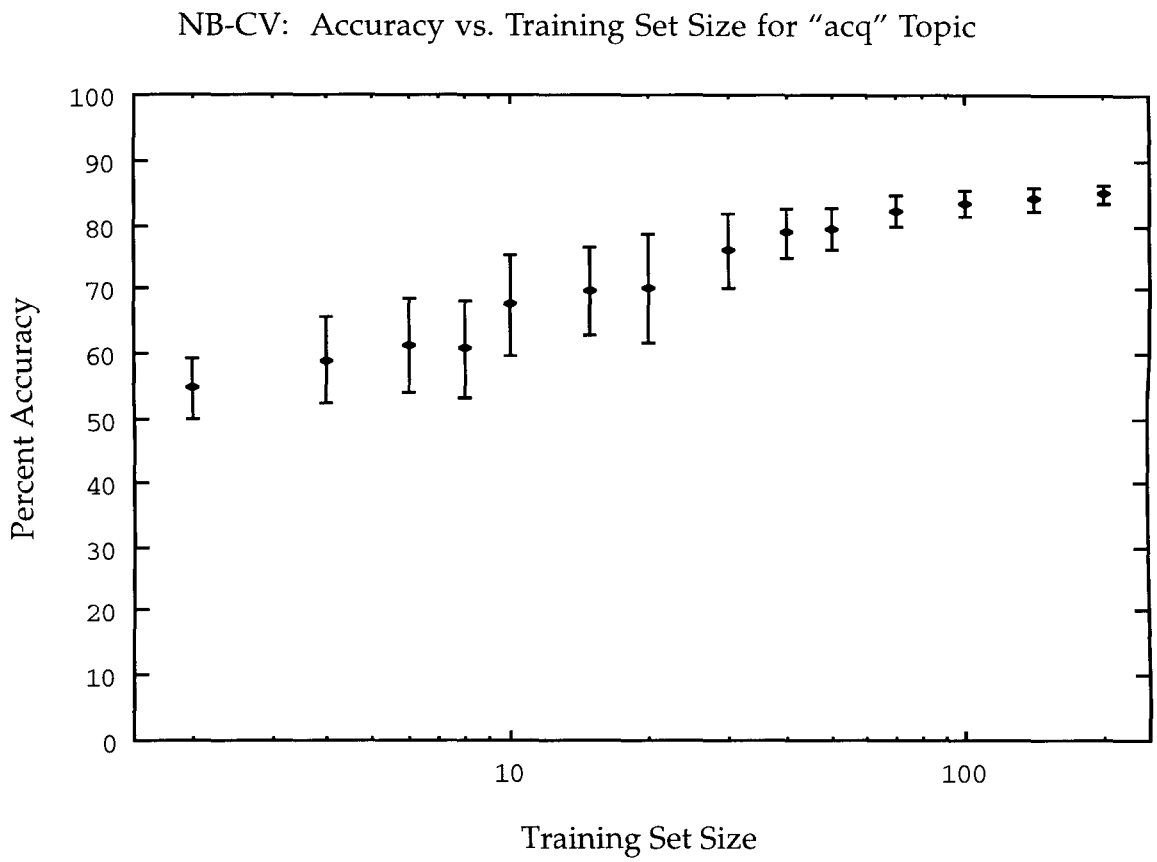


Figure 3.23: Improvement in accuracy of Naive Bayes using Cross Validation (NB-CV) as a function of training set size for the "acq" topic which deals with corporate acqusitions and mergers. For each training set size, 20 independent data sets were generated. The data point represents the average accuracy on the test data, while the error bar indicates the standard deviation of the accuracy due to the variations in the training data. As explained in the Section 4.2, NB-CV was chosen for use in Poirot.

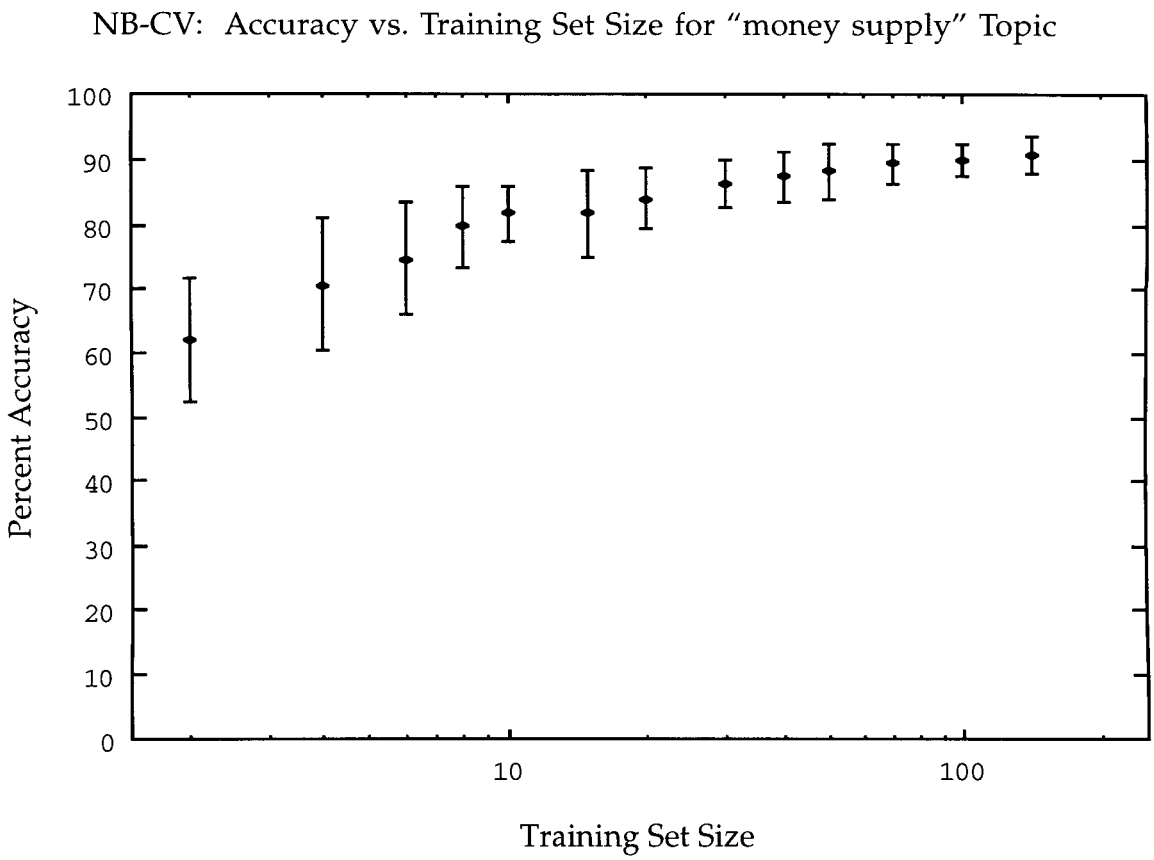


Figure 3.24: Improvement in accuracy of Naive Bayes using Cross Validation (NB-CV) as a function of training set size for the "money supply" topic which deals with Federal monetary policy. For each training set size, 20 independent data sets were generated. The data point represents the average accuracy on the test data, while the error bar indicates the standard deviation of the accuracy due to the variations in the training data. As explained in Section 4.2, NB-CV was chosen for use in Poirot.

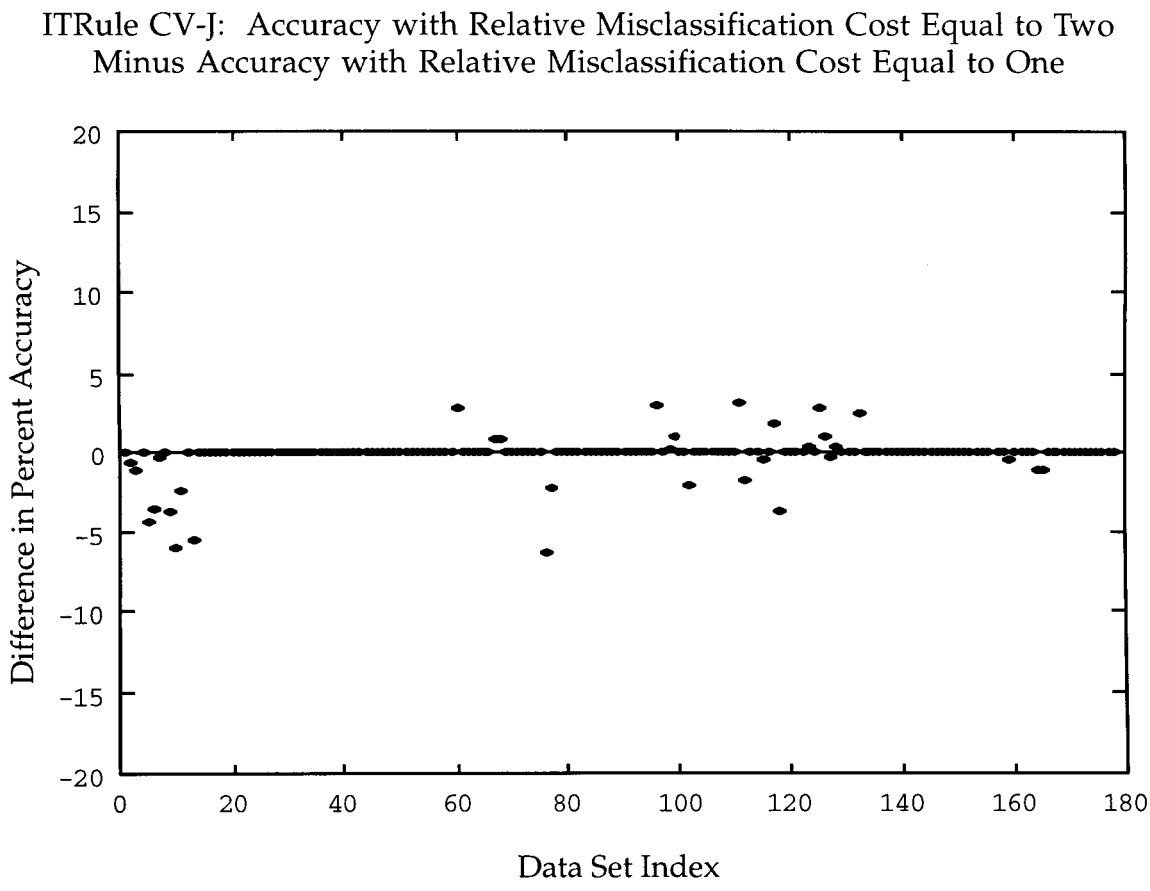


Figure 3.25: Difference in accuracy of Cross Validation using the J-measure (ITRule CV-J) between using a relative misclassification cost of two and one. Misclassification costs are discussed in Section 3.2. The average difference is 0.019%.

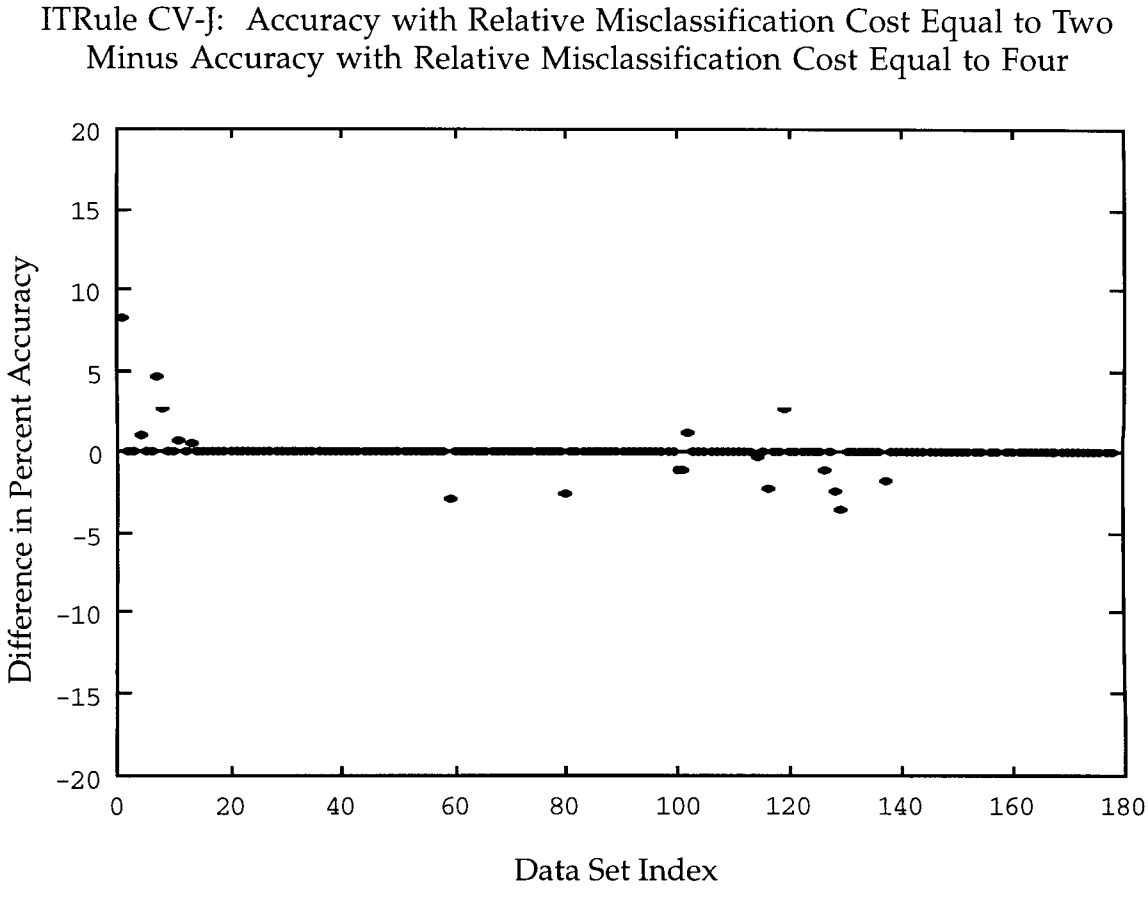


Figure 3.26: Difference in accuracy of Cross Validation using the J-measure (ITRule CV-J) between using a relative misclassification cost of two and four. Misclassification costs are discussed in Section 3.2. The average difference is 0.012%.

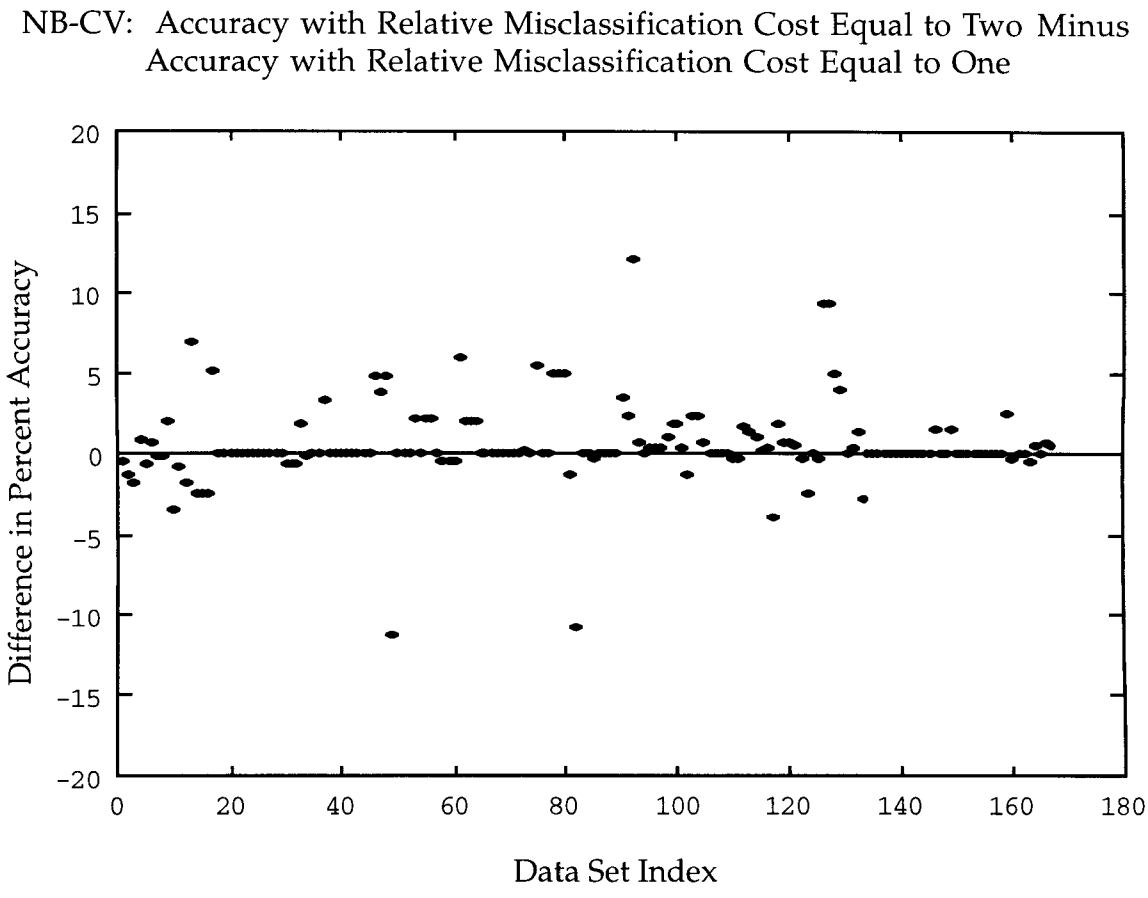


Figure 3.27: Difference in accuracy of Naive Bayes using Cross Validation (NB-CV) between using a relative misclassification cost of two and one. Misclassification costs are discussed in Section 3.2. The average difference is 0.54%.

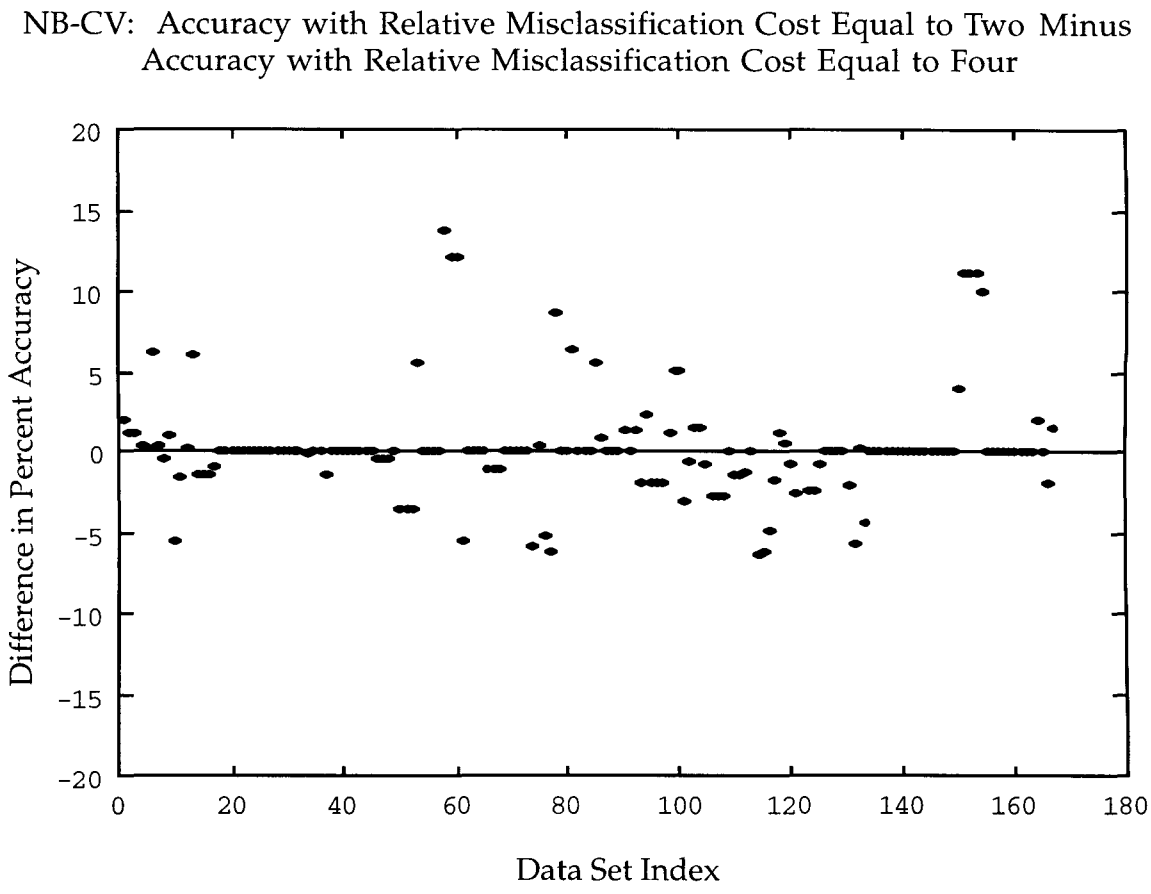


Figure 3.28: Difference in accuracy of Naive Bayes using Cross Validation (NB-CV) between using a relative misclassification cost of two and four. Misclassification costs are discussed in Section 3.2. The average difference is 0.57%.

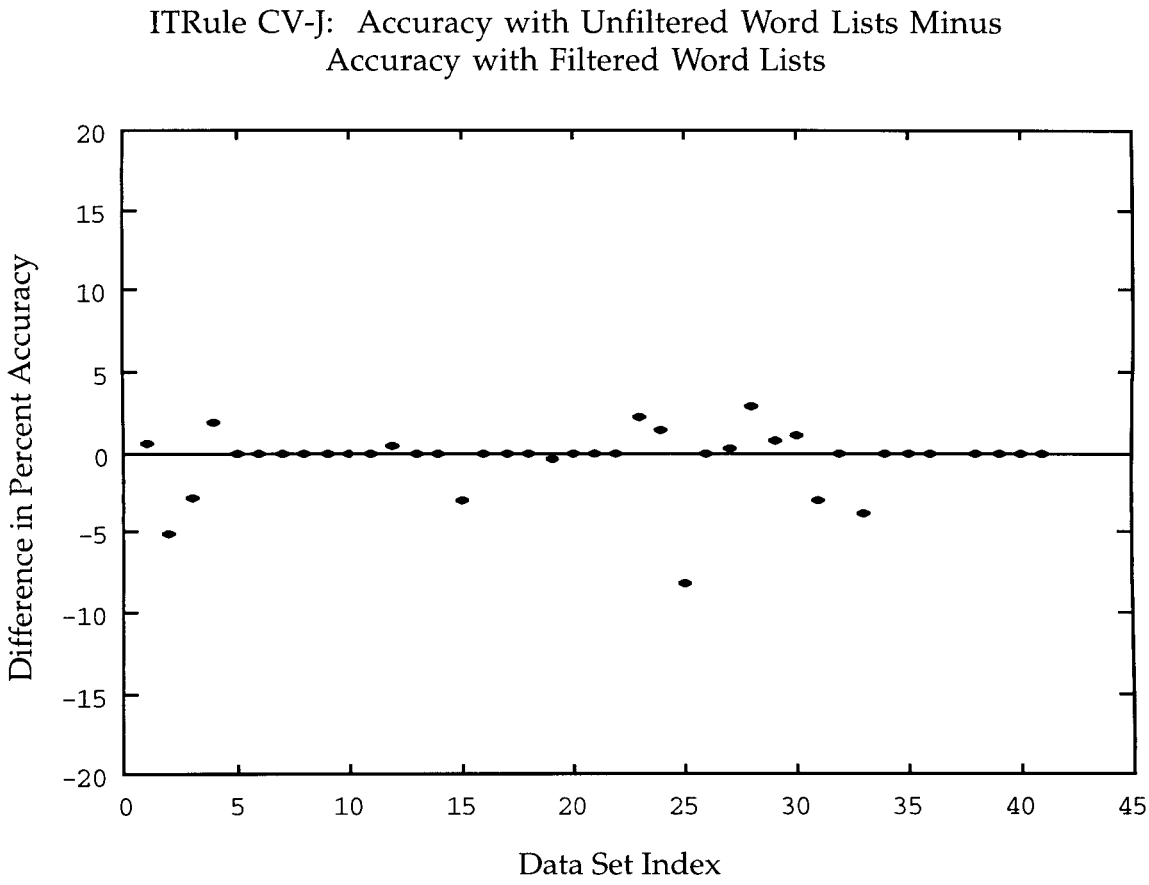


Figure 3.29: Difference in accuracy of Cross Validation using the J-measure (ITRule CV-J) between using unfiltered and filtered word lists to identify relevant articles. The average difference is –0.36%.

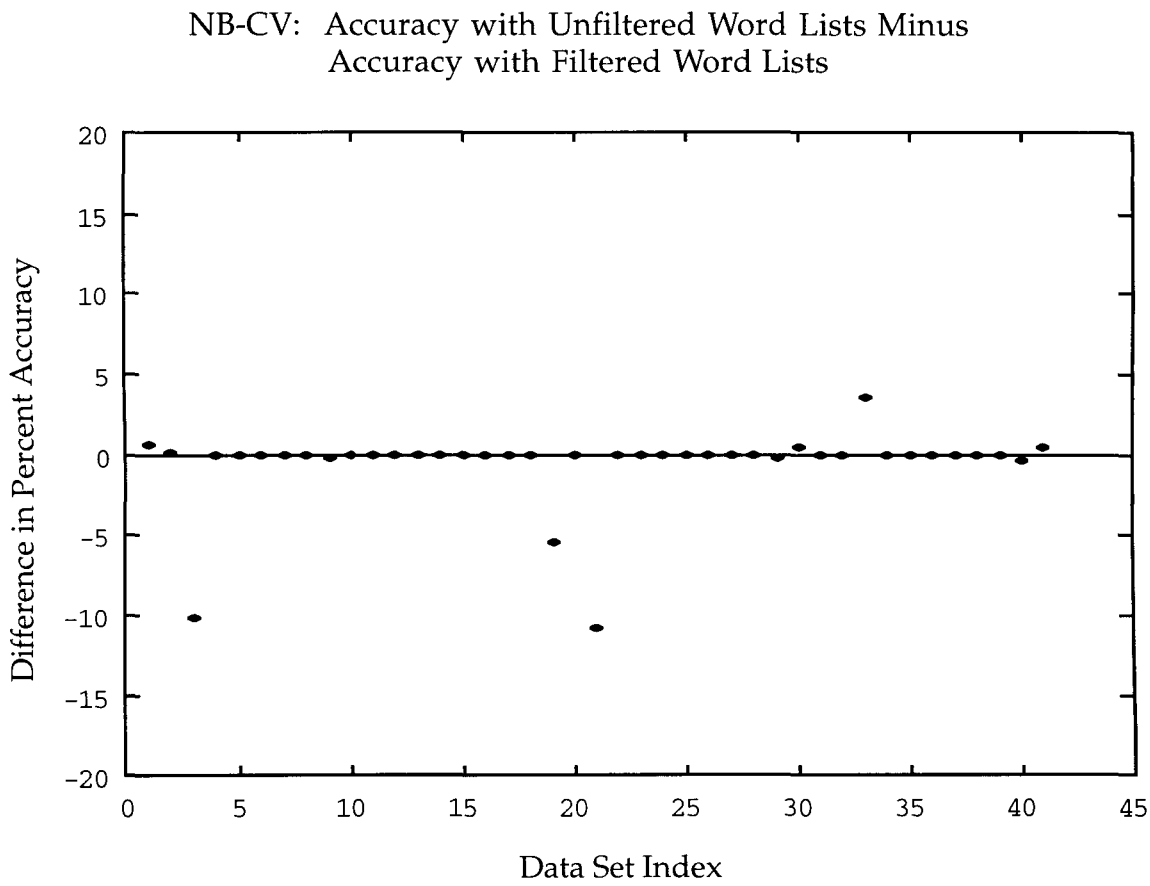


Figure 3.30: Difference in accuracy of Naive Bayes using Cross Validation (NB-CV) between using unfiltered and filtered word lists to identify relevant articles. The average difference is –0.53%.

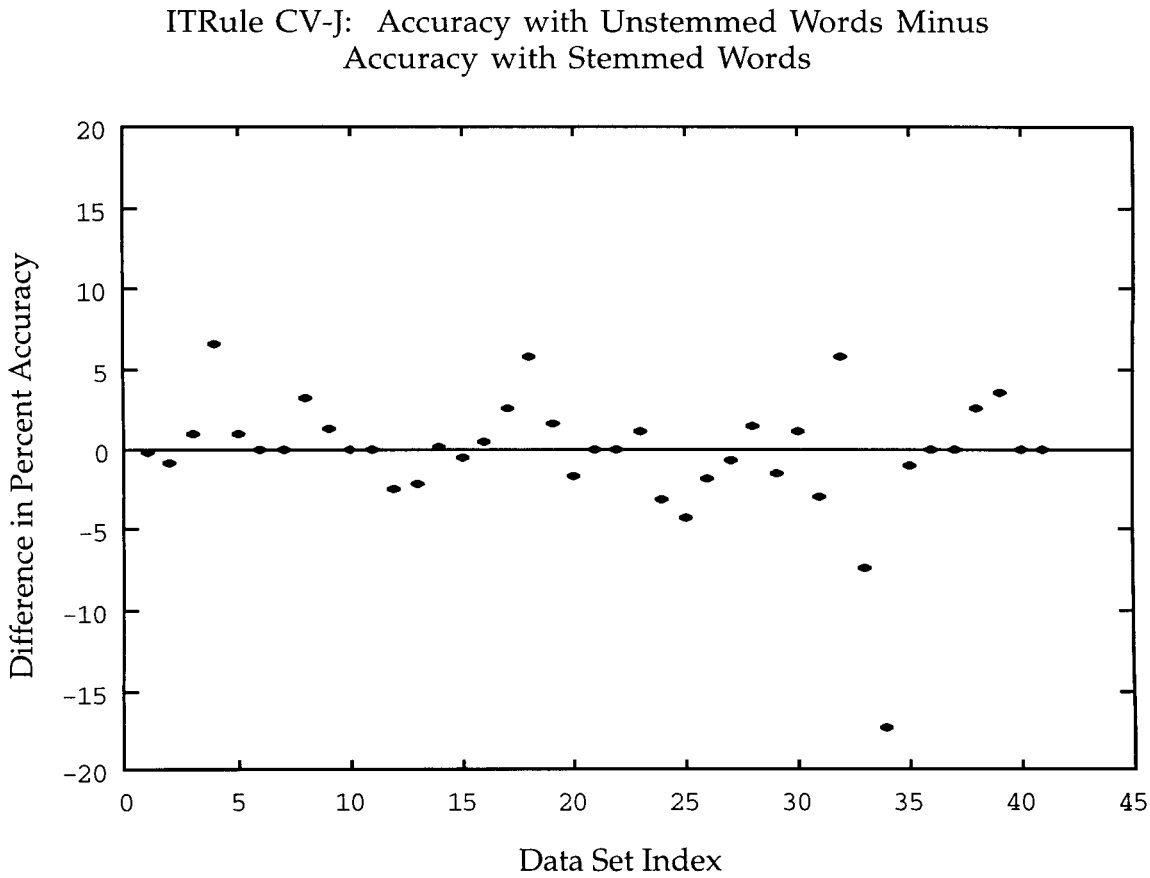


Figure 3.31: Difference in accuracy of Cross Validation using the J-measure (ITRule CV-J) between using unstemmed and stemmed words to identify relevant articles. The average difference is –0.20%.

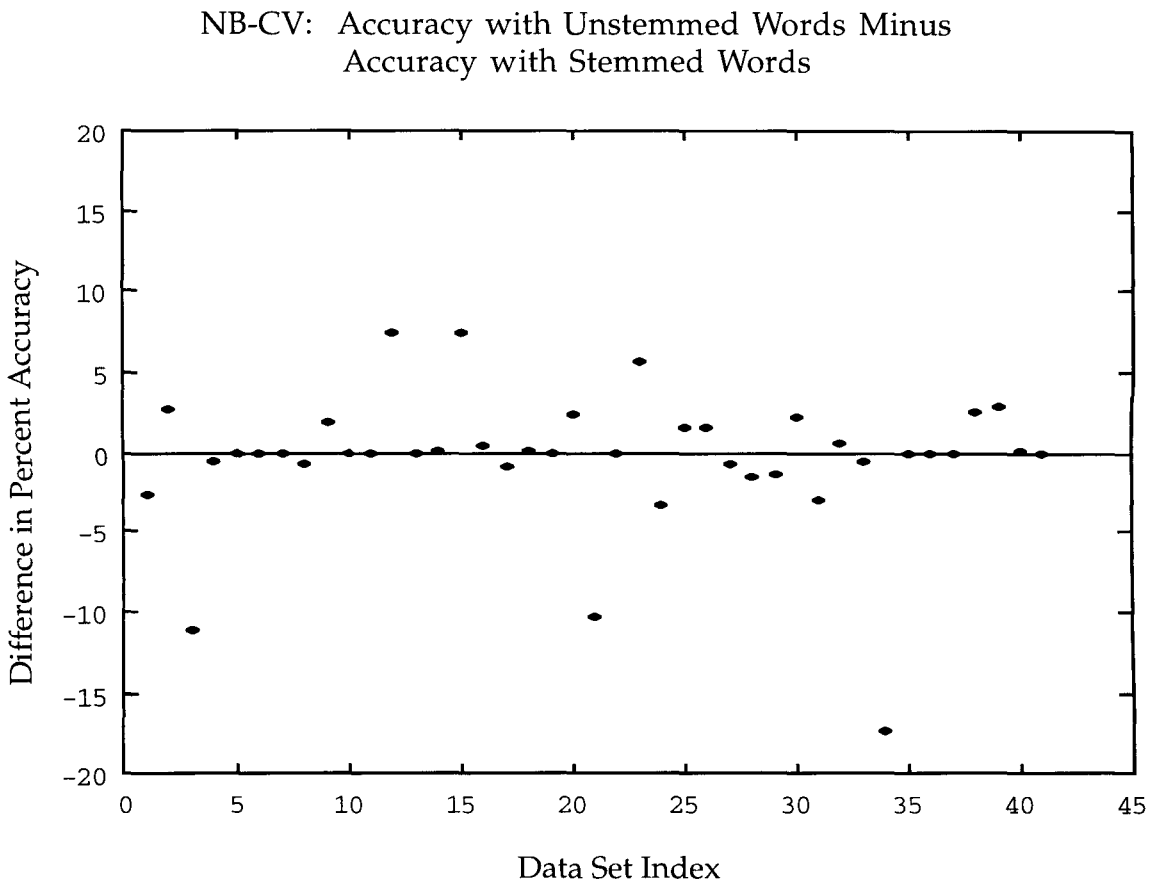


Figure 3.32: Difference in accuracy of Naive Bayes using Cross Validation (NB-CV) between using unstemmed and stemmed words to identify relevant articles. The average difference is –0.33%.

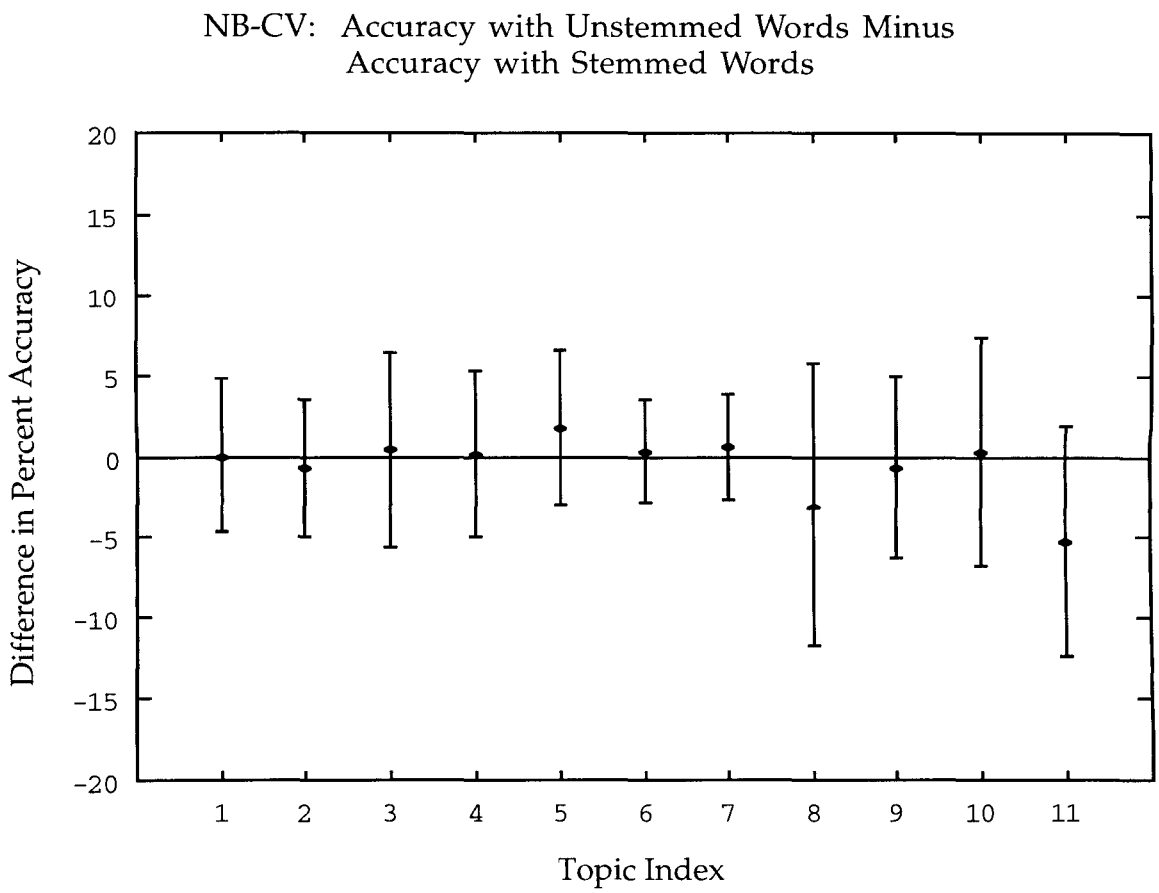


Figure 3.33: Difference in accuracy of Naive Bayes using Cross Validation (NB-CV) between using unstemmed and stemmed words to identify relevant articles. For each topic, 20 independent data sets were generated. The data point represents the difference in average accuracy on the test data, while the error bar indicates the standard deviation of the difference in accuracy. The average difference in accuracy over all 11 topics is –0.52%. As explained in the Section 4.2, NB-CV was chosen for use in Poirot.

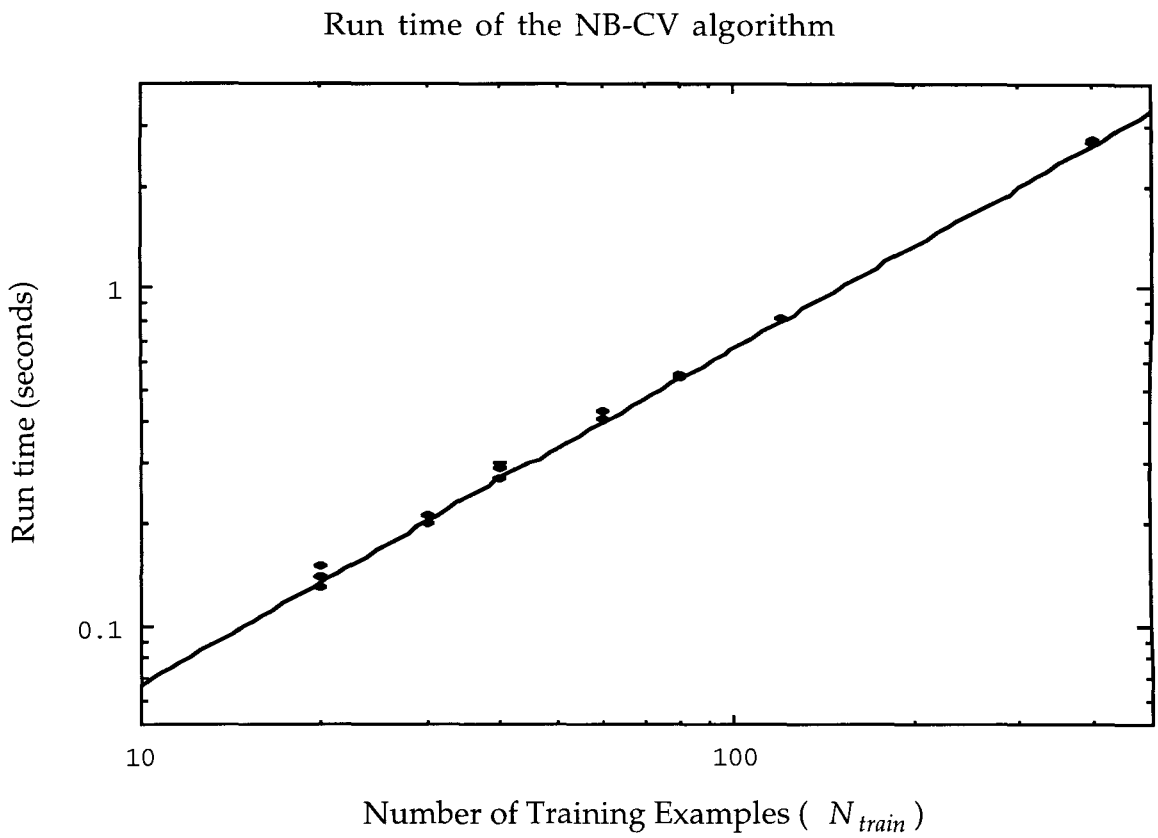


Figure 3.34: The discrete points show the measured run times of the NB-CV algorithm as a function of the number of training examples. The solid line has the slope of N_{train}. By comparing the data points and the line, it is evident that the NB-CV run time is approximately proportional to N_{train}.

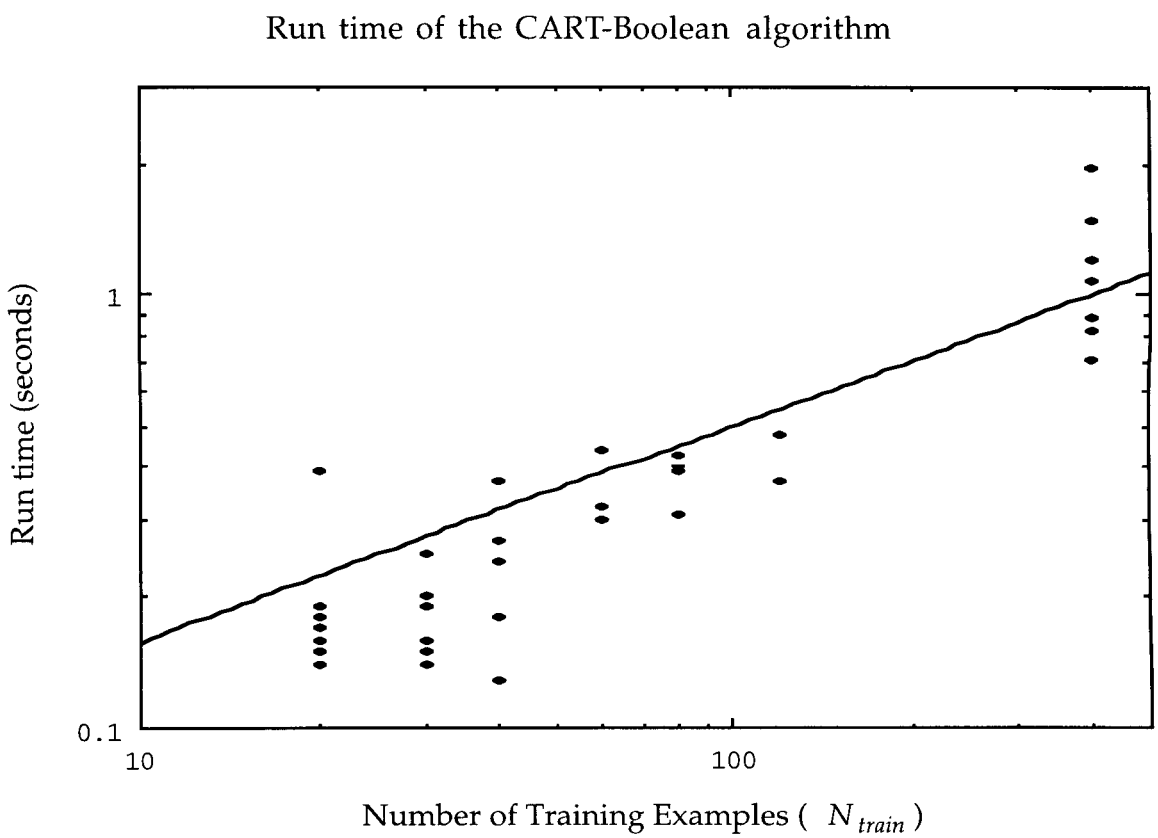


Figure 3.35: The discrete points show the measured run times of the CART-Boolean algorithm as a function of the number of training examples. The solid line has the slope of $\sqrt{N_{train}}$. By comparing the data points and the line, it appears that the CART run time is approximately proportional to $\sqrt{N_{train}}$.

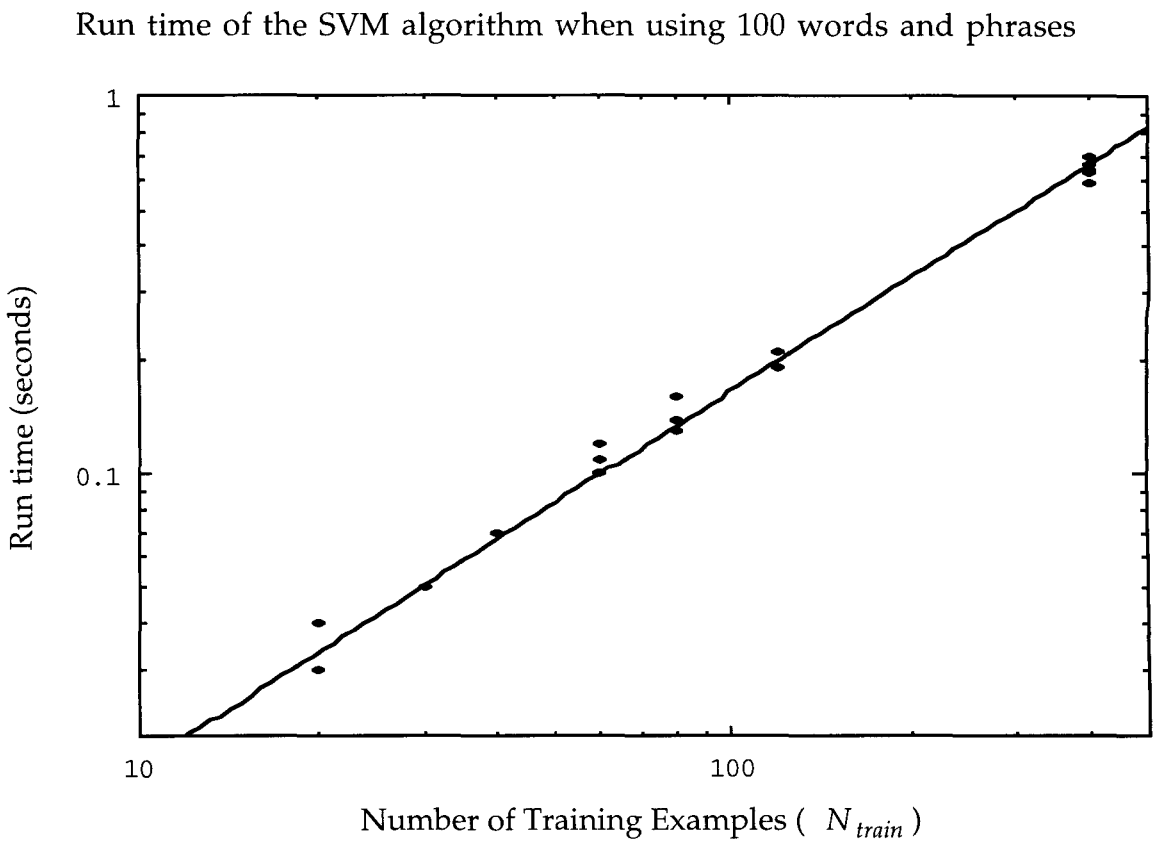


Figure 3.36: The discrete points show the measured run times of the SVM algorithm as a function of the number of training examples. The solid line has the slope of N_{train}. By comparing the data points and the line, it is evident that the SVM run time is approximately proportional to N_{train}. Since cross validation was done by a separate program and was therefore very slow, as explained in Section 3.4.5, the time required for this computation was not measured. Instead, the run time was measured for the case when the top 100 words and phrases were used as input variables. If all the computations were done in one program, cross validation would, in the worst case, produce an additional factor of N_{train} in the computational complexity.

Appendix 3-A: List of stop words

a
about
above
according
across
actual
added
after
against
ahead
all
almost
alone
along
also
among
amongst
an
and
and-or
and/or
anon
another
any
are
arising
around
as
at
award
away
be
because
become
becomes
been
before
behind
being
below
best
better
between
beyond
birthday
both
but
by
can
certain
come
comes
coming
completely
concerning
consider
considered
considering
consisting
de
department
der
despite
discussion
do
does
doesnt
doing
down
dr
du
due
during
each
either
especially
et
few
for
forward
from
further
get
give
given
giving
has
have
having
his
honor
how
in
inside
instead
into
is
it
items
its
just
let
lets
little
look
looks
made
make
makes
making
many
meet
meets
more
most
much
must
my
near
nearly
next
not
now
of
off
on
only
onto
or
other
our
out
outside
over
overall
per
possibly
pt
put
really
regarding
reprinted
same
seen
several
should
shown
since
so-called
some
spp
studies
study
such
take
taken
takes
taking
than
that
the
their
them
then
there
therefrom
these
they
this
those
through
throughout
to
together
toward
towards
under
undergoing
up
upon
upward
various
versus
very
via
vol
vols
vs
was
way

ways
we
were
what
whats
when
where
which
while
whither
who
whom
whos
whose
why
with
within
without
yet
you
your

Appendix 3-B: The analytical expression for computing the precision/recall breakeven point (PRBEP) based on the Probability of Relevance (PRR) algorithm

The Probability of Relevance (PRR) formula for computing the precision/recall breakeven point is derived in Raghavan et al. (1989). Only a short summary is presented here.

Elaborating on the discussion in Section 3.7, assume that the threshold for predicting "relevant" is initially set larger than any value in the sorted list of test articles, and that this threshold is then lowered so that it is placed successively between each pair of adjacent values in this sorted list. In most cases, the list will actually consist of clusters of the same value, so lowering the threshold by one step actually moves past several test articles instead of only one. Let the numbers of relevant and irrelevant articles above the threshold be represented by $n_{r,prev}$ and $n_{\bar{r},prev}$, respectively, and the numbers of relevant and irrelevant articles in the cluster, C, directly below the threshold be represented by n_r and $n_{\bar{r}}$, respectively. If the precision is greater than the recall when the threshold is above C and visa versa when the threshold is below C, and if N_r denotes the total number of relevant articles in the test data, then the PRR formula for computing the breakeven point where the precision, P, is equal to the recall, R, can be derived from Theorem 3.5 in Raghavan et al. (1989). The result is:

$$P = R = \frac{N_r - n_{\bar{r},prev} + \frac{n_{r,prev}}{n_r} n_{\bar{r}}}{N_r \left(\frac{n_{\bar{r}}}{n_r} + 1\right)}$$

Chapter 4

The design of the autonomous agent Poirot

4.0. Introduction

The primary goal of the research presented in this thesis was to develop an autonomous computer program that can help World Wide Web users stay up to date on any topic of interest.[1] The resulting system is called Poirot because in a sense, it acts like a dedicated private investigator. It is designed to repeatedly search the World Wide Web and interact with the user in order to learn why particular web pages are relevant and others are not. A block diagram of the system is shown in Figure 4.1.[2]

4.1. An example of a brief session with Poirot

The simplest way to explain how the user can interact with Poirot is to provide an example. Figures 4.2 through 4.7 show screen shots from an actual session with Poirot. The goal of the session was to find web pages related to Dr. Rodney Goodman and his research at the California Institute of Technology. In order to make the task challenging, the user asked Poirot to search for web pages containing only the word "Goodman," as illustrated in Figure 4.2. After Poirot displayed the initial search results depicted in Figure 4.3, the user studied a few of the web pages and rated them as either relevant (+) or irrelevant (–), as shown in Figure 4.4. Figure 4.5 shows the window after the user requested that Poirot train its classifier on the rated pages. Note that in this case, only the relevant pages received a high score from the re-

[1] As an example, medical doctors might use the program to keep track of new developments in their field, including both legitimate new treatments that might be recommended to their patients and drugs that should be avoided. Poirot's autonomy should be particularly valuable in this case because using it would free physicians from having to search the Web manually and thereby allow them to spend more time with their patients.

[2] The software currently runs only on UNIX systems. The algorithms are not system dependent, however, so the only obstacle to implementing Poirot on Macintosh and Windows systems is the effort required to translate the source code.

sulting classifier. The words chosen by Poirot for use in the classifier are shown in Figure 4.6. The user subsequently requested that Poirot search for web pages matching these new words. The results of this search are displayed in Figure 4.7. Note that the top ranked pages are all relevant to Dr. Goodman's research, i.e., his projects, his collaborators, and the Caltech Electrical Engineering Department. It is also important to note that these pages are not limited to Dr. Goodman personally, i.e., that the results are much broader than the initial keyword "Goodman."

4.1.1. General description of the user's interaction with Poirot

Before Poirot can start on a new topic, it must be provided with keywords pertaining to that topic, as illustrated in Figure 4.2. Poirot sends the keywords to several search engines[3] in order to obtain web pages for the user to rate. If the user knows of other relevant web pages, they can be added manually.

Once Poirot has downloaded the web pages, it displays a list of their titles in a separate window devoted to the topic. An example is shown in Figure 4.3. The user can double click on an item in the list to display the corresponding web page in a web browser. To indicate that a page is or is not relevant, the user must rate the page as + or −, respectively, in Poirot's window, as illustrated in Figure 4.4. If the user follows a link from one of these pages and finds another relevant page, this can be added manually. A third designation, "Index," denoted by  , is also provided to distinguish pages that consist primarily of links to other pages. Instead of rating these index pages, Poirot uses

[3] Several search engines are required because each engine covers only a small fraction of the web (Selberg and Etzioni, 1995; Lawrence and Giles, 1998).

them as starting points to search for additional relevant pages. The use of the Index designation is demonstrated in Figure 4.7.

The next step is for Poirot to train a classifier on the pages that have been rated. Simultaneously, it also follows the links from the rated web pages and index pages in search of other relevant pages. Once the training is complete, Poirot uses the words from the classifier to retrieve more pages from the web search engines. Finally, Poirot rates all the newly discovered pages.

Since downloading and analyzing many web pages can potentially take several hours, the process is usually performed at night so the results will be available to the user the next morning. Since there is less traffic on the Internet at night, this strategy has the additional benefit that the downloading will require less computer time than during the day. The next morning, when the user starts Poirot again, it presents the list of all the newly rated pages so the user can study them and then provide more feedback. An example of the results is shown in Figure 4.7.

The process outlined above can be repeated indefinitely. The classifier is only retrained if the user changes any of the ratings or adds new pages. The web search engines are only contacted if the list of words used by the classifier changes significantly, i.e., more than 10% of the words are different, or if they haven't been contacted within a user specified interval, e.g., one month. In addition, known web pages are periodically retrieved and checked for changes. As discussed in Section 4.3, a separate rating indicates whether or not the changes are important.

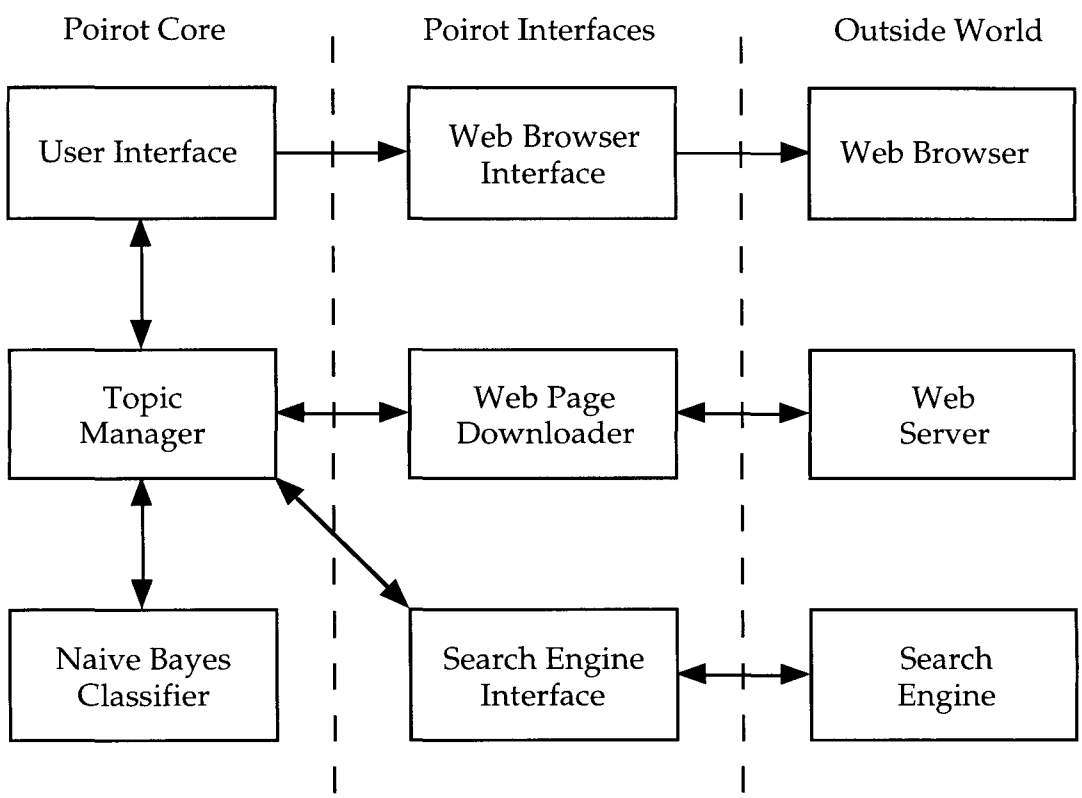


Figure 4.1: System block diagram for the autonomous agent called Poirot

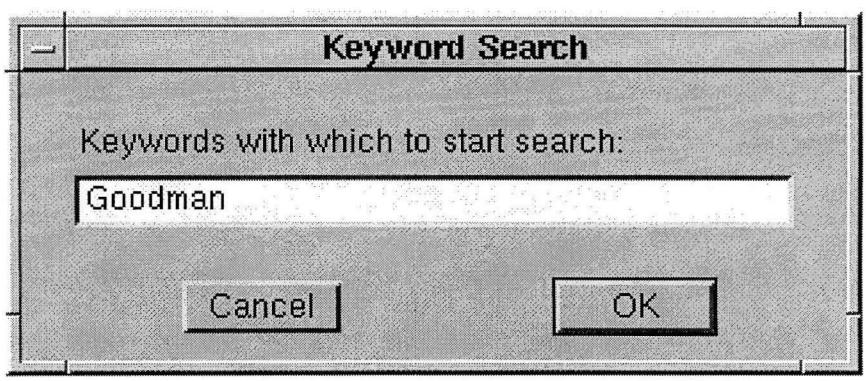


Figure 4.2: Screen shot from Poirot showing the dialog window where the user enters one or more initial keywords describing a topic of interest

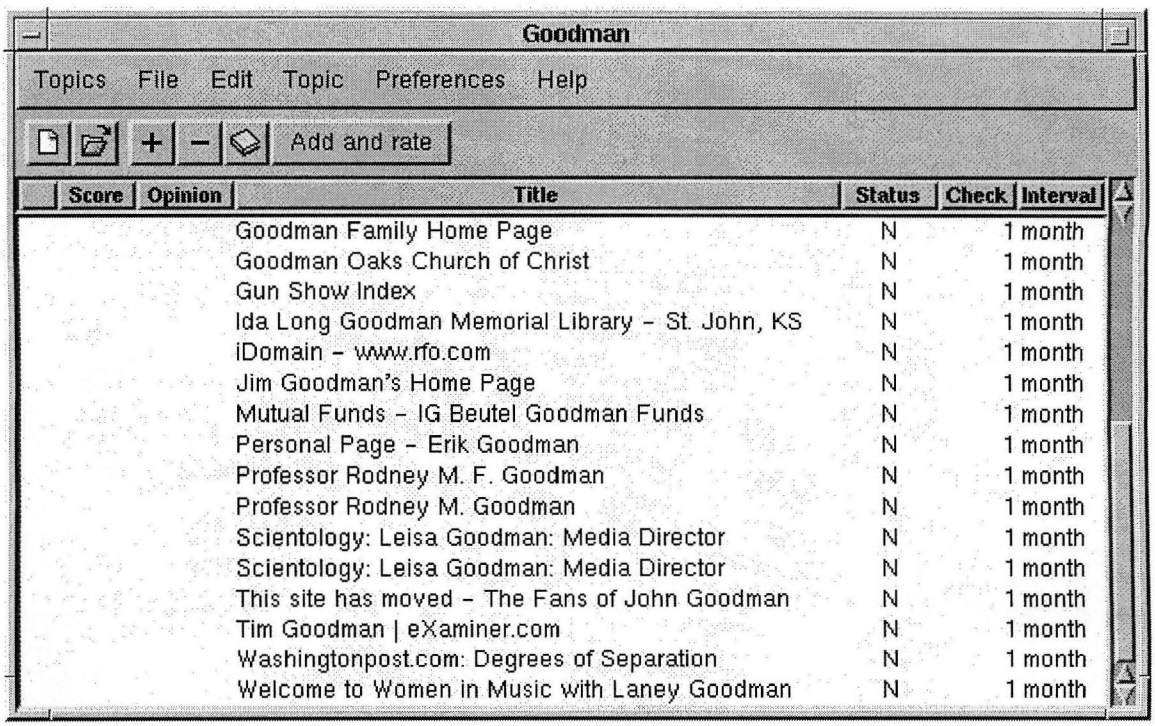


Figure 4.3: Screen shot showing the results of Poirot's initial search for web pages containing the word "Goodman." The pages are listed in alphabetical order by title. The "N" in the Status column stands for "New" and indicates that the user has not yet viewed the page with a web browser. The values in the "Check" and "Interval" columns specify how often Poirot should check for changes in each web page.

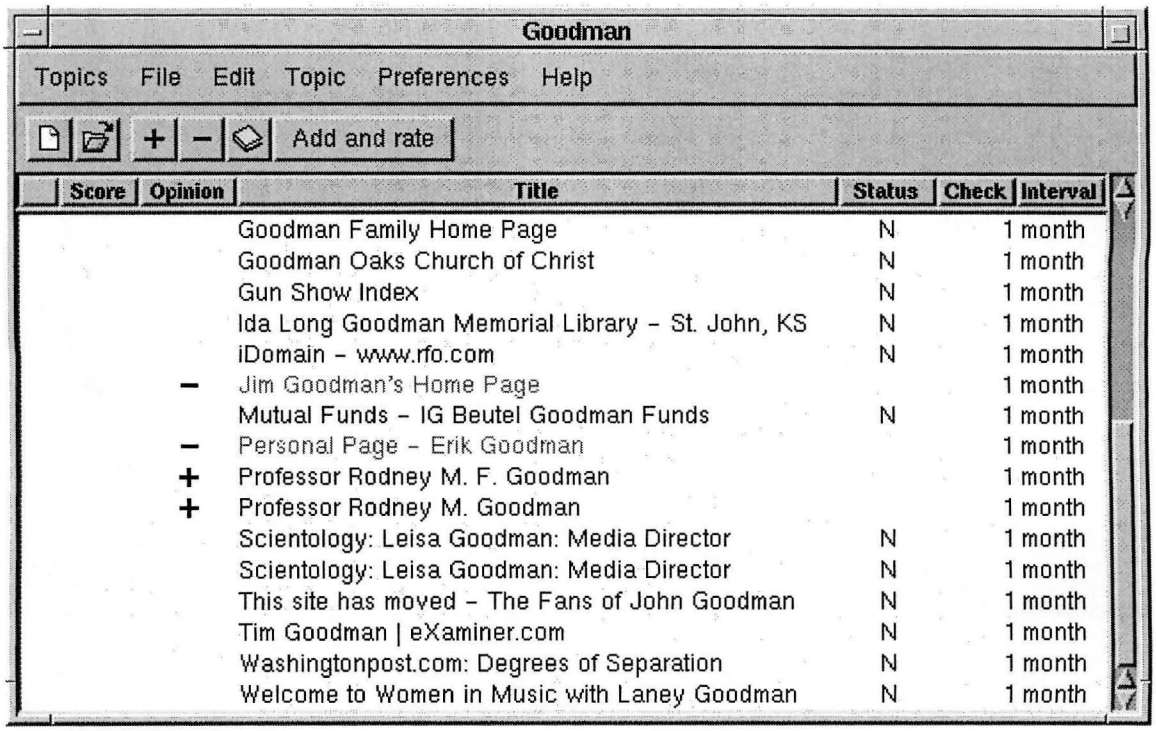


Figure 4.4: Screen shot from Poirot after the user has studied and rated a few of the web pages by placing + and – symbols in the Opinion column

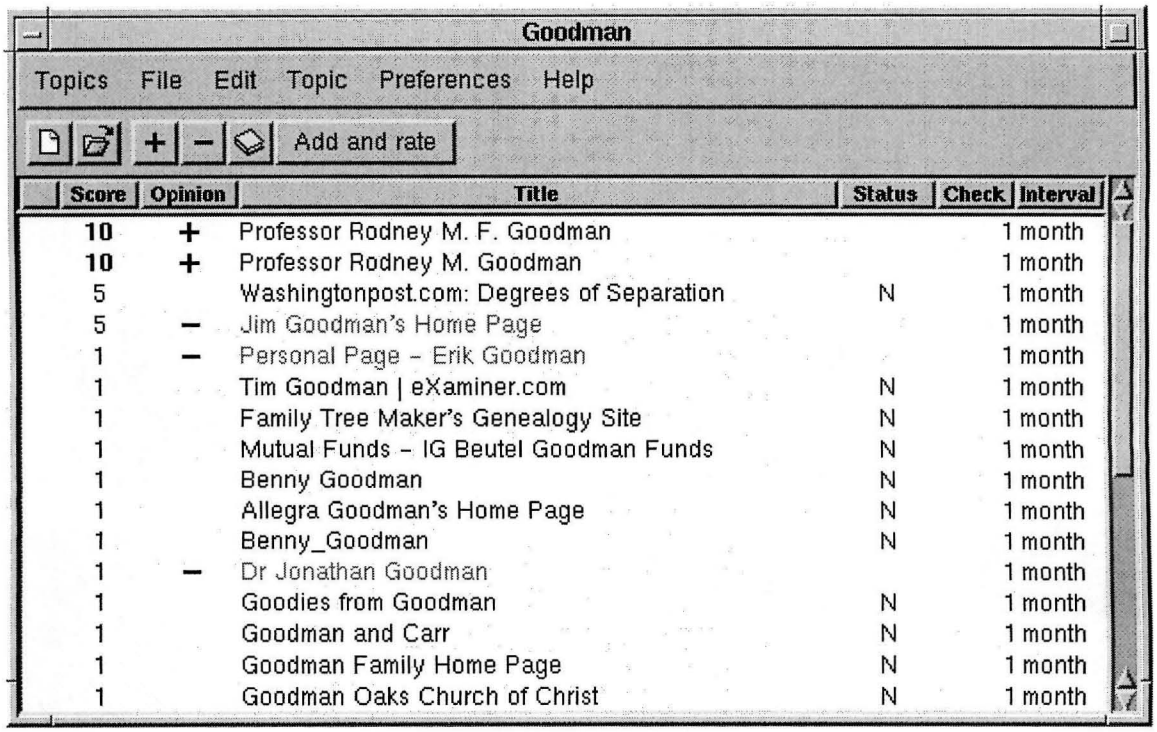


Figure 4.5: Screen shot from Poirot after it has constructed a classifier based on the ratings provided by the user in the Opinion column. The words chosen by the classifier are shown in Figure 4.6. The numeric values in the Score column are the output from the classifier. The pages are sorted in descending order of this rating. Note that the pages rated + by the user have the highest numeric rating, and that the pages rated – by the user have a lower numeric rating. As explained in Section 4.2.1, a numeric rating greater than five indicates that the web page is relevant.

```
expected accuracy (approximate): 100%

prior p(relevant) = 0.103448

"electrical engineering"
if not there:  p(relevant) = 0.0359801
if     there:  p(relevant) = 0.935484

"information processing"
if not there:  p(relevant) = 0.0359801
if     there:  p(relevant) = 0.935484

"microsystems"
if not there:  p(relevant) = 0.0359801
if     there:  p(relevant) = 0.935484

"rodney"
if not there:  p(relevant) = 0.0359801
if     there:  p(relevant) = 0.935484

"rodney goodman"
if not there:  p(relevant) = 0.0359801
if     there:  p(relevant) = 0.935484

"signal"
if not there:  p(relevant) = 0.0359801
if     there:  p(relevant) = 0.935484

"signal processing"
if not there:  p(relevant) = 0.0359801
if     there:  p(relevant) = 0.935484

"vlsi"
if not there:  p(relevant) = 0.0359801
if     there:  p(relevant) = 0.935484

"information"
if not there:  p(relevant) = 0.0584677
if     there:  p(relevant) = 0.215881

"work"
if not there:  p(relevant) = 0.0467742
if     there:  p(relevant) = 0.311828

"research"
if not there:  p(relevant) = 0.0550285
if     there:  p(relevant) = 0.233871
```

Figure 4.6: Output from Poirot showing all the words and phrases used by the classifier that was constructed from the user's feedback displayed in Figure 4.4. Also shown are the probabilities that a web page is relevant if the word or phrase enclosed in quotation marks is present or absent.

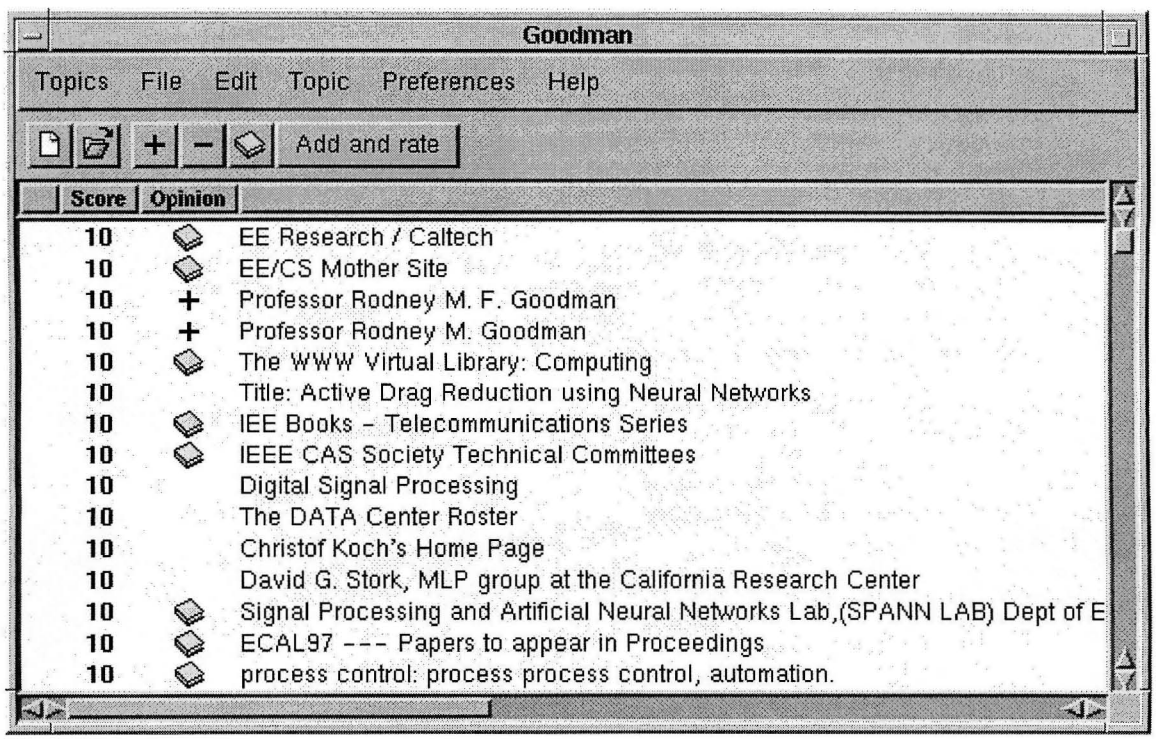


Figure 4.7: Screen shot showing the results of Poirot's second search for web pages. This search used the words and phrases from the classifier shown in Figure 4.6 instead of the original keywords. The numeric values in the Score column are the output from Poirot's classifier. Note that both pages which the user rated as + are near the top of the list, and that all the pages that are shown pertain to Dr. Goodman's research interests. Also, note that the user has marked several of the items as index pages. These provide excellent starting points for further exploration of the topic.

4.1.2. A note on obtaining feedback from the user

In order to minimize the number of questions that the user must answer, some systems have been designed to use indirect methods of measuring the relevance of a web page. One popular approach is to assume that the relevance is proportional to the amount of time that the user spends studying the page (Morita and Shinoda, 1994; Voigt, 1995; Mladenic, 1996; Nichols, 1998). However, since Poirot does not run inside web browsers, there is no way to measure this time interval. Moreover, the question of how the time is spent cannot be answered without actually asking the user. The user might for instance have been distracted or simply decided to take a break instead of studying the web page. If the user did actually spend the time studying the page, however, it might simply have taken a while to decide that it was not relevant, or the user might have spent the time because it was relevant to a different topic. Conversely, if the user does not spend much time on a page, it might still be relevant, but the user might have decided to follow an interesting link near the top of the page. These uncertainties add an unacceptable amount of noise to the training data. Manual rating by the user is therefore the most reliable approach. It should be noted, however, that if an accurate method of obtaining implicit feedback can be found, then the NB-CV algorithm will be able to use this information to construct an accurate classifier.

4.2. Description of Poirot's page rating algorithm

In order to simplify the problem, Poirot's learning algorithm is designed to work on only one topic at a time. Web pages that previously have been rated by the user as germane to other topics are utilized as additional negative training examples for the current topic. This approach eliminates the need to maintain a list of irrelevant web pages for each topic.

The experimental results presented in the previous chapter show that the accuracies of the NB-CV, ITRule CV-J, and SVM algorithms are essentially identical. Moreover, all three perform significantly better than the other algorithms that were tested. Thus, from this stand point, one could use any of them in Poirot.[4] However, as explained in Section 3.4.5, the implementation of the SVM algorithm is very slow because cross validation is performed by a separate program. The current implementation of the ITRule CV-J algorithm is also slow because it is disk-based in order to allow it to be used on extremely large data sets. Rewriting either the SVM or the ITRule software would require a considerable amount of work. Thus, since Poirot only needs to handle relatively small data sets, it uses the much faster NB-CV implementation that was employed during the experiments discussed in Chapter 3.

In the web pages that the user has rated, all single words and all two and three word phrases are potential input features. The classifier uses the presence or absence of the words and phrases chosen by cross validation to decide whether or not a new page is likely to be relevant or irrelevant. Depending on the probabilities that are calculated from the training data, the presence of a word or phrase may increase or decrease the likelihood of relevancy.

[4] As discussed in Section 3.4.2, NB-CV and ITRule CV-J can be considered *variants of the* same algorithm.

It should be noted that it is the NB-CV algorithm which makes the final decision concerning which words and phrases to use. Thus, the classifier is not constrained to use the keywords initially provided by the user[5] since there is no guarantee that these words are the best features for discriminating between relevant and irrelevant pages. In some rare situations, the initial keywords might not even occur in the pages that the user has rated, in which case no statistics could even be computed. However, in practice, the keywords will almost always occur in the rated pages. Thus, the keywords will usually be included in the list of potential input features.

Poirot's learning algorithm only considers the unformatted text of the web page, i.e., fonts, styles, and paragraph breaks are ignored. Some studies have attempted to use formatting to select the important words from each web page (Krulwich and Burkey, 1997). There are two reasons why this approach is not used in Poirot. First, formatting tends to vary widely between different web sites and is non-existent in plain text (.txt) web pages. Second, rigorous statistical analysis is a far more robust method of choosing words. Headlines and other emphasized words are only meant to provide visual cues while the user is reading the web page. A word may be emphasized for a wide variety of reasons including such trivial situations as italicizing "and" between two statements that are related in an unexpected way. Thus, there is no guarantee that such words are correlated with the reader's judgment of relevance. At best, they may correlate with the author's interests. A direction for future research might be to explore the possibility of using emphasis to give particular words more weight by, for example, treating them as if they

[5] As demonstrated by the example session in Section 4.1, this can be very important for serendipity, i.e., making fortuitous, relevant discoveries beyond the scope of the original intention and related keywords.

had been repeated several times, with the number of repetitions proportional to the level of emphasis. However, since none of the classifiers that were tested in Chapter 3 were able to use word frequency effectively, and since the Reuters-21578 data set which was used for most of the experiments did not contain any formatting information, this issue was not explored.

Poirot's learning algorithm also ignores the images on a web page. Analyzing the images is beyond the scope of this thesis because it is an entirely different field of research. At the present time, the field is considered to be wide open, and no proven algorithm is available for use in Poirot. However, ignoring images is not a serious deficiency as long as the web page includes captions or other text describing the images.

4.2.1. Computing the rating displayed in the Score column

Misclassifying a relevant page is normally considered to be more serious than misclassifying an irrelevant page. Poirot is therefore designed to minimize the amount of trouble caused by its mistakes rather than merely minimizing the number of mistakes.[6] Mathematically, one can define the amount of trouble caused by misclassifying an irrelevant page to be one and then use the positive parameter C to represent the amount of trouble caused by misclassifying a relevant page. Under the assumption that the probabilities produced by the classifier are correct, the expected amount of trouble that will be caused by predicting "relevant" is $0 \cdot p(r \mid page) + 1 \cdot p(\bar{r} \mid page)$, while the expected amount of trouble that will be caused by predicting "irrelevant" is $C \cdot p(r \mid page) + 0 \cdot p(\bar{r} \mid page)$. Here, $p(r \mid page)$ represents the probability of the

[6] This is a special case of the general concept of risk minimization discussed by Duda and Hart (1973).

web page being relevant, while $p(\bar{r}\,|\,page)$ represents the probability of the page being irrelevant. Poirot minimizes the expected amount of trouble by predicting "relevant" when this decision will cause less trouble, i.e., when

$$p(\bar{r}\,|\,page) \;<\; C\,p(r\,|\,page)$$

This formula is intuitively reasonable because when the value of C is increased, a larger range of the output probability, $p(r\,|\,page)$, is mapped to the decision "relevant." Thus, the page is only classified as irrelevant if this is overwhelmingly likely. Based on the results presented in Section 3.5.3, C was set equal to two in Poirot.

Since C is not equal to one, the breakpoint between "relevant" and "not relevant" does not occur at the midpoint of the classifier's output, i.e., not where $p(r\,|\,page)$ is equal to $p(\bar{r}\,|\,page)$. In order to compute a symmetric rating between zero and ten that can be displayed to the user, Poirot first defines the function:

$$R_1 \;=\; C\,p(r\,|\,page) - p(\bar{r}\,|\,page)$$

A web page is considered to be relevant when the value of this function is greater than zero, i.e., when $p(r\,|\,page) > 1/(C+1)$. Since the value of R_1 lies in the interval $[-1,C]$, an intermediate function, R_2, is introduced to shift and scale the value R_1 so that the result lies in the interval $[0,1]$:

$$R_2 \;=\; \frac{R_1 + 1}{C + 1} \;=\; \frac{\left(C\,p(r\,|\,page) - p(\bar{r}\,|\,page)\right) + 1}{C + 1} \;=\; p(r\,|\,page)$$

If one defines n=C+1, then the breakpoint between "relevant" and "not relevant" occurs when $R_2 = 1/n$. In order to shift this breakpoint to five and achieve symmetry in the interval $[0, 10]$, a parabola was fitted to the three points $(0, 0)$, $(1/n, 5)$, and $(1, 10)$. This provides a smooth transformation of

the interval [0, 1] to [0, 10]. The resulting rating function is:

$$R = 10\left(\frac{n\ (2-n)\ R_2^2 + (n^2-2)\ R_2}{2\ (n-1)}\right) = \frac{5}{2}[7 - 3\ p(r \mid page)]\ p(r \mid page)$$

The rounded off value of R is the page rating displayed to the user in the Score column, as illustrated in Figures 4.5 and 4.7. At the breakpoint between "relevant" and "not relevant," R=5, and $p(r \mid page)$=1/3.

4.3. Reporting significant changes to a web page since it was last visited

The previous sections have dealt with the problem of how to rate a newly discovered web page. These are the ratings displayed in the Score column in Figures 4.5 and 4.7. Since the World Wide Web is extremly dynamic, Poirot must also periodically revisit web pages to check for significant changes, i.e., addition, deletion, or modification of relevant information. However, insignificant modifications such as changes in font or spelling and grammar corrections should not be reported. In order to filter out trivial alterations, the changes to the web page are given a rating, R_{change}. Note that this change rating is different from the rating assigned to the entire page. When R_{change} is non-zero, it is displayed in the Status column using the transformation discussed in Section 4.2.1, as illustrated in Figure 4.8.[7]

The change in the rating of the entire page, R_{new}–R_{old}, is usually the most important contribution to R_{change}. However, significant additions and deletions must also be taken into account because it is possible for R_{new} and

[7] The number of new links on the web page to other web pages, images, PostScript files, etc., is also often of interest, but for different reasons, so this is displayed separately in the Status column, as illustrated in Figure 4.8.

R_{old} to be nearly equal.[8] Poirot therefore computes R_{change} as the maximum of the change in the rating of the entire page, the rating of the text that was added, if any, and the rating of the text that was removed, if any[9]:

$$R_{change} = \max\left(|R_{new} - R_{old}|,\ w\left(\frac{\Delta N_{add}}{N_{new}}\right) R_{add},\ w\left(\frac{\Delta N_{del}}{N_{old}}\right) R_{del} \right)$$

Here, R_{new} is the rating of the new version of the page, R_{old} is the rating of the old version of the page, R_{add} is the rating of the added text, and R_{del} is the rating of the deleted text. The "max" function is used because each of the three terms represents a separate reason for the user to re-read the web page, and one good reason is enough to warrant bringing the page to the user's attention. The absolute value is used in the first term because a large decrease in relevance may be just as important as a large increase.

The function w is a weighing function that reduces the effect of R_{add} and R_{del} when the number of words that were added to the new version, ΔN_{add}, is small relative to the total number of words in the new version, N_{new}, or when the number of words that were deleted from the old version, ΔN_{del}, is small relative to the total number of words in the old version, N_{old}, respectively. As an example of why this is appropriate, the addition of a sentence to a long thesis is probably not nearly as important as the addition of a paragraph to a short abstract. Since additions and deletions may be equally inter-

[8] Two examples of a change that does not affect the overall rating are (1) significant additions to a page that already has a high rating and (2) significant removals that leave behind enough keywords so that the rating remains high.

[9] The current implementation of Poirot uses the UNIX utility program called "diff" to find the text that was added or removed. A paragraph formatted version of the text is compared, not the raw Hypertext Mark-up Language (HTML) source code. When "diff" reports that a paragraph has changed, this is treated as an addition if the new text is more than twice the length of the old text, and a removal if the old text is more than twice the length of the new text.

esting, the weights for R_{add} and R_{del} have the same form. The functional form that Poirot uses for w is:

$$w(x) = \begin{cases} 1 - \dfrac{(x - x_0)^2}{x_0^2} & 0 \le x < x_0 \\ 1 & x_0 \le x \le 1 \end{cases}$$

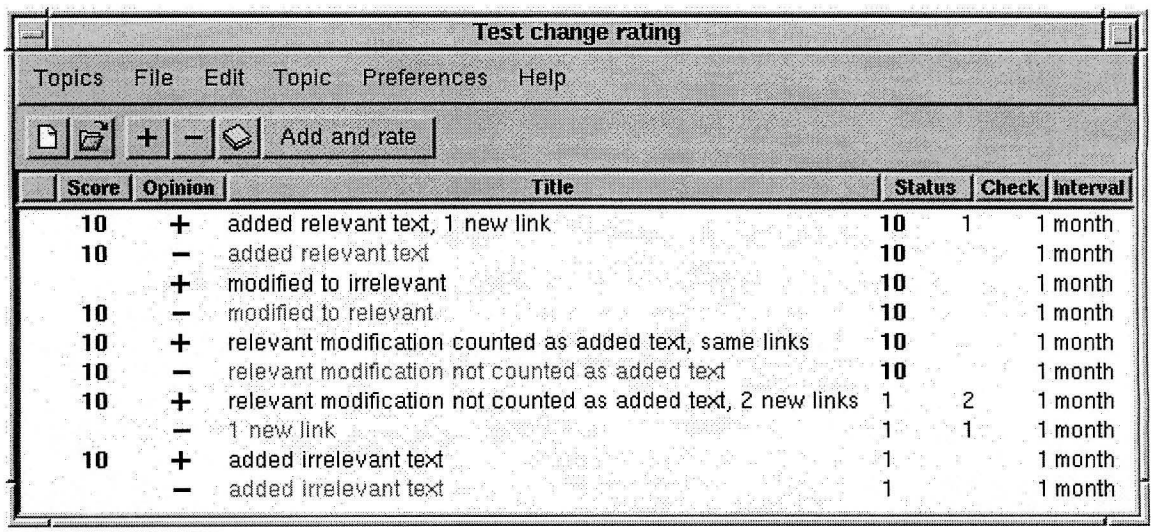
The value of x_0 is set to 0.1 so that R_{add} and R_{del} are only reduced if the total size of the additions or deletions is less than 10% of the total length of the web page.

Unfortunately, this method of computing R_{change} cannot detect all significant changes. As an example, the addition or removal of the word "not" can be quite significant, but since "not" is a so called stop word that Poirot ignores, this change will never be considered significant.[10] More generally, it is often possible to rewrite a paragraph so that it contains the same keywords but says something very different. Without a complete understanding of the text, such changes cannot be correctly labeled as significant. The only solution appears to be to display R_{change} for all pages that have changed and to warn the user that a low value does not guarantee that the changes are not significant. This is therefore the approach used in the design of Poirot.

It is also worth noting that web pages may change at different rates. Some pages change suddenly because the maintainer decides to restructure, rewrite, or even replace the entire text. Other pages change gradually as the maintainer adds or modifies information bit by bit over a long period of time.

[10] One could check for the special case of "not" occurring in front of a keyword, but there are two problems with this approach. First, English is flexible enough to allow one to obtain the same meaning by placing "not" somewhere else in the sentence. The second and more serious problem is that one will never run out of special cases that need to be fixed. This endless spiral leads inevitably to the topic of Natural Language Processing (NLP). However, since this is a very different research problem, it was deemed to be beyond the scope of this thesis.

In order to correctly report the cumulative effect of gradual changes, Poirot calculates R_{change} based on the changes from the user's last visit instead of Poirot's last visit.[11] In this way, a page that repeatedly undergoes minor changes may eventually acquire a large R_{change} and thus rise to the top of the list where it will be noticed by the user.



Score	Opinion	Title	Status	Check	Interval
10	+	added relevant text, 1 new link	10	1	1 month
10	−	added relevant text	10		1 month
	+	modified to irrelevant	10		1 month
10	−	modified to relevant	10		1 month
10	+	relevant modification counted as added text, same links	10		1 month
10	−	relevant modification not counted as added text	10		1 month
10	+	relevant modification not counted as added text, 2 new links	1	2	1 month
	−	1 new link	1	1	1 month
10	+	added irrelevant text	1		1 month
	−	added irrelevant text	1		1 month

Figure 4.8: Screen shot from Poirot showing how the rating, R_{change}, which is assigned to the changes in each web page is displayed on the left-hand side of the Status column. R_{change} is different from the rating assigned to the entire page which is displayed in the Score column. Note that the number of new links is displayed separately on the right-hand side of the Status column. Also note that the pages are sorted by the value of R_{change}, rather than by the main rating displayed in the Score column.

Unlike the other screen shots, this one does use actual web pages. Since it is not possible to modify web pages created by somebody else, it was necessary to create a set of simple test pages. The Title column in the above screen shot is used to explain the changes that were made to each test page so that the values displayed in the Status column can be interpreted correctly.

[11] The user is assumed to have visited a page if he double clicks on the page in Poirot's list, and at least 30 seconds elapse before he double clicks on another page in the list.

4.4. Sharing relevant pages with other Poirot users via index pages

The previous sections have discussed the problem of filtering the results returned by web search engines. Filtering is necessary because search engines typically return more irrelevant pages than relevant pages. If one instead were able to search web pages discovered by other users with similar interests, then the signal-to-noise ratio would presumably be much higher since more pages would be relevant.

One challenge associated with this method is to disseminate each user's discoveries. Poirot provides a very simple, yet powerful mechanism to accomplish this task. For each topic, Poirot creates an index page ranking all the discovered web pages. Other Poirot users may find this topic index page via a web search engine or by personal communication. If they add the page to their own lists of web pages on the topic and designate the page as an index page, i.e.  , Poirot will automatically rate all the pages mentioned and watch for the addition of links to other pages.[12]

In order to make it easy for search engines to find topic index pages created by Poirot, the program generates a master index web page that contains links to all the individual topic index pages. A user only has to add a single link from his home page to this master index page in order to provide full access to all the topics.[13]

[12] Poirot "signs" the index pages that it creates by including its name in the meta information, as illustrated in Figure 4.9. This allows Poirot to recognize these pages when they are returned by a search engine. Poirot can then automatically label the pages as index pages instead of having to wait for the user to study them.

[13] For this method to work, the user's home page must be easily locatable by search engines. Poirot cannot enforce this, but it is not a problem because users normally have a *strong incentive* to ensure that the requirement is satisfied.

The likelihood that a search engine will return the topic index page in response to a query from another Poirot user is improved by explicitly including the words employed by the classifier on the index page via the special "meta" tag, as illustrated in Figure 4.9. Most search engines give significant weight to such words. In addition, on each index page, Poirot uses the title of each listed web page to anchor the link to the actual web page, as depicted in Figures 4.9 and 4.10. These titles often include words that are strongly correlated with the topic, thereby further increasing the probability that the index page will be returned during a search.

In addition to the improvement in the signal-to-noise ratio, there is another reason why using an index page created by Poirot instead of the results from a search engine can significantly improve the accuracy of Poirot's ratings. In general, the meaning of a word depends on the context in which it is used. This is the primary theoretical objection to computing ratings based only on the presence or absence of words.[14] Including phrases helps somewhat since they tend to have fewer possible meanings. However, the best way to ensure that a word has a particular meaning is to restrict the topic of discussion. Utilizing index pages created by Poirot users with similar interests enforces this constraint. Since the words used by the classifier are likely to have the desired meaning, Poirot is less likely to make mistakes in this case than when evaluating web pages returned from a keyword search of all the web pages known to a search engine.

[14] The results presented in Chapter 3 indicate that this objection is apparently not a serious problem in practice.

Poirot's method of sharing discoveries can also aid serendipity. Each user will have a unique approach to a given topic which will be evident from the contents of the index page. By presenting the union of these index pages, Poirot may provide some users with ideas that would not have occurred to them if they instead had worked in isolation.

Previous research has estimated the relevance of an item to a given user directly from the ratings assigned to the item by other users (Shardanand and Maes, 1995; Alspector et al., 1997; Billsus and Pazzani, 1998; Herlocker et al., 1999). This approach is called collaborative filtering. It requires information about how well matched the interests of the users are. If there is little overlap between the subsets of items that each user has rated, as is very likely in the case of web pages since there is effectively an unlimited number of them, then this may degenerate into pure guessing.[15] In contrast, Poirot's approach ensures that all ratings are computed from each individual user's classifier, thereby eliminating the possibility of not having any information from which to calculate a rating. This is very similar to the approach used in the Do-I-Care system (Starr et al., 1996), except that they only disseminated the index pages through direct personal communication, not via existing web search engines. Poirot's method is therefore more efficient because it requires negligible effort to reach everybody on the Internet.

[15] Most collaborative filtering systems are designed to work in a very restricted domain, e.g., music, movies, or single topic Usenet newsgroups. In such restricted domains, it is much easier to find several users who have rated the same items.

```
<html>

<head>
<title> Index Page for "Goodman" </title>
<meta name="description" content="Poirot Index Page">
<meta name="keywords" content="electrical engineering, infor-
mation processing, microsystems, rodney, rodney goodman, sig-
nal, signal processing, vlsi">
</head>

<body>

<h3> Index Page for "Goodman" </h3>

<a href=http://www.micro.caltech.edu/micro/goodman/>Professor
Rodney M. Goodman</a>

<p>
<a
href=http://www.cns.caltech.edu/Faculty/Goodman.html>Professor
Rodney M. F. Goodman</a>

</body>
</html>
```

Figure 4.9: Hypertext Mark-up Language (HTML) source code for the topic index page generated by Poirot from the topic "Goodman" shown in Figure 4.5. The "meta" information near the top specifies the keywords that are relevant to the topic so that web search engines can exploit them.

Index Page for "Goodman"

Professor Rodney M. Goodman

Professor Rodney M. F. Goodman

Figure 4.10: The result of displaying the source code from Figure 4.9 in a web browser. The meta information is not displayed but is taken into account by search engines. Underlining indicates that the text is the anchor for a link to another page. The Universal Resource Locator (URL) for the page comes from the hypertext reference (href) specification in the source code of Figure 4.9. The URL is not displayed, but when the user clicks on the text with the mouse cursor, the web browser will display the page.

4.5. Miscellaneous features provided by Poirot

4.5.1. User interface design

Poirot's user interface is designed with the idea in mind that all relevant information should be available to the user. However, in order to prevent the user from feeling overwhelmed, it should also be easy to ignore as much as possible. The various ratings and other information provided by Poirot are therefore discreetly placed in peripheral columns in the windows, as illustrated in Figure 4.8. Furthermore, the user can choose to have Poirot sort the web pages by either the main rating, which is displayed in the Score column (cfr. Figure 4.5), or the change rating, which is displayed in the Status column (cfr. Figure 4.8). In this way, one does not have to manually inspect the values. Moreover, the values that indicate "relevant" are displayed in bold so it is easier to find the cutoff between relevant and irrelevant pages.[16] In addition, information about the performance of the learning algorithm, such as the words that are used and the expected probability of correctly rating new web pages, is available on demand in separate windows, as shown in Figure 4.6.

4.5.2. Avoiding loss of information

Web pages are sometimes deleted or moved to different locations. When this happens, Poirot displays an X in the Status column to indicate that the page is unavailable. The information is not lost, however, because

[16] Since the data used in Chapter 3 only provides a Boolean measure of relevancy, it is not certain that a lower rating is less relevant than a higher rating, e.g., that a rating of eight will be less relevant than a rating of ten. However, since the web pages are displayed in a list and must therefore be presented in some particular order, sorting in descending order of rating seems least likely to confuse the user.

the content of every relevant page that Poirot has located is stored in a cache. Poirot provides the option to view the cached version.

Poirot also provides the option to search through the cache for relevant information[17], e.g., particular names, statistics, or quotations. This approach requires both less time and effort than using a web search engine. The reason is that only relevant web pages are searched, not everything that would be returned by a web search engine. This is especially helpful when the set of pages grows very large because, in this case, one would otherwise be faced with the same problem as at the start, namely searching a large number of web pages for specific information. If the desired information is not found among the cached pages, Poirot automatically offers to create a new topic and start searching the Web.

4.5.3. Benefiting from newly available search engines

New search engines are constantly becoming available. As indicated in the block diagram in Figure 4.1, the modular design of Poirot makes it easy to provide support for additional search engines. The interface to a search engine is simply the Universal Resource Locator (URL) for initiating the search, the URL for continuing the search, and a regular expression (regex) for extracting the web page links from the results returned by the search engine.

[17] Poirot thus subsumes the service provided by Backflip, `http://www.backflip.com/`, a web site that lets each user store a personalized set of links to web pages and includes the ability to search the contents of these pages.

4.6. Results of initial user testing

Several users have tested Poirot on topics of personal interest. All of them have reported that Poirot successfully located many relevant pages, just as it did in the example session documented in Section 4.1. One user stated that the quality of Poirot's results was significantly better than that of his favorite web search engine. Another user reported that the focus of his interest tended to drift as he studied the web pages discovered by Poirot, and that the program successfully tracked this drift as it interacted with him. Moreover, all users agreed that Poirot often found unexpectedly relevant pages. This helped them both improve their understanding of the topic and broaden their search for more relevant web pages.

4.7. Comparison with other systems

Poirot differs from systems such as Letizia (Lieberman, 1995), LIRA (Balabanovic et al., 1995), and WebWatcher (Armstrong et al., 1995) which were designed to supply immediate answers to individual questions while the user is browsing the Web rather than providing continuing support for topics of long term interest.[18] The "Syskill & Webert" system (Pazzani and Billsus, 1997), on the other hand, resembles Poirot more closely. It was also designed to support the user's long term interests. However, as with the previously mentioned systems, "Syskill & Webert" only provides suggestions while the user browses, and does not independently search the Web for additional, relevant pages while the user is occupied with other tasks.

[18] The system called Watson (Budzik and Hammond, 1999) takes this goal to the extreme by providing immediate answers via what the authors refer to as "just-in-time information retrieval system." It analyzes the text that the user is editing in a word processor and automatically searches for relevant web pages.

Furthermore, "Syskill & Webert" uses the NB-96 algorithm which has been shown to perform quite poorly (see Section 3.5).

The project with design goals closest to that of Poirot seems to be EUROgatherer (Amato et al., 2000). Unfortunately, at this point in time[19], there does not appear to be any published information on its algorithms or its performance, and the software is not available, so a quantitative comparison with Poirot cannot be made.

Another system that deserves mention is WebACE (Boley et al., 1999). Although the authors do not directly discuss the issue, it appears that it might be possible to use this system to track topics of long term interest since it is designed to first group web pages into related clusters and then search for more web pages to add to each cluster. However, WebACE uses an unsupervised learning algorithm. Thus, even though it does group web pages into a small number of clusters, it can be difficult to determine what topic each cluster represents, in contrast to Poirot where each topic is defined explicitly by the user.

Poirot also differs from most other systems by being able to decide whether or not changes made to previously located web pages are important. There are services such as URL-minder[20] that will send the user email when a specific page changes. However, none of these systems are able to determine why the user considers the web page to be relevant in the first place. Thus, they are unable to evaluate whether or not a detected change is likely to be of interest. The user will therefore have to manually reread web pages with only insignificant changes such as spelling or grammatical corrections.

[19] Since the EUROgatherer project is no longer being funded (Amato, 2000), it seems unlikely that any more information will be made available in the future.

[20] `http://www.urlminder.com/`

The Do-I-Care system (Starr et al., 1996) appears to be the only other system that attempts to filter out such trivial and irrelevant changes.[21] However, unlike Poirot, Do-I-Care trains separate classifiers for new and changed web pages, thereby requiring considerably more feedback from the user. Furthermore, Do-I-Care provides only a single rating, while Poirot displays the number of new links separately from the rating for the changes to the text. In addition, the rating displayed by Do-I-Care is computed from only the additions to the web page since the program's last visit. This ignores the other two issues discussed in Section 4.3 that are accounted for in Poirot's formula for R_{change} and also the possibility that a page may infrequently change in small increments. Do-I-Care will therefore report only large, sudden changes to a web page and will miss the accumulation of smaller, gradual changes.

4.8. Suggestions for future work

4.8.1. Natural Language Processing (NLP)

One possible direction for future research is to attempt to improve Poirot's learning algorithm by using Natural Language Processing (NLP). Other researchers have addressed the problem of using NLP to determine the topic of a document, but these efforts have so far concentrated on restricted vocabularies or fixed sets of documents (Lewis et al., 1989; Jacobs and Rau, 1990). This might be sufficient, however, assuming that each topic is analyzed separately. The issue was not considered in this thesis because the NB-

[21] Amato et al. (2000) claims that EUROgatherer also does this, *but they do not provide* any details.

CV algorithm performs so well that there is not much room left for improvement. The uncertain prospect of only a small increase in accuracy does not seem to warrant the considerable effort required to implement an NLP system.

4.8.2. Using a thesaurus to expand the list of keywords

A more modest approach than implementing NLP would be to add an on-line thesaurus that could provide additional keywords for both the initial search and the nightly updates. Web pages that use synonyms of the user's initial keywords would then also be retrieved. However, since a thesaurus stores synonyms for all meanings of each word, one must be careful to pick the appropriate ones (Krovetz and Croft, 1992). Gauch and Smith (1991) have demonstrated that retrieval is most effective when using a combination of user supplied keywords, user selected terms from a thesaurus, and statistically relevant words from previously retrieved documents. Thus, Poirot should only use a thesaurus to suggest additional keywords, not to augment the keyword list automatically. Several on-line thesauri are available from `http://www-a2k.is.tokushima-u.ac.jp/member/kita/NLP/lex.html`.

4.8.3. Exploiting the links between web pages

Spertus (1997) has suggested that one might be able to determine the relevance of web pages from the connectivity graph of links between the pages. The issue was not considered in this thesis because it was beyond the scope of comparing classification algorithms that use the contents of each individual web page. It would be possible for Poirot to search for additional web pages by

using a special feature of the AltaVista web search engine[22] to retrieve web pages that link to known, relevant pages. However, it is not clear that this would yield any pages that would not be returned by direct searches. It is possible that it might provide a way to find index pages, but it seems very likely that these index pages would constitute a negligible fraction of all the pages that would be returned. If so, the results would not be worth the effort.

4.8.4. Extracting information from Usenet and mailing lists

Usenet and mailing lists have been suggested as possible sources of information. If one subscribed to specific newsgroups or mailing lists, Poirot could automatically scan every message for links to web pages and then test each page for relevance to each of the user's topics. Unfortunately, many users automatically include miscellaneous links in the "signature" (.sig) at the end of every message they send. Without sophisticated filters to detect and ignore these signatures, Poirot will likely be overwhelmed with irrelevant web pages. Developing such filters was deemed to be beyond the scope of this thesis.

It has also been suggested that Poirot could directly filter messages sent to Usenet newsgroups and mailing lists. This is likely to be far more difficult than filtering web pages, however, because topics of discussion may vary widely within a single newsgroup or mailing list and each topic is typically discussed for only a short time, after which interested users are politely referred to an archive somewhere on the Web. It therefore seems better to simply let Poirot extract relevant messages from these archives via the search engines that index them.

[22] http://www.altavista.com/

Chapter 5

Summary

Substantial improvements have been made to ITRule. Extensive tests conducted with the Reuters-21578 data set show that the new CV-J algorithm constructs a significantly more accurate classifier than the original MDL algorithm. In addition, ITRule has been extended in several ways to provide better support for data exploration beyond that of merely printing a list of rules sorted by their J-measures. A robust algorithm for quantizing continuous variables has also been developed so that this task no longer has to be done manually by an experienced user.

In addition, the design, experimental justification, and experimental demonstration of a completely new, user friendly information gathering system called Poirot has been presented. This autonomous software agent can assist World Wide Web users in staying up to date on new developments of importance by first learning what is of interest to the user and then independently searching for relevant new web pages and significant changes to previously discovered ones. The performance of Poirot's learning algorithm on the Reuters-21578 and WebKB data sets demonstrates that this autonomous system can provide substantial improvements over manually surfing the Web or performing a keyword search via a Web search engine.

References

Alspector, J, A. Kolcz, and N. Karunanithi. (1997). Feature-based and Clique-based User Models for Movie Selection: A Comparative Study. *User Modeling and User-Adapted Interaction* 7(4):279-304.

Alspector, J. (2001). Personal communication.

AltaVista.

http://www.altavista.com/

Amato, G. (2000). Personal communication.

Amato, G., C. Thanos, and U. Straccia. (2000). EUROgatherer: A Personalized Gathering and Delivery Service on the Web. *In* "Proc. of the 4th World Multiconference on Systemics, Cybernetics and Informatics (SCI-2000)."

http://pc-erato2.iei.pi.cnr.it/eurogatherer/

Armstrong, R., D. Freitag, T. Joachims, and T. Mitchell. (1995). WebWatcher: A Learning Apprentice for the World Wide Web. *In* "Proceedings of the 1995 AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments."

http://www-ai.cs.uni-dortmund.de/PERSONAL/joachims.html

Backflip.

http://www.backflip.com/

Balabanovic, M., Y. Shoham, and Y. Yun. (1995). An Adaptive Agent for Automated Web Browsing. Technical Report CS-TN-97-52. Palo Alto, CA: Stanford University.

Bay, S. D. and M. J. Pazzani. (1999). Detecting Change in Categorical Data: Mining Contrast Sets. *In* "Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining."
http://www.ics.uci.edu/~sbay/

Billsus, D. and M. J. Pazzani. (1998). Learning Collaborative Information Filters. *In* "Proceedings of the International Conference on Machine Learning." Madison, WI: Morgan Kaufmann.
http://www.ics.uci.edu/~pazzani/Publications/

Boley, D., M. Gini, R. Gross, E. Han, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. (1999). Document Categorization and Query Generation on the World Wide Web Using WebACE. *Journal of Artificial Intelligence Review* **13**(5-6):365-391.
http://maya.cs.depaul.edu/~mobasher/

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone. (1984). Classification and Regression Trees. New York, NY: Chapman & Hall.

Budzik, J. and K. J. Hammond. (1999). Watson: Anticipating and Contextualizing Information Needs. *In* "Proceedings of the Sixty-second Annual Meeting of the American Society for Information Science." Medford, NJ: Information Today, Inc.
http://dent.infolab.nwu.edu/infolab/projects/projectmain.asp

Buntine, W. and R. Caruana (1992). Introduction to IND Version 2.1 and Recursive Partitioning. NASA Ames Research Center, Mail Stop 269-2, Moffet Field, CA 94035.

Cohen, W. W. (1995). Fast Effective Rule Induction. *In* "Machine Learning Conference Proceedings," 115-123.

Corana, A., M. Marchesi, C. Martini, and S. Ridella. (1987). Minimizing Multimodal Functions of Continuous Variables with the "Simulated Annealing" Algorithm. *ACM Transactions on Mathematical Software* **13**(3):262-280.

Cover, T. M. and J. A. Thomas. (1991). Elements of Information Theory. New York, NY: Wiley.

Craven, M., D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. (1998). Learning to Extract Symbolic Knowledge from the World Wide Web. *In* "AAAI-98."

`http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/`

`http://www.cs.cmu.edu/~textlearning/`

Dietterich, T. G. (1998). Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Computation,* **10**(7):1895-1924.

`http://www.cs.orst.edu/~tgd/`

Domingos, P. and M. Pazzani. (1996). Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier. *In* "Proceedings of the Thirteenth International Conference on Machine Learning," 105-112. San Francisco, CA: Morgan Kaufmann.

`http://www.ics.uci.edu/~pazzani/Publications/`

Dougherty, J., R. Kohavi, and M. Sahami. (1995). Supervised and Unsupervised Discretization of Continuous Features. *In* "Machine Learning Conference Proceedings," 194-202.

`http://robotics.stanford.edu/users/ronnyk/`

Duda, R. O. and P. E. Hart. (1973). Pattern Classification and Scene Analysis. New York, NY: John Wiley & Sons.

Ezawa, K. J. and T. Schuermann. (1995). Fraud/Uncollectable Debt Detection Using a Bayesian Network Learning System. *In* "UAI-95," 157-166.

Frakes, W. B. and R. Baeza-Yates, eds. (1992). Information Retrieval, Data Structures and Algorithms. Englewood Cliffs, NJ: Prentice Hall.

Gauch, S. and J. B. Smith. (1991). Search Improvement via Automatic Query Reformulation. *ACM Transactions on Information Systems* **9**(3):249-280.

Goodman, R. M., P. Smyth, C. M. Higgins, and J. W. Miller. (1992). Rule-Based Neural Networks for Classification and Probability Estimation. *Neural Computation* **4**(6):781-804.

Haykin, S. (1999). Neural Networks: A Comprehensive Foundation, 2nd edition. Upper Saddle River, NJ: Prentice Hall.

Herlocker, J. L., J. A. Konstan, A. Borchers, and J. Riedl. (1999). An Algorithmic Framework for Performing Collaborative Filtering. *In* "Proceedings of the 1999 Conference on Research and Development in Information Retrieval."

`http://www.cs.umn.edu/Research/GroupLens/`

Jacobs, P. S. and L. F. Rau. (1990). SCISOR: Extracting Information from On-line News. *Communications of the ACM* **33**(11):88-97.

Joachims, T. (1996). A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. Technical Report CMU-CS-96-118. Pittsburgh, PA: Carnegie Mellon University, School of Computer Science.

Joachims, T. (1998). Text Categorization with Support Vector Machines: Learning with Many Relevant Features. LS-8 Report 23. Dortmund, Germany: University of Dortmund, Computer Science Department.

Joachims, T. (1999). Making Large-scale SVM Learning Practical. *In* "Advances in Kernel Methods - Support Vector Learning." MIT-Press.

http://ais.gmd.de/~thorsten/svm_light/

John, G. H., R. Kohavi, and K. Pfleger. (1994). Irrelevant Features and the Subset Selection Problem. *In* "Machine Learning: Proceedings of the Eleventh International Conference," 121-129. San Francisco, CA: Morgan Kaufmann.

http://robotics.stanford.edu/users/ronnyk/ronnyk-bib.html

JX Application Framework.

http://www.newplanetsoftware.com/jx/

Kohavi, R., B. Becker, and D. Sommerfield. (1997). Improving Simple Bayes. Poster at ECML-97.

http://robotics.stanford.edu/users/ronnyk/ronnyk-bib.html

Koller, D. and M. Sahami. (1996). Toward Optimal Feature Selection. *In* "Machine Learning: Proceedings of the Thirteenth International Conference," San Francisco, CA: Morgan Kaufmann.

Krovetz, R. and W. B. Croft. (1992). Lexical Ambiguity and Information Retrieval. *ACM Transactions on Information Systems* **10**(2):115-141.

Krulwich, B. and C. Burkey. (1997). The InfoFinder Agent: Learning User Interest through Heuristic Phrase Extraction. *IEEE Intelligent Systems Journal (Expert)* **12**(5):22-27.

http://www.geocities.com/ResearchTriangle/9430/

Lawrence, S. and C. L. Giles. (1998). Searching the World Wide Web. *Science* **280**:98-100.

Lewis, D. D., W. B. Croft, and N. Bhandaru. (1989). Language-Oriented Information Retrieval. *International Journal of Intelligent Systems* **4**:285-318.

http://www.research.att.com/~lewis/

Lieberman, H. (1995). Letizia: An Agent That Assists Web Browsing. *In* "Proceedings of the International Joint Conference on Artificial Intelligence."

http://lieber.www.media.mit.edu/people/lieber/Lieberary/Letizia/

Lieberman, H. (1997). Autonomous Interface Agents. *In* "Proceedings of the ACM Conference on Computers and Human Interface, CHI-97."

http://lieber.www.media.mit.edu/people/lieber/Lieberary/Letizia/

McCallum, A. and K. Nigam. (1998). A Comparison of Event Models for Naive Bayes Text Classification. AAAI-98 Workshop on "Learning for Text Categorization."

http://www.cs.cmu.edu/~mccallum/

Mladenic, D. (1996). Personal WebWatcher: Implementation and Design. Technical Report IJS-DP-7472. Ljubljana, Slovenia: J. Stefan Institute, Department for Intelligent Systems.

http://www-ai.ijs.si/DunjaMladenic/home.html

Morita, M. and Y. Shinoda. (1994). Information filtering based on user behavior analysis and best match text retrieval. *In* "Proceedings of the Seventeenth Annual International ACM SIGIR Conferencon Research and Development in Information Retrieval," 272-281.

Nelder, J.A., and R. Mead. (1965). A Simplex Method for Function Minimization. *Comput. J.* **7**:308-313.

Nichols, D. M. (1998). Implicit Rating and Filtering. *In* "Proceedings of the Fifth DELOS Workshop on Filtering and Collaborative Filtering," 31-36. Budapest, Hungary.

http://tina.lancs.ac.uk/computing/users/dmn/

Oard, D. W. and Marchionini, G. (1996). A Conceptual Framework for Text Filtering. Technical Report CAR-TR-830. College Park, MD: University of Maryland, Human Computer Interaction Laboratory.

Pazzani, M. and D. Billsus. (1997). Learning and Revising User Profiles: The Identification of Interesting Web Sites. *Machine Learning* **27**:313-331.

http://www.ics.uci.edu/~pazzani/Publications/

Quinlan, J. R. (1993). C4.5: Programs for Empirical Learning. San Francisco, CA: Morgan Kaufmann.

Raghavan, V. V., G. S. Jung, and P. Bollmann. (1989). A Critical Investigation of Recall and Precision as Measures of Retrieval System Performance. *ACM Transactions on Information Systems* **7**(3):205-229.

Rocchio, J. (1971). Relevance Feedback in Information Retrieval. *In* "The SMART Retrieval System: Experiments in Automatic Document Processing," 313-323. Prentice-Hall.

Reuters-21578, Distribution 1.0. (1987).

http://www.research.att.com/~lewis/

Salton, G. (1991). Developments in Automatic Text Retrieval. *Science* **253**:974-979.

Schapire, R. E., Y. Singer, and A. Singhal. (1998). Boosting and Rocchio Applied to Text Filtering. *In* "SIGIR '98: Proceedings of the 21st Annual International Conference on Research and Development in Information Retrieval."

`http://www.research.att.com/~schapire/`

Segal, R. and O. Etzioni. (1994). Learning Decision Lists Using Homogeneous Rules. *In* "Proceedings of the Twelfth National Conference on Artificial Intelligence."

`http://www.cs.washington.edu/homes/etzioni/`

Selberg, E. and O. Etzioni. (1995). Multi-Service Search and Comparison Using the MetaCrawler. *In* "Proceedings of the 4th World Wide Web Conference," 195-208.

`http://www.cs.washington.edu/homes/etzioni/`

Setiono, R. (1997). Extracting Rules from Neural Networks by Pruning and Hidden-Unit Splitting. *Neural Computation* **9**:205-225.

Shardanand, U. and P. Maes. (1995). Social Information Filtering: Algorithms for Automating "Word of Mouth." *In* "CHI95: Proceedings of the Conference on Human Factors in Computing Systems," 210-217. Denver, CO: ACM Press.

Smyth, P. (1988). Ph.D. thesis, California Institute of Technology.

Smyth, P. and R. M. Goodman. (1992). An Information Theoretic Approach to Rule Induction from Databases. *IEEE Transactions on Knowledge and Data Engineering* **4**(4):301-316.

Smyth, P. (1996). Personal communication.

Spangler, R. R. (1999). Ph.D. thesis, California Institute of Technology.

Spertus, E. (1997). ParaSite: Mining Structural Information on the Web. *Computer Networks and ISDN Systems* **29**:1205-1215.

Starr, B., M. S. Ackerman, and M. Pazzani. (1996). Do-I-Care: A Collaborative Web Agent. *In* "Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'96)," 273-274.

http://www.ics.uci.edu/~pazzani/Publications/

Towell, G. G. and J. W. Shavlik. (1993). The Extraction of Refined Rules from Knowledge-Based Neural Networks. *Machine Learning* **13**:71-101.

http://www.cs.wisc.edu/~shavlik/

UCI machine learning data set repository.

http://www.ics.uci.edu/~mlearn/MLRepository.html

URL-minder.

http://www.urlminder.com/

Vapnik, V. (1995). The Nature of Statistical Learning Theory. New York, NY: Springer.

Voigt, K. (1995). Reasoning about Changes and Uncertainty in Browser Customization. *In* "AAAI-95 Fall Symposium Series, AI Applications in Knowledge Navigation and Retrieval." Cambridge, MA: MIT.

http://www.csci.csusb.edu/voigt/

Weiner, E., J. O. Pedersen, and A. S. Weigend. (1995). A Neural Network Approach to Topic Spotting. *In* "Proceedings of the Fourth Annual Symposium on Document Analysis and Information Retrieval (SDAIR'95)," 317-332.

http://www.stern.nyu.edu/~aweigend/

Weiss, S. M. and N. Indurkhya. (1995). Rule-based Machine Learning Methods for Functional Prediction. *Journal of Artificial Intelligence Research* **3**:383-403.

Yang, Y. and X. Liu. (1999). A Re-examination of Text Categorization Methods. *In* "Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval," 42-49.
http://www.cs.cmu.edu/~yiming/