Text Categorization Through Probabilistic Learning: Applications to Recommender Systems

Paul N. Bennett

CS 379H
Department of Computer Sciences
Plan II Honors Program
University of Texas
Austin, TX 78712
Email: pbennett@cs.utexas.edu

May 4, 1998

________________________________________

Raymond J. Mooney, Ph.D.
Department Of Computer Sciences
Supervising Professor

________________________________________

Risto Miikkulainen, Ph.D.
Department Of Computer Sciences
Second Reader
Abstract

Author: Paul N. Bennett
Title: Text Categorization Through Probabilistic Learning: Applications to Recommender Systems
Supervising Professor: Raymond J. Mooney, Ph.D.

With the growth of the World Wide Web, recommender systems have received an increasing amount of attention. Many recommender systems in use today are based on collaborative filtering. This project has focused on LIBRA, a content-based book recommending system. By utilizing text categorization methods and the information available for each book, the system determines a user profile which is used as the basis of recommendations made to the user. Instead of the bag-of-words approach used in many other statistical text categorization approaches, LIBRA parses each text sample into a semi-structured representation. We have used standard Machine Learning techniques to analyze the performance of several algorithms on this learning task. In addition, we analyze the utility of several methods of feature construction and selection (i.e. methods of choosing the representation of an item that the learning algorithm actually uses). After analyzing the system we conclude that good recommendations are produced after a relatively small number of training examples. We also conclude that the feature selection method tested does not improve the performance of these algorithms in any systematic way, though the results indicate other feature selection methods may prove useful. Feature construction, however, while not providing a large increase in performance with the particular construction methods used here, holds promise of providing performance improvements for the algorithms investigated. This text assumes only minor familiarity with concepts of artificial intelligence and should be readable by the upper division computer science undergraduate familiar with basic concepts of probability theory and set theory.
Contents

1 Introduction

2 Background
   2.1 Machine Learning ................................................. 2
   2.2 Recommender Systems ............................................. 3
   2.3 Text Categorization .............................................. 4
   2.4 Information Extraction ......................................... 4
   2.5 Naive Bayes Algorithm .......................................... 4
   2.6 Feature Selection and Feature Construction ................. 5
   2.7 Information Gain ................................................. 6

3 System Description .................................................. 7
   3.1 Extracting Information and Building a Database .............. 7
   3.2 Book Representation ........................................... 7
   3.3 Learning a Profile ............................................. 8
   3.4 Producing, Explaining and Revising Recommendations ......... 8
   3.5 System Details .................................................. 9
      3.5.1 Estimating Probabilities .................................... 9
      3.5.2 Real Valued Scores .................................... 11
      3.5.3 Feature Selection ...................................... 12

4 Experimental Results ................................................ 13
   4.1 Methodology ..................................................... 13
      4.1.1 Data Collection ..................................... 13
      4.1.2 Performance Measures ................................... 14
      4.1.3 Systems and Hypotheses ................................ 15
   4.2 Results .......................................................... 16
   4.3 Discussion ..................................................... 20

5 Future Work .................................................................. 23

6 Conclusions .................................................................. 23

7 Acknowledgements ..................................................... 24

Appendix A Sample Amazon Page ......................................... 25

Appendix B LIBRA Extraction from Sample Amazon Page .......... 26

References ................................................................. 27
1 Introduction

The growth of the World Wide Web has brought to nearly everyone’s attention a trend that has been steadily increasing in recent years; namely, a jump in the amount of digital information available, but a lack of effective access to this information, has prompted an increase in development of recommender systems. A recommender system is a system that suggests items (e.g. records, books, news articles, pictures, etc.) of interest to the user (Maes, 1994; Resnik & Varian, 1997). The majority of systems in existence are based on collaborative filtering. These collaborative filtering systems make recommendations by matching users with other “like-minded” users where “like-minded” is indicated by correlations among user ratings of items. This approach tends to break down when (a) the system does not know of any similar users for a given user, (b) “marginal” items fail to be rated by enough users, and (c) “new” items cannot be recommended until others have rated them.

In contrast, a content-based recommender builds a profile of a user based on the content of the items and the user’s ratings (Balabanovic & Shoham, 1997). This allows the system to make recommendations to a user based solely on that user’s interests. Our prototype system applies text categorization learning methods to items with semi-structured text descriptions in order to make recommendations to the user. Other content-based recommenders that use text categorization have been used to recommend web pages (Pazzani, Muramatsu, & Billsus, 1996) and Usenet messages (Lang, 1995). The system developed for this research is dubbed LIBRA (Learning Intelligent Book Recommending Agent) and was conceived as an interdisciplinary project between Ray Mooney of the UT-Austin Department of Computer Sciences and Loriene Roy of UT-Austin Library and Information Sciences. Currently LIBRA operates by building a database of books retrieved from web pages at Amazon.com (the current recommender at Amazon.com appears to be based on collaborative filtering). These web pages contain semi-structured text descriptions of books. The system, however, can be applied to any semi-structured samples of text. A user provides an integer-rating in the range of 1 - 10. The system uses these ratings to generate a user profile.

Most other content-based systems that use text categorization do not exploit semi-structured text that may appear in the data. It is common for many text-categorization approaches to use a bag-of-words method which represents a text sample as an unordered set of all words appearing in the text (regardless of position). LIBRA, however, uses a simple pattern-based extraction method to identify values for various slots, e.g. Author, ISBN, Price, etc. The book is represented as a group of set-valued features (Cohen, 1996a, 1996b) where each slot is a feature. The value of a slot is the set of words that occur in the value portion of the pattern for the slot. Currently not all of the slots extracted are used for learning. By taking advantage of a semi-structured representation, the system is allowed the latitude to favor more informative types of information (e.g. An author’s name occurring in one slot, e.g. “AUTHOR: J.R.R. Tolkien”, could theoretically carry more weight than when it appears in a less meaningful slot, “REVIEW: Best books since J.R.R. Tolkien’s ...”).

Since most recommender systems output a ranked list of samples, it is more important to assess the effectiveness of the ranking produced rather than a simple category prediction (such as “like” or “dislike”) or even strict scores. The performance metric used, Spearman’s ranked correlation coefficient, actually evaluates the quality of the ranking. It accomplishes this by correlating the
ranking of items by user scores to the ranking of items by system predictions.

The prototype system was developed previous to this research—that is, the software tools had already been constructed for: downloading and extracting text from web pages; learning with the Naive Bayes algorithm extended for set-valued features; and learning with the Weighted Binary Naive Bayes algorithm. The current research involved: constructing data sets appropriate for performance evaluation; integrating the system into an external testing system; development of the 10-Ratings learning algorithm (described below); addition of alternate feature construction methods (described below); development of the distinction between rating accuracy and ranking accuracy; investigation and software construction of a testing metric appropriate for evaluating ranking accuracy; and systematic analysis of the learning algorithms’ performance, feature construction methods, and feature selection efficacy.

The remainder of the paper is organized as follows: Section 2 provides general background information needed to understand the methods applied in the research and the results reported here; Section 3 gives a complete description of the system; Section 4 details the experimental setting, the results obtained, and a discussion of the results; Section 5 discusses future work that could extend the research, and section 6 summarizes the paper.

2 Background

2.1 Machine Learning

In general, learning is the use of experience in such a way that performance at a certain task is greater with that experience than without it (Mitchell, 1997). Machine Learning seeks to construct systems that have the ability to automatically generalize from their experience in a manner that increases performance at a given task. However, in order to scientifically analyze a system’s performance, we seek to capture certain notions such as what it means for a system’s performance to increase, when it can be said that one system experimentally outperforms another, and how a difference in performance is judged to be significant. To this end, the field of Machine Learning has standard tools to use when assessing a system.

Usually a learning system’s experience comes in the form of training examples. Sometimes these examples carry useful tags of information provided by a “teacher” (either the user or another software system) such as a category associated with the example, an underlying concept with which it corresponds, a score, or other information which the learner can possibly make use of during the process of learning (also called training). In other cases, the examples come without any additional information provided by an outside source. The former case is called supervised learning and the latter unsupervised (Russell & Norvig, 1995).

In order to judge the performance of the system, a learning curve is plotted with the number of training examples on the x-axis and some performance metric on the y-axis (where an increase in performance is shown by an increase in the y value). Several features of this curve are indicative of the system’s properties. The first of these is the rate at which the curve climbs to a high value. A very steep curve means that the system requires only a small number of values before its ability to generalize from these examples leads to significant performance gains. This is particularly
important when obtaining examples is expensive (whether the resource expense is computational, monetary, or user time), as it means the system will “cost” little to perform well. Another major feature of a learning curve is the asymptotic height of the curve on the y-axis; typically this point comes at the maximum number of training examples. This part of the curve is an indicator of the best performance generally obtainable from a system.

In order to compensate for the random possibility that a system performed well because of the particular kind or order of the training examples, the system is typically run $n$ times and the average of the performance criteria over the $n$ times is used in constructing the learning curves. To further ensure that this is not a skewed sampling, the choice of the training examples to use in these $n$ trial runs are usually chosen by randomly partitioning the $x$ number of examples into $n$ subsets with $\frac{x}{n}$ examples in each partition; each subset is used as the “test” examples (the examples that will be used to judge system performance) for exactly one of the trials and the remaining $x - \frac{x}{n}$ are used as the training examples during that trial. Thus each example is used as a test example once and a training example $n - 1$ times. This process is termed $n$-fold cross-validation (Mitchell, 1997)—the standard choice of $n$ in Machine Learning is 10.

As mentioned above, the average of all the trial runs on some metric(s) is used as an indicator of the systems performance. In order to compare two such averages, a standard statistical tool, Student’s $t$, is used (Spatz & Johnston, 1984; Mitchell, 1997). Student’s $t$ (t-test) allows one to judge whether the difference in two means is significant at a certain confidence level, $p$. A smaller $p$ corresponds to an increased confidence that the difference is significant; typically confidence scores of $p > 0.05$ (also referred to as below 95% confidence) are considered insignificant. When choosing the metric that will later be compared with the t-test, it is important that the metric measure an aspect of the system relevant to the task performed.

### 2.2 Recommender Systems

A recommender system uses items rated by the user to suggest new items that the user will like. While recommender systems have been used in AI for sometime, their widespread investigation and use has come only recently with the explosion of electronic data which has inundated nearly every computer user. Users seldom have enough time to review all of the information that is available to them, therefore a recommender system can prove extremely useful in prioritizing and filtering the information at which users will look.

In order for a system to be usable as a recommender system, however, it must not only perform well, but it should possess several other desirable properties. These are: (1) A user should be allowed to change a rating—either to a different value or completely withdrawing the rating. (2) The user should be able to extend the current set of rated examples with new rated examples. (3) A user’s request for recommendations from the system would be processed within a reasonable (specific to the task) amount of time. (4) The benefits the system provides should outweigh the cost of training the system. (5) The user’s role in training should be integrated seamlessly into the actions they would normally be performing to accomplish the task. Preferably, all of these functions would be performed in real-time; that is, the ideal system would allow the user to interact effortlessly with it.
There are two general categories of recommender systems, content-based and collaborative filtering systems. A content-based system uses a representation of the item and the profile of the user it has constructed to analyze whether it is likely that the user will like the item based on its content. This is similar to a friend that might recommend an item to you because he knows your preferences and has tried the item himself. A collaborative filtering system uses the ratings of other "similar" users to make a recommendation. These systems often judge similarity by overlap in ratings among other items. This approach is like the friend who will recommend an item to you because other friends, believed to be similar to you, liked the item. Some recommender systems use a hybridization of these two approaches (Balabanovic & Shoham, 1997).

2.3 Text Categorization

Text categorization in general involves assigning a category (or categories) to a sample of text. Sometimes the task is to determine the part-of-speech that a word has in some text; in other cases it may involve classifying a sample into relevant subject categories. Typically the text is represented using some set of features extracted from the sample. This ranges from using each single word as a feature to using logical sentences built to correspond to the meaning of a segment of text.

Given a certain text sample, we would like to be able to determine the most probable category for that sample given all of the information; this category is referred to as the maximum a posteriori category (Cat_{MAP}) (Mitchell, 1997). A classifier system that, for each sample, can determine the Cat_{MAP} and chooses the Cat_{MAP} as the predicted class, cannot on average be outperformed by another system which uses the same information and hypothesis space. A classifier that performs this way is called a Bayes Optimal Classifier (Mitchell, 1997).

2.4 Information Extraction

Information Extraction attempts to apply patterns (or templates) to text in order to extract information relevant to certain areas (Lehnert & Sundheim, 1991; Cardie, 1997; Calif & Mooney, 1998). In essence, it attempts to make advantage of certain shallow regularities in language as a means to extract information relevant to certain highly informative fields. For instance, for text describing a conference, we may want to automatically extract features for fields such as date, place, price, etc. Information Extraction can be helpful in breaking down free form text into meaningful segments. For the interested reader, appendices A and B show a sample Amazon page and the information extracted from it by LIBRA, respectively.

2.5 Naive Bayes Algorithm

The Naive Bayes algorithm is a simple algorithm but has performed quite well in most domains (Mitchell, 1997). It is based on Bayes' "inversion" theorem which can be stated as:

Let Hypothesis and Evidence be any random variables,

\[
P(Hypothesis | Evidence) = \frac{P(Evidence | Hypothesis)P(Hypothesis)}{P(Evidence)}
\]  

(1)
Assuming that we can represent an example, \( \varepsilon \), as a conjunction of \( n \) features \( f_i \in F \) with no loss of information, then in order to construct a Bayes Optimal Classifier that chooses the \( \text{Cat}_{\text{MAP}} \) of a classification space, \( \text{Category} \), of \( j \) categories given an example, we have (where \( \text{argmax} \) returns the index that maximizes the value of its argument):

\[
\text{Cat}_{\text{MAP}} = \arg\max_{\text{Cat}_k \in \text{Category}} P(\text{Cat}_k | \varepsilon)
\]

\[
= \arg\max_{\text{Cat}_k \in \text{Category}} P(\text{Cat}_k | f_1, f_2, \ldots, f_n)
\]

From Bayes' Theorem we have,

\[
= \arg\max_{\text{Cat}_k \in \text{Category}} \frac{P(f_1, f_2, \ldots, f_n | \text{Cat}_k) P(\text{Cat}_k)}{P(f_1, f_2, \ldots, f_n)}
\]

Since the denominator is independent of the \( \text{argmax} \) index \( \text{Cat}_k \),

i.e. it is simply a normalization term

\[
= \arg\max_{\text{Cat}_k \in \text{Category}} P(f_1, f_2, \ldots, f_n | \text{Cat}_k) P(\text{Cat}_k)
\]

(2)

The Naive Bayes algorithm makes an approximation to this Bayes Optimal Classifier by assuming that each of the \( f_i \in F \) are independent of any subset of \( F - \{f_i\} \) given the category value. This allows for the following simplification:

\[
\text{Cat}_{\text{MAP}} = \arg\max_{\text{Cat}_k \in \text{Category}} P(f_1, f_2, \ldots, f_n | \text{Cat}_k) P(\text{Cat}_k)
\]

From the Naive Bayes independence assumption

\[
= \arg\max_{\text{Cat}_k \in \text{Category}} \prod_{f_i} [P(f_i | \text{Cat}_k) P(\text{Cat}_k)]
\]

(3)

That the Naive Bayes algorithm is a Bayes Optimal Classifier when the independence assumption holds, is obvious, but what is less obvious is that the Naive Bayes algorithm sometimes performs optimally even when this assumption doesn’t hold. This results from the fact that under many less restrictive conditions, even though the probability estimates are strictly incorrect, the actual \( \text{Cat}_{\text{MAP}} \) is still the category with the maximal probability estimate (Domingos & Pazzani, 1996).

### 2.6 Feature Selection and Feature Construction

It is important to note that we have used the term *features* to refer both to the slots (set-valued features) and when referring to the words that were extracted as the values of the slots. Unless it is explicitly made clear, when we use *word*, *feature*, *term*, etc., we are referring to an attribute that is an atom composed of Slot-name.token, e.g. *AUTHOR.Lewis, REVIEWS.Lewis*, etc. It is atomic in the sense that, as far as the learning algorithm is concerned there is no connection between *AUTHOR.Lewis* and *REVIEWS.Lewis*. Also, we would like to make it clear that a feature is in no way limited to being simply an actual English word or proper name, etc. concatenated to the slot name. There are various methods to construct features, and a system can apply these to the text occurring in a slot. For example, if a system is using multiple word phrases, then we
might have attributes such as TITLE/(Big Boy/), REVIEWS/(Big Boy/), TITLE/(Boy Big/), etc. All of which are distinct attributes. Even using the single word method, the construction of the attributes depends on how the system represents a “single word”. For instance, we might treat an author’s entire name as a “single word” which might yield, AUTHOR, “C.S.Lewis”.

There are several problems with our feature representation of examples. The main problem is that the number of features is quite large. There are about 30,000 in each of the two data sets, and this is only with using each word slot pair. If one were to use multiple word phrases or other construction, the number of features becomes much higher. As learning algorithms tend to have a computational complexity proportional to the size of the feature set (Koller & Sahami, 1996), large feature sets can be quite expensive. While Naive Bayes is one of the cheaper algorithms in this respect, in order to apply other more sophisticated algorithms to this task, it would most likely require feature selection in order to be computationally feasible. In addition, the probability distribution function for the probability of a category given an example is often extremely complex in problems with a large feature set (Koller & Sahami, 1996). When data is limited, it is very difficult to accurately estimate the numerous probabilistic parameters needed for this high dimensional space; thus overfitting, estimating parameters that are overly specific to the training set and thus don’t generalize well to the test set, is likely (Koller & Sahami, 1996). In addition, the large number of irrelevant and redundant features that tend to be present in large feature spaces often end up misleading the learning system (Koller & Sahami, 1996).

Thus, the general goals of feature selection are more accurate results and reduced running time. Feature selection generally tries to achieve its goals by: (1) reducing the feature set to a smaller but highly informative subset; (2) and constructing higher order features whose higher relevance (will hopefully) more than compensates for adding an additional feature (Yang & Pedersen, 1997). The first we will always refer to as feature selection. The second of these we will refer to as feature construction. One method of constructing higher order features we have tried uses both multiple word phrases (pairs of adjacent words) and single words. Consider that using this method more than doubles the number of features in the domain. It more than doubles because of duplicates being identical as single words. That is, “Black Cat Black Dog” would be represented as [Black Cat Dog] in a single word approach, but as [Black (Black Cat) Cat (Cat Black) (Black Dog) Dog] in a multiple word phrase approach. Thus adding higher order features to the initial feature pool sometimes increases the need for using feature selection to select only a subset to actually use—since the much higher number of features may worsen those problems mentioned above. We experiment with methods of both feature selection and feature construction.

2.7 Information Gain

If we are to perform feature selection over the data sets, intuitively we would like to choose the optimal subset $F_G$ of $F$ such that $F_G$ maintains the original relevant information available to us in $F$. One way to approximate this optimal subset is to choose the $n$ most “informative” features of $F$ according to some criterion for judging how informative a feature is. One standard way to determine the amount of information a feature has is by applying standard ideas of information theory, Entropy and Information Gain. Entropy is a measure of the homogeneity of a data sample. That is, given a data set contains items from $j$ categories, Entropy measures the extent to which
these items are dispersed among the \( j \) categories. Information Gain measures the reduction in
Entropy when given the value of a certain feature. Thus, features that yield a large reduction
in Entropy are often helpful in classification because their ability to divide the training set into
"purer" subsets than the other features, is a good indicator they will continue to do so over the
whole domain. The standard definitions for Entropy and Information Gain are (Mitchell, 1997):
Let \( E \) be the set of all training examples, \( \{c_i\}_{i=1}^j \) be the set of \( j \) categories in the target space, \( \tau \) be
the set of all attributes, \( T \) be an attribute \( T \in \tau \), \( Values(T) \) be the set of values which \( T \) can take
on, and \( E_v(T) \) denote \( \{x \in E \mid \text{the value of } T \text{ in } x \text{ is } v, \text{where } v \in Values(T)\} \). Since it will be
clear from context which \( T \in \tau \) we mean we will abbreviate \( E_v(T) \) as \( E_v \). Then Entropy (Entropy)
and Information Gain (Gain) can be defined as:

\[
\text{Entropy}(E) \equiv \sum_{i=1}^{j} [-P(c_i \mid E) \log_2 P(c_i \mid E)]
\]

(4)

\[
\text{Gain}(E, T) \equiv \text{Entropy}(E) - \sum_{v \in Values(T)} \left[ \frac{|E_v|}{|E|} \text{Entropy}(E_v) \right]
\]

(5)

It can easily be seen from the definition how Information Gain measures the reduction in Entropy
induced by partitioning a set according to the values of a given attribute.

3 System Description

3.1 Extracting Information and Building a Database

LIBRA currently has accumulated a database of 2,600 science fiction books and a database of 3,061
general literary fiction books. These databases were built by first performing a keyword/subject
search and then downloading and parsing the pages corresponding to the resulting URL’s from
the search. Only the title, authors, synopsis, and subject slots are currently employed in learning;
however, values for URL, type, length, price, ISBN, etc. are also extracted.

The values for the slots are extracted with a pattern-based matcher that uses handwritten rules,
including pre-filler, filler, and post-filler patterns (Califf & Mooney, 1998), to extract information
from the text. The Amazon pages are fairly structured making it easy to design information
extraction rules. The values extracted for each slot are stored as an unordered set of words. The
collection of these set-valued features make up the complete representation for a book.

3.2 Book Representation

Each slot is treated as a vector of binary features; thus this differs from approaches which consider
only the words that occur in a given text sample (Mitchell, 1997; Joachims, 1997). Furthermore,
we have applied methods of feature construction and feature selection.

For feature construction, we have tried one type of higher order feature—the multiword method.
This makes a feature out of every word that occurs in a slot as in the normal method, and in
addition, makes a feature out of every pair of sequential words. This acts as a higher order feature since it is essentially weighting the fact that order and co-occurrence matters.

While Naive Bayes can work efficiently with large dimensional spaces, it may be of help to perform feature selection over the features prior to learning. One reason why this may be helpful is that the sheer space required to store all of the probabilistic estimates can get expensive. We have used the information gain criterion to choose the $N$ most informative features to be used during training and prediction. $N$ was varied to be 500, 1000, 2000, 4000, 8000, and 16000. However, it must be noted here that because Naive Bayes makes explicit independence assumptions about the terms, it is thought to be less sensitive to changes in context caused by feature selection. Thus, it may make a poor indicator as to the superiority of a given feature selection method (Yang & Pedersen, 1997).

### 3.3 Learning a Profile

A Naive Bayes (NB) Classifier which has been extended to efficiently deal with set-valued features is used for the text categorization task. The probability estimates are smoothed using Laplace estimates as described in Kohavi, Becker, and Sommerfeld (1997). The smoothing includes near-zero estimates for novel words encountered in test samples but not encountered in training examples. The 1 - 10 user rating for an item is treated as the category of that item. So, in order to calculate the posterior probabilities of the categories, the probabilities of a feature given a category (rating) are computed. The probabilities are mapped into a logarithmic space to avoid underflow. In order to be able to efficiently compute a probability estimate for a sample at testing time, the posterior probability of each category given the empty set is precomputed (i.e. the probability of a category given no words occur in the example). These estimates can then be adjusted for the actual words that occur in a sample.

In fact, revising the posterior probabilities estimates is quite easy with the Naive Bayes approach. Naive Bayes (and Bayesian methods in general) estimates can be incrementally updated—both adding and retracting information (Pearl, 1988). Thus, Naive Bayes allows us to achieve several of the criteria for a useful recommender system enumerated in section 2.2. Namely, the user can retract ratings, extend the set of examples rated, and obtain recommendations, all fairly efficiently. Currently the system is not constructed in a way to fully exploit all of the incremental attributes of the algorithm.

### 3.4 Producing, Explaining and Revising Recommendations

In order to produce recommendations, LIBRA learns a profile, predicts scores for the non-rated samples, and finally ranks the samples by their scores. Currently LIBRA uses one of three methods to learn a user profile. The first method is simply a binary NB classifier. It treats items rated 1 - 5 as negative instances, and those rated 6 - 10 as positive instances. The scores are ranked based on the natural log of the posterior odds of positive, \( \ln \left( \frac{P(\text{Positive} | \text{Example})}{P(\text{Negative} | \text{Example})} \right) \) (Pearl, 1988). A second method treats the 10 ratings as 10 distinct categories. When predicting for a test sample, the system first computes the posterior probability of each category given the test sample. Then the expected value of the posterior probability distribution for the categories is computed and used as
the predicted score, $\sum_{i=1}^{10} iP(i)$, where $P(i)$ is the posterior probability for category $i$. We use the expected value rather than simply choosing the most probable category in order to better represent the continuity of scores. Consider the case where $P(3) = 0.35$, $P(9) = 0.32$, and $P(10) = 0.33$; even though 3 is the most probable category, the “closeness” of the other categories makes it more likely that the example would fall toward the high end. Using the expected value of 7.23 addresses this issue. When using this 10-category model to predict a binary category (positive: rating $> 5$; negative: rating $\leq 5$), we classify an example as positive if and only if $\sum_{i=6}^{10} P(i) > \sum_{i=1}^{5} P(i)$. The final method used is a weighted binary model that maps the user’s 1-10 rating $r$ into a weight, $w_r$, in the closed interval $[0,1]$, where $w_r = \frac{r-1}{max-min}$. The general formula for this is $w_r = \frac{r-min}{max-min}$, where $0 \leq min \leq r \leq max$ and $max \neq min$. Then, if a word occurs in $n$ training examples given a rating of $r$, it is counted as occurring $nw_r$ times in positive examples and $n(1-w_r)$ in negative examples. The ranked predictions are once again produced by ordering based on posterior odds of positive.

Both the Binary and the Weighted Binary approach have a limited explanatory capability. The explanations consist of the top features that most contributed to the score, e.g.

The Gods Themselves by Issac Asimov classified as POSITIVE because:
- words:award(4.20), words:earth(4.20), words:terrify(4.20), words:truth(3.71),
- words:Nebula(2.96), words:Hugo(2.96), words:alien(2.96), words:die(2.96),
- words:scientist(1.25), author:Asimov(1.08).

The weight given for each feature $f$ is $\log(P(f \mid P)/P(f \mid N))$ where $P$ and $N$ represent the positive and negative class respectively.

After examining the rankings produced by the system, the user can choose examples (the user would probably want to choose ones where there is disagreement with the system) to rate. Then allow the system to use these new ratings to revise its recommendations. As with the use of relevance feedback (Salton & Buckley, 1990), this can be repeated in order to further improve recommendations.

3.5 System Details

This section relates details of the algorithms used that would be necessary for reproducing the results reported here.

3.5.1 Estimating Probabilities

This section relates the estimations and smoothing factors used for estimating the probability parameters needed to make a prediction.

Let $E_{Cat_k}$ denote $\{x \in E \mid \text{the category of } x \text{ is } Cat_k, \text{where } Cat_k \in \text{Category}\}$ and assume the notation used earlier in the paper. Then the following are the probability estimates we would use prior to smoothing.

$$P(Cat_k) = \frac{|E_{Cat_k}|}{|E|} \quad (6)$$
\[ P(f_i = t | \text{Cat}_k) = \frac{P(f_i = t \cap \text{Cat}_k)}{P(\text{Cat}_k)} \]

By definition of conditional probability,
\[
= \frac{|E_t \cap E_{\text{Cat}_k}|}{|E|} \\
= \frac{|E_t \cap E_{\text{Cat}_k}|}{|E_{\text{Cat}_k}|} \\
= \frac{|E_t \cap E_{\text{Cat}_k}|}{|E_{\text{Cat}_k}|} \tag{7}
\]

Of course, \( P(f_i = \bar{t} | \text{Cat}_k) = 1 - P(f_i = t | \text{Cat}_k) \). The weighted binary estimates according to the same estimations above. However, the matter of weighting makes how the form looks in the end slightly different; they can be estimated as follows (where \( w_r \) is the function of \( r \) mentioned above):

\[
P(\text{Positive}) = \frac{|E_{\text{Positive}}|}{|E|} \\
= \frac{\sum_r [w_r | E_r |]}{|E|}
\]

\[
P(f_i = t | \text{Positive}) = \frac{|E_t \cap E_{\text{Positive}}|}{|E_{\text{Positive}}|} \\
= \frac{\sum_r [w_r | E_t \cap E_r |]}{\sum_r [w_r | E_r |]}
\]

In order to estimate for \( \text{Negative} \) for the weighted binary classifier, simply substitute \( 1 - w_r \) for \( w_r \). When one of the probabilities in equation 3 is zero, it will dominate the whole computation by making the whole value from going to zero. It could easily happen that the true probability was not zero, but the rarity of the term or category compared to the relatively small number of input examples caused the item not to appear in the training examples. In order to account for these issues, we use smoothing methods reported in (Kohavi et al., 1997). Namely, we use a Laplace-\( m \) estimate where \( m \) is the number of training examples. In general the Laplace-\( m \) estimate is:

For \( N \) matches out of \( n \) instances for a \( k \)-valued problem,
\[
\frac{N + \frac{1}{m}}{n + \frac{k}{m}}
\]

So, for our problem we have
\[
P(\text{Cat}_k) = \frac{|E_{\text{Cat}_k}| + \frac{1}{|E|}}{|E| + \frac{|\text{Category}|}{|E|}} \tag{8}
\]
\[ P(f_i = t \mid Cat_k) = \frac{| E_t \cap E_{Cat_k} | + \frac{1}{|E|}}{|E_{Cat_k}| + \frac{2}{|E|}} \] (9)

Notice that both of these estimates approach a uniform distribution as the number of examples goes to zero and they approach the original estimates as the number of examples goes to infinity. Thus the weighting reflects the fact that we are more confident in the original estimates according to how many training examples we have.

Each novel word is given a small estimate as well which is $\frac{1}{|E||E_{Cat}|^2}$. Note that this formula weights a novel word toward a less likely a priori category. The reason for this is that the more occurrences of a category in the training examples, the less likely it is that a word associated with that category has been left out because of a skewed sample.

### 3.5.2 Real Valued Scores

In order to produce rankings, we need to obtain values that can be partially ordered. In addition, it is important that these values are normalized probability estimates. That is, we dropped the normalization term during the derivation of equation 2 as it was irrelevant when comparing probabilities for the same example since they each were weighted by that same normalization term. However since the normalization term ($P(\varepsilon)$) may differ across several examples, in order to compare probability estimates across examples, we must have comparable values. As mentioned above, we use the natural log of the posterior odds of positive to rank the predictions for the Weighted Binary and Binary Classifier. We show in the following derivation that the normalization factor drops out of the estimation of the posterior odds as follows:

By definition (Pearl, 1988),

\[
O(Positive \mid \varepsilon) = \frac{P(Positive \mid \varepsilon)}{P(-Positive \mid \varepsilon)}
\]

Since Positive and Negative are mutually disjoint exhaustive subsets of the domain,

\[
= \frac{P(Positive \mid \varepsilon)}{P(Negative \mid \varepsilon)}
\]

Substitution of Bayes Theorem yields,

\[
= \frac{P(\varepsilon \mid Positive)P(Positive)}{P(\varepsilon \mid Negative)P(Negative)}
\]

\[
= \frac{P(\varepsilon \mid Positive)P(Positive)}{P(\varepsilon \mid Negative)P(Negative)}
\]

Note that the numerator and the denominator of the final line in derivation 10 are exactly those needed to compute the Naive Bayes prediction derived in 3. Since the probabilities have already been mapped into a logarithmic space, continuing to work in the logarithmic space is easiest when
possible. So, computing the log of the final line of 10 is equivalent to simply subtracting (since they are logs) the estimate used in prediction for negative from that of positive.

In order to rank the predictions for the 10-Category Classifier, we can simply use the expected value as mentioned above. However in order to produce the expected value, we must normalize the 10 different estimates we have at this point of \( \ln [P(\epsilon | \text{Cat}_i)P(\text{Cat}_i)] \). This presents a minor problem. We began working with logarithms in the first place in order to avoid underflow, but we would like to normalize the actual estimates. If we tried to map the logarithms directly back into the exponential space, we would have underflow in some cases (this was experimentally confirmed). However, if we were to add some term \( \mu \) to each of the logarithmic estimates that enabled us to exponentiate them without underflow, the addition of this factor, since it is equivalent to multiplication in the regular space, would be naturally accounted for in the normalization process. In order to choose an appropriate \( \mu \), we simply use the maximum of the logarithmic estimates. Thus, it is guaranteed that we will not have underflow for the \( \text{Cat}_{\text{MAP}} \) estimate, and it is only if a category has magnitudes lower a posteriori probability, that it might underflow. If this were the case, the underflow term’s correct value would be so low that the computation of the expected value would have remained relatively unchanged even with the correct value.

3.5.3 Feature Selection

The definition we use for Information Gain is similar to that used in general in text categorization problems for \( j \)-category problems (Yang & Pedersen, 1997) and can easily be derived from the standard definitions of Information gain and Entropy as follows:

Assume we have the definitions used for equations 4 and 5. Furthermore, let \( T \) be a binary attribute whose values are \( \{t, \bar{t}\} \) denoting the presence or absence of a term, respectively. Let \( P(c_i | D = \sigma) \) denote the probability of randomly choosing \( x \in \sigma \) such that the category of \( x \) is \( c_i \) (\( D \) and \( \sigma \) can be thought of as a random variable over the domain of examples and some instantiation of the variable, respectively). We will abbreviate this as \( P(c_i | \sigma) \). Let \( P(t | D = \sigma) \) denote the probability of randomly choosing \( x \in \sigma \) such that \( v(T) = t \), and let \( P(\bar{t} | D = \sigma) \) be defined similarly. We abbreviate these \( P(t | \sigma) \) and \( P(\bar{t} | \sigma) \) respectively.

\[
\text{Gain}(E, T) = \text{Entropy}(E) - \sum_{v \in \text{Values}(T)} \left[ \frac{E_v}{E} \text{Entropy}(E_v) \right]
\]

\[
= \sum_{i=1}^{j} [-P(c_i | E) \log_2 P(c_i | E)] \frac{E_t}{E} \text{Entropy}(E_t) - \frac{E_t}{E} \text{Entropy}(E_{\bar{t}})
\]

Substitution of the definition of Entropy yields

\[
= \sum_{i=1}^{j} [-P(c_i | E) \log_2 P(c_i | E)] \frac{E_t}{E} \sum_{i=1}^{j} [-P(c_i | E_t) \log_2 P(c_i | E_t)]
\]

\[
- \frac{E_t}{E} \sum_{i=1}^{j} [-P(c_i | E_{\bar{t}}) \log_2 P(c_i | E_{\bar{t}})]
\]
\[
= \sum_{i=1}^{j} [-P(c_i \mid E) \log_2 P(c_i \mid E)] + \left| \frac{E_t}{E} \right| \sum_{i=1}^{j} [P(c_i \mid E_t) \log_2 P(c_i \mid E_t)] \\
+ \left| \frac{E_t}{E} \right| \sum_{i=1}^{j} [P(c_i \mid E_t) \log_2 P(c_i \mid E)] \\
\text{Since } P(t \mid E) = \left| \frac{E_t}{E} \right| \text{ and } P(\bar{t} \mid E) = \left| \frac{E_t}{E} \right|, \\
= \sum_{i=1}^{j} [-P(c_i \mid E) \log_2 P(c_i \mid E)] + P(t \mid E) \sum_{i=1}^{j} [P(c_i \mid E_t) \log_2 P(c_i \mid E_t)] \\
+ P(\bar{t} \mid E) \sum_{i=1}^{j} [P(c_i \mid E_t) \log_2 P(c_i \mid E)] \\
\tag{11}
\]

Though the computation of some of the terms here could be done at the same time as computation of estimates needed for the Naive Bayes algorithm, we have kept them separate in order to preserve abstraction. Furthermore, since the feature selection is completely done as a preprocessing phase, any features that are encountered during testing that have been feature selected out of the set are treated as novel features.

4 Experimental Results

4.1 Methodology

4.1.1 Data Collection

The first 5500 URL’s returned from the keyword search “literature fiction” were downloaded from the Amazon website and parsed into a book representation. During the processing of these 5500 URL’s, there were 28 errors. These URL’s were discarded. Of the remaining 5,472, 2,409 were classified as inadequate information pages and 3,063 were classified as adequate information pages. A page was deemed to have inadequate information if it did not contain an instance of at least one of the following slots: comments, reviews, or synopses. These pages were written to a separate file for possible use in the future as they do contain information such as title, author, subject, etc. that could be of use to the learning system. In many cases however, these pages lack so much information that it would be difficult for a user who was unfamiliar with the book to rate it and would be likewise pragmatically useless for a similar user of the recommending system. Of the 3,063 adequate information pages, two pages duplicated the ISBN’s and exact features of other titles present in the data, and as a result, these two were discarded. The remaining 3,061 titles have unique ISBN’s and form the corpus of our general fiction database. The database does contain duplicates in the sense that some titles are present as a “hardcover edition” and a “cassette edition”; while LIBRA does possess heuristics for associating related instances, they were left distinct at this time. This was done for two reasons: (1) some users may express regularities in interests such as audio vs. written, but more importantly (2) obtaining user ratings for very similar items will allow the testers to evaluate, to at least some degree, the consistency in user ratings.