Application of a Probability-Based Algorithm to Extraction of Product Features from Online Reviews

Christopher Scaffidi
June 2006
CMU-ISRI-06-111

Institute for Software Research, International
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

Prior research has demonstrated the viability of automatically extracting product features from online reviews. This paper presents a probability-based algorithm and compares it to an existing support-based approach. Specifically, I used each algorithm to extract features from 7 Amazon.com product categories and then asked end users to rate the features in terms of helpfulness for choosing products. The end users preferred the features identified by the probability-based algorithm. This probability-based algorithm can identify features that comprise a single noun or two successive nouns (which end users rated as more helpful than features comprising only one noun), yet even for collections of tens of thousands of reviews, it still executes fast enough (at around 1ms per review) for practical use.

Over one dozen colleagues helped pre-test early versions of the survey. Norman Sadeh and George Duncan provided valuable input concerning the study's design and analysis. This work has been funded in part by the EUSES Consortium via the National Science Foundation (ITR-0325273) and by the National Science Foundation under Grant CCF-0438929. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the sponsors.

Keywords: information extraction, mining, personalization, product reviews

1. Introduction

Suppose an online shopper wants a digital camera—perhaps one with good image quality and battery life. Unfortunately, at present, finding such a camera may require sifting through dozens of product reviews to locate those that discuss the right features (battery life and image quality) rather than other features (such as lenses) that are irrelevant to this shopper.

The ultimate solution might be a search tool that would accept a category name ("digital camera") and present a set of desired features ("image quality and battery life"). The user would check off the features of interest, and then the search tool would list products that impressed previous customers with those specific features, as evidenced by online reviews. Integrating such a tool into an e-commerce website like Amazon.com might help end users quickly locate the optimal product based on personal needs, potentially improving customer satisfaction and offering that website a sizable competitive advantage.

Although building such a system involves overcoming several challenges, I focus on only one of those in this paper. Specifically, I compare two algorithms for extracting the features of a product category from review text. For example, such algorithms might identify "image quality," "battery life," and "lenses" as features of "digital cameras," and they might identify "graphics" and "gameplay" as features of "PC video games." The algorithm evaluated in this paper is the feature-extraction portion of a new system called "Red Opal" ("Review Entries Digested Onto Product Attribute Levels"). It incorporates a probability model of term occurrence counts in a category's reviews. I evaluate it relative to an algorithm inspired by prior work that uses feature "support" heuristics [18].

Extracting thousands upon thousands of features and putting them into an enormous menu of product features would be counter-productive, since it would *complicate* rather than simplify the process of finding relevant products. Instead, the menu of product features should be short and valuable: the feature extraction algorithm should extract a *small number* of features that would help end users quickly find appropriate products. For such purposes, the most relevant evaluation metric is users' perception on the value of extracted features, rather than traditional metrics such as precision and recall. In addition, the algorithm should be fast enough for practical use.

To evaluate Red Opal's algorithm against the existing support-based algorithm on these criteria, I used each algorithm to extract features, and then surveyed end users to determine which algorithm's features were preferred. Specifically, I used each algorithm to generate 6 features for each of 7 product categories; combining these two sets of 42 features yielded 70 distinct features (since some were generated by both algorithms). I asked one dozen end users to rate each of the 70 features according to how helpful each feature would be for finding products during online shopping. Features identified only by Red Opal had a higher average score than features identified only by the support-based algorithm ($P<0.001$). On average, end users also favored features comprising two or more nouns, compared to features comprising one noun ($P<0.001$).

Moreover, this probability-based algorithm can process lemmatized text from tens of thousands of reviews in under a minute. Like existing support-based algorithms, the probability-based algorithm's asymptotic computational complexity is linear with respect to the number of reviews, suggesting that it would scale well even when used on larger collections of reviews.

2. Related Work

The ultimate aim is to summarize the strengths and weaknesses of products. Achieving this involves four tasks:

1. Identifying terms that characterize product features
2. Identifying which reviews evaluate products with respect to each feature
3. Using opinion words in those reviews to produce a net score for the product on each feature
4. Summarizing results in a way that helps users complete purchases.

The work discussed in this paper only addresses the first of these tasks. However, in the sub-sections that follow, I also mention work related to the other tasks in order to situate my work amid the remaining challenges.

2.1 Task 1: Identifying Feature Terms

The most relevant term-extraction work is that of Hu, Liu, and Cheng, who have tackled all four tasks listed above. In [13], they outline a feature-extraction algorithm based on heuristics that depend on feature terms' respective occurrence counts (called the "support" of the terms); they extend this algorithm in [18] to deal with a specific "pro vs. con" review format (which does not occur on Amazon.com). In Section 4, below, I use a simplified version of this algorithm as a baseline for evaluating Red Opal's probability-based feature-extraction algorithm.

Other researchers in this area have used heuristics which rely only on occurrence counts (such as the heuristics proposed in [23] for extracting features comprising multiple nouns), but there also exists a large collection of algorithms that incorporate probability and statistics theory, as well (as reviewed by [15]). One ingredient in these algorithms is the recognition that

documents are best characterized by terms which occur more densely in those particular documents than in random English text (a fact noted as early as 1961 [7]). The other ingredient is a probability distribution function that models how often words occur; the Poisson distribution is one time-honored and popular option first proposed in [5], then elaborated in [31] and [10]. I have incorporated these probability-based ingredients into Red Opal's feature-extraction algorithm, as discussed in Section 3.3.

A third approach adopts an information-theoretic view that represents entities (such as documents and terms) as points in an abstract space, then infers relatedness among those entities based on distances separating them. For example, many document-clustering methods represent documents as weighted bit vectors whose coordinates correspond to the presence or absence of terms; this approach assigns documents to clusters based on whether they contain certain key terms, which are selected in order to maximize the inter-cluster distances while minimizing intra-cluster distances [28].

In the dual problem, term-clustering methods depend on the insight that frequent co-occurrence of terms implies that they may have similar subject matter [3]. Matsuo and Ishizuka incorporated this insight into an algorithm that blends the information-theoretic approach with the probability-based approach; this algorithm can reliably extract keywords from a single article (in contrast to the aforementioned algorithms, which operate on a corpus of documents) [21]. In addition, Morinaga and Tateishi have developed an information-theoretic approach to categorizing opinions about products, but unlike that of Matsuo and Ishizuka (and unlike Red Opal's probability-based algorithm), it requires some supervised training on practice data beforehand [21], which could limit its adoption due if it needs to be retrained for each category of products.

Historically speaking, except for the aforementioned [13], [18] and [22], researchers have generally designed terminology-extraction algorithms for use on specialized or technical text rather than conversational text such as product reviews. Example domains include articles discussing cold war politics [5] as well as titles and abstracts of physics articles [27]. Researchers have developed specialized algorithms for searching speech, which has posed efficiency challenges as well as theoretical challenges related to properly handling homophones, "document" length, and normalization [14]. More recently, other researchers have extracted topic terms from blogs, which have a conversational style that yields an interesting mixture of common and proper nouns [9]. Each of these applications of term-extraction to specialized domains has presented special challenges that motivated researchers to blend multiple approaches into their algorithms. Thus, it would not be surprising in the case of a new specialized domain, product reviews, if the ideal algorithm also turns out to be a blend of approaches.

2.2 Task 2: Identifying Relevant Reviews

Once product features have been identified, the reviews can be indexed, then searched and ranked in terms of their relevance to each product feature. When designing algorithms to achieve this task, researchers may find inspiration in the wealth of prior research in the area of document-retrieval, since numerous term-weighting and document-ranking methods have already been proposed and evaluated [25], including the well-known TFIDF method [33].

2.3 Task 3: Scoring Products Using Opinions

After product features are identified and relevant reviews are retrieved for each product feature, the next task is to assess how positive or negative each review is with respect to the feature, and then to combine those scores into an overall product score with respect to the feature.

Some researchers advocate the use of online resources like WordNet or other thesauri and lexica to enable easy identification of frequently occurring opinion words [12] [29] [32]. This approach relies on the insight that when a section of text contains opinionated words, then those words usually reflect positively or negatively on the subject of the text [24]. Unfortunately, if a word happens not to appear in the thesaurus or lexicon (perhaps because it is a specialized term, such as "thrashing" in the context of disk drives), then it would slip past the algorithm. To sidestep this issue, other researchers have attempted to use supervised or semi-supervised machine learning to build a context-aware model of which words convey opinions [6], though of course the learned patterns may not generalize to reviews of products outside the training corpus. Finding an optimal balance between these approaches appears to be an open research topic—as is the problem of determining what segment of the review each opinion word applies to, though existing approaches using lexical chains offer a start [1].

2.4 Task 4: Presenting Results to End Users

Finally, once a system has calculated products' scores for each feature, how should it present those scores to end users? Hu, Liu, and Cheng have proposed a graphical approach involving bar graphs [18], while Dave's system presents review snippets [6].

However, to take fuller advantage of end users' ability to read, designers of future systems may consider generating natural language summaries. Researchers have already made excellent progress in solving sub-problems related to this task: identify-

ing the documents' main semantic constituents [20] (even when they span many documents [26]), joining segments of text so they sound natural [16], and presenting results in a capsule format [2].

System designers may still want to provide machine-readable numeric product-feature scores in addition to human-readable summaries, since this could facilitate the creation of software agents that seek out products with specific attributes. At present, such "shopbots" are mainly intended to help shoppers find low prices [30], though some newer shopbots allow users to search for products based on product features [4]. However, these features currently appear to be manually coded on a per-product-category basis by humans (rather than automatically inferred, as in the case of Red Opal's algorithm), which may be an expensive proposition for large, diverse product catalogues.

3. Two Algorithms for Extracting Product Features

In this section, I outline two algorithms for extracting feature terms from a corpus of reviews of products in a category.

The first algorithm is an existing support-based algorithm. That is, it depends on heuristics concerning how often combinations of words occur in the collection of reviews. It was developed by Hu, Liu, and Cheng [12] [13] [18], with some simplifications noted below. The second algorithm is probability-based. Specifically, it estimates the probability that densely occurring review terms would appear at a high frequency in random English text.

As discussed in Section 4.3.1, I then blended a list of terms extracted by the support-based algorithm with an analogous list extracted by Red Opal's algorithm, and end users rated the usefulness of each term for finding products.

3.1 Preconditions for the Algorithms

Some preprocessing must occur before the text can be fed into the two algorithms discussed below. The first preprocessing step is to retrieve the reviews' text from the Amazon.com XML web services and to cache the text into a local MySQL4 database.

The other preprocessing step is to run each review's text through a part-of-speech (POS) tagger, which appends the respective POS to each word. In addition, the tagger appends the dictionary form of the word, known as its "lemma." I used the free MontyLingua POS tagger [19], which transforms a sentence like "I played this game for hours" into "I/PRP/I played/VBD/play this/DT/this game/NN/game for/IN/for hours/NNS/hour /././"[1] The tagged words of each review are then stored in the local database cache.

3.2 Support-Based Algorithm

This support-based algorithm's input is a list of POS-tagged reviews for products in a single category. The output is a list of terms that represent likely features of products in this category. This algorithm is similar to that described by Hu and Liu in [13].

The purpose of defining this support-based algorithm is to provide a baseline of comparison in Section 4. This evaluation is intended to determine which algorithm would be preferable for populating a short menu of product features in a new search tool. Thus, this algorithm is optimized to produce a *short, high-quality* list of features. Unlike the algorithm developed by Hu and Liu, it does not attempt to produce an *exhaustive* list of the products' features.

3.2.1 Algorithm Specification

The algorithm splits each review into sentences, each of which contains a series of words. A word occurrence is considered a noun if the POS tagger marked it as a common noun and, in addition, the corresponding lemma is not a stop word. Here, a lemma is considered a stop word if it occurs in a baseline lemma-frequency table [17] at least as often as the word "how"; this cutoff eliminates words like "I," "you," and "now."

As with prior research, this support-based algorithm is built on the assumption that product features comprise one or more nouns that are not necessarily adjacent but at least within the same sentence [13] [23]. So for each sentence, the algorithm identifies unordered "lemma sets" containing 1, 2 or 3 noun lemmas[2]. For example, in the sentence "The memory card costs fifty dollars," the 6 lemma sets of potential interest are {"memory"}, {"card"}, {"dollar"}, {"card", "memory"}, {"dollar", "memory"}, {"card", "dollar"}, and {"card", "dollar", "memory"}.

---

[1] MontyLingua uses the same tags as the Penn TreeBank. Refer to http://www.cis.upenn.edu/~treebank/ for a POS tag manual.

[2] The support-based algorithm originated from an extension of association mining, which uses the more generic term "item sets" in place of "lemma sets." [13]

A lemma set z has "support" s_z if z occurs in s_z sentences. If a lemma set appears in a sentence, then supersets of that lemma set might also appear; a lemma set z has "proper support" ps_z if z appears in ps_z sentences that do not contain a superset of z.

The algorithm loops through all sentences, incrementing s_z and ps_z for each lemma set z that occurs. Finally, the algorithm returns the 6 lemma sets with the highest proper support.

For the purposes of the evaluation described in Section 4.3, I manually organized the lemmas of each set into a string of text that seemed to make the most sense. If a lemma set contained the name of the category, then I removed the category name. (For example, I manually transformed the lemma set {"camera", "card", "memory"} to "camera memory card" and then reduced it to "memory card.") If a lemma set contained *only* the category name (such as "camera"), then I manually eliminated that lemma set and brought the next best replacement into the results.

3.2.2 Simplifications Relative to Previous Work

The algorithm above simplifies Hu and Liu's original algorithm (described in [13]) in four ways.

First, Hu and Liu mention fuzzy-matching of lemmas to help correct for misspelled words. Refining the baseline algorithm in this manner could theoretically improve the quality of the resulting feature list—but perhaps not substantially, since surprisingly few words were misspelled. (For example, in the data set described in Section 4, of approximately 60000 noun occurrences, fewer than 4800 occurrences involved terms with a frequency of ≤ 1 part per million in a corpus of English text [17].)

Second, the algorithm in [13] only considered lemma sets if they occurred in at least 1% of all sentences. In practice, if a lemma set was one of the top 6 lemma sets generated by this support-based algorithm (as judged by proper support), then it generally exceeded the 1% criterion by a large margin. Hence, applying a 1% minimal support filter was unnecessary here.

Third, the algorithm in [13] filters lemma sets based on a measure called "compact support." Although a lemma set is unordered, it occurs within a sentence in a certain order. If a lemma set z contains k lemmas, and if z appears in a sentence, then the sequence of lemmas would be $<n_1, n_2, \ldots, n_k>$. Lemma set z is "compact" in the sentence if the word distance between any two adjacent lemmas (n_i and n_{i+1}) in the lemma set does not exceed 3. The algorithm in [13] filters out any lemma sets that are not compact in at least 2 sentences, whereas the simple support-based algorithm does not apply such filtering. In general, the top 6 lemma sets had extremely high compactness, so there was little need to apply this filter, as it would have affected only 1 out of 42 features extracted from 7 product categories in Section 4.

Last, after Hu and Liu extract features using an algorithm like the one above, they then examine nouns that occur near opinion words. This allows them to augment their list of frequent features with infrequent features, which boosts their algorithm's recall when tested on a sample corpus. But the simple algorithm above generates more features than could possibly be presented to end users in a menu, so generating even more features (particularly infrequent ones) was undesirable.

Because of these differences, at the cost of additional complexity, the algorithm of Hu and Liu is suited for exhaustively listing all product features. However, the simple support-based algorithm above is adequate for the purpose of generating a short menu of product features, and for providing a baseline to evaluate Red Opal's probability-based algorithm.

3.3 Red Opal's Probability-Based Algorithm

Red Opal's feature-extraction algorithm examines a collection of reviews and uses probability-based criteria to identify terms that represent product features. The inputs are the name of a product category, such as "digital camera," and a list of POS-tagged reviews for products in that category. This category name could be identified manually by a human or identified automatically by textual summary techniques [2]. The output of Red Opal's feature extraction algorithm is a list of terms that represent likely features of products in this category.

3.3.1 Configure with Baseline Frequency Data

The algorithm utilizes a sizeable bundle of configuration information that represents the baseline frequency of word lemmas in English. Specifically, for each lemma in English, this configuration indicates how often that lemma occurs as a noun or as any other POS (in parts per million). For example, of every 1 million words, "trouble" or its variants occur 111 times, including 99 times as a noun and 12 times as another POS [17].

This configuration should come from statistics on text written in roughly the same style as the reviews under examination. In the case of Amazon.com, the reviews generally have a conversational flavor, almost as if the writer were actually speaking rather than typing. Consequently, I configured Red Opal to use lemma-frequency data derived from a 100 million-word corpus based on a blend of spoken and written text [17]. If Red Opal were used on reviews with a different writing style (for example, manufacturers' product descriptions, or reviews of academic papers), then it would be desirable to use another baseline lemma-frequency configuration based on a corpus that more closely matches that writing style.

3.3.2 Step 1: Calculate Single-Lemma Statistics

Prior research has strongly suggested that technical terms and product features are generally nouns [23], so the first step is to create a normalized frequency table that only includes occurrences of lemmas as nouns. Let b_x represent the baseline occurrence rate of lemma *x* as a noun. (For example, $b_{\text{"trouble"}} = 99$ in the example above.) Then Equation 1 gives the probability p_x that a randomly selected noun would equal lemma *x*, where *k* ranges over all lemmas in the baseline lemma-frequency table.

$$p_x = b_x \div B, \quad \text{where } B = \sum_k b_k \qquad \textit{Eq 1}$$

As a practical consideration, not all words in reviews also appear in the baseline lemma-frequency data. This absence from baseline data occurs for two main reasons. First, the word may be a misspelling that is so rare that it did not happen to occur in the baseline corpus (e.g.: "professnial"). Alternatively, the word might represent a legitimate lemma that is so specific to the product category that it could not have occurred in the baseline corpus unless that corpus happened to include a discussion of such products (e.g.: "megapixel"). In cases where Red Opal cannot use Equation 1 to determine p_x for a lemma, it defaults p_x to the unweighted average p_x of all lemmas (approximately 1.05E-4); the implications of this assumption are discussed below. (As a separate problem, due to rounding in the baseline lemma-frequency table, b_x may be present but "equal" to 0 for some lemmas; in such cases, to avoid multiplying or dividing by zero, Red Opal defaults p_x to 1/B in Equations 3 through 7 below.)

Red Opal also counts how often each lemma appears in the reviews as a common noun. However, the POS-tagger sometimes makes mistakes, so Red Opal discards any lemma that is listed in the baseline frequencies table as being more likely to occur as a non-noun than as a common noun (such as "roll"). After applying these filters, Red Opal counts how many times each lemma occurs as a noun in the reviews of products in this category. Let n_x represent the number of occurrences of lemma x. Then Equation 2 gives the total number of noun occurrences N.

$$N = \sum_k n_k \qquad \textit{Eq 2}$$

3.3.3 Step 2: Calculate the Probability of Observed Lemma Occurrence Counts

What the reviews and the baseline corpus have in common is their conversational style. What sets them apart is the fact that the reviews focus on a specific topic, which is the category of product under consideration. Consequently, some words occur far more frequently in the review text than would be expected in a random section of English text of equal length.

That observation forms the basis for Red Opal's term-selection criterion. For each lemma, this criterion asks the question, "Would this lemma have occurred so often in random English?" If the answer is "probably not," then the term might be a feature of the products under discussion.

Answering this question requires selecting a model for calculating the probability that a lemma *x* would occur n_x times in a random piece of English text containing a series of N noun occurrences $\langle w_1, w_2, \ldots, w_i, \ldots, w_N \rangle$. Two assumptions facilitate the construction of this model:

- It is assumed that the occurrence of lemma *x* in position *i* is independent of whether lemma *x* occurs in any other position *j*. That is, $P(w_i = x \mid w_j = x) = P(w_i = x)$.
- It is assumed that the occurrence of lemma *x* in position *i* is independent of the value of *i*. Then $P(w_i = x) = p_x$.

Strictly speaking, these assumptions are false. For example, English speakers almost never use the same noun twice in a row, so the first assumption is false. In addition, for example, some nouns like "hotness" almost never occur as the first word in a sentence, so the second assumption is false.

Despite these qualms, researchers often accept these assumptions for two reasons. First, the assumptions do not necessarily harm the quality of results [33]. Second, the assumptions help simplify the problem so it is computationally tractable [15].

Under these assumptions, the binomial distribution represents the probability that lemma *x* occurs n_x times in a series of N nouns:

$$P(n_x) = \frac{N!}{(N-n_x)!\,n_x!}\, p_x^{\,n_x} (1-p_x)^{N-n_x} \qquad \textit{Eq 3}$$

For large N and small p_x, the Poisson distribution approximates a binomial distribution:

$$P(n_x) \approx \frac{(p_x N)^{n_x} \cdot e^{-(p_x N)}}{n_x!} \qquad \textit{Eq 4}$$

To avoid numeric underflow, Red Opal calculates the logarithm of $P(n_x)$ rather than $P(n_x)$. Taking the logarithm of Equation 4, applying Stirling's approximation, and using the fact that $p_x << 1 << N$ yields Equation 5.

$$\ln(P(n_x)) \approx (n_x - p_x N) - n_x \ln(\frac{n_x}{p_x N}) - \frac{\ln(n_x)}{2} \qquad \textit{Eq 5}$$

If $n_x > p_x N$ and $\ln(P(n_x))$ is small, then it is unlikely that so many occurrences of *x* could occur by chance. In actual use, the probability in Equation 5 is extremely small for product feature terms. For example, in one sample of 1690 digital camera reviews, which included N = 58543 noun occurrences, the lemma *x* = "lens" occurs n_x = 1174 times, even though "lens" constitutes a proportion of only p_x = 3.1E-5 of all noun occurrences in English. As a result, $\ln(P(n_x))$ = -6429. This is an astonishingly small probability, leading Red Opal to identify "lens" as a likely feature term.

As a practical consideration, when p_x cannot be determined because lemma *x* does not appear in the baseline lemma-frequency table, then Red Opal defaults p_x to the average of all p_x values among English nouns. This sets a relatively high bar for how many times a word must be misspelled before it appears to be a product feature. Yet this is not too high of a bar to overcome in the case of technical terms that truly do deal with the product category. For example, the word "megapixel" occurs 195 times in the 1690 reviews of digital cameras, even though it does not appear in the baseline lemma-frequency table at all. Using the default p_x of 1.05E-4 yields $\ln(P(n_x))$ = -488. From this, it might be suspected that products could be usefully categorized based on whether their reviews contain the word "megapixel."

3.3.4 Step 3: Calculate the Probability of Observed Bigram Occurrence Counts

Researchers have noted that most technical terms are compound nouns [23]. An adaptation of the above derivation applies to estimating the probability that two nouns would occur successively a certain number of times as a bigram.

If *x* and *y* are nouns, let the rate of occurrence of bigram <*x, y*> among noun bigrams be denoted ρ_{xy}. (For example, if ρ_{xy} = 1E-3, then 1 out of every 1000 noun bigrams would be <*x, y*>.) Now, viewing a bigram as a 2-noun segment of English text, and using the second assumption in Section 3.3.3, the probability is p_x that *x* is the first noun of a randomly selected bigram, and the probability is p_y that *y* is the second noun of the bigram. Then, using the first independence assumption in Section 3.3.3, the probability of the bigram as a whole is the product of these, so $\rho_{xy} = p_x p_y$.

If a bigram <*x*, *y*> occurs a total of η_{xy} times in the corpus, then Equation 6 gives the total number of bigram occurrences H.

$$H = \sum_{x,y} \eta_{xy} \qquad \textit{Eq 6}$$

Carrying over the line of reasoning in Section 3.3.3 leads to the logarithm of the probability $P(\eta_{xy})$ of observing η_{xy} occurrences of <*x*, *y*> in H randomly selected English bigrams.

$$\ln(P(\eta_{xy})) \approx$$

$$(\eta_{xy} - \rho_{xy} H) - \eta_{xy} \ln(\frac{\eta_{xy}}{\rho_{xy} H}) - \frac{\ln(\eta_{xy})}{2} \qquad \textit{Eq 7}$$

To compute this quantity, Red Opal first counts how often each bigram <*x*, *y*> appears in the category's reviews, which is η_{xy}. It computes H as the sum of all η_{xy}, then computes $\rho_{xy} = p_x p_y$ for each bigram that occurred. Combining these using Equation 7 yields the logarithm of the probability of seeing so many occurrences of each bigram in English text. If $\eta_{xy} > \rho_{xy}$ H and $\ln(P(\eta_{xy}))$ is very small, then it is unlikely that so many occurrences of <*x*, *y*> could have occurred by chance. This implies that <*x*, *y*> might be worth presenting as a feature of the products under discussion.

3.3.5 Filtering Words Inherent to the Category

For each lemma *x* and each bigram <*x*, *y*> in the category's reviews, Red Opal estimates the probability of observing the lemma or bigram a certain number of times in a random piece of English text. However, some terms appear extraordinarily often yet offer no information about the features of a product category.

For example, 92% of the reviews on digital cameras contain the lemma "camera." Indeed, it is hard to imagine any review writer discussing a camera without mentioning the word "camera" (or a misspelling such as "camara" or an approximate synonym such as "Powershot A70"). Since "camera" applies to all products in the category, it does little to help customers distinguish among products within the category.

One of Red Opal's inputs was the name of the category, such as "digital camera" or "grill." Red Opal discards feature terms and bigrams comprised solely of words from the category name. For example, in the "digital camera" category, Red Opal would discard "digital," "camera" and "digital camera," but it would keep "film camera" and "camera shutter."[3]

3.3.6 Final Selection of Features

After calculating the probabilities for each lemma and bigram in the reviews, Red Opal identifies a final list of terms. The obvious approach is to combine the most promising lemmas and bigrams into a short list, sort the list in descending order of probabilities, and return the best-scoring terms. In practice, this is the approach that Red Opal uses, which produces a blend of single-lemma and bigram features.

However, in order to assess if end users found bigrams to be substantially more useful than single-noun lemmas (as described in Section 0), I modified Red Opal's algorithm to ensure that it returns a good supply of both types of term. Specifically, I modified Red Opal to return the 3 bigrams that have the lowest $P(\eta_{xy})$, as well as the 3 lemmas that have the lowest $P(n_x)$ but that do not appear within any of the 3 bigrams as a component noun. For example, if it identifies "battery life" as a feature of digital cameras, then it would not include "battery" as well, even if "battery" has a strong probability score.

Note that this modification forces Red Opal to return some product feature terms that might not have had the lowest log-probability score. In other words, it forces Red Opal to return possibly inferior terms. Consequently, the evaluation in Section 4 probably represents a lower bound for the perceived quality of the features returned by Red Opal.

4. Comparison of the Algorithms' Perceived Utility by End Users

Algorithms that extract product features are an essential building block for a new type of search engine. Such a search engine would let users rapidly find products with "the right features," rather than sifting through hundreds of reviews.

Note that this application would allow users to rapidly navigate to relevant products, perhaps through a simple drop-down widget or a short menu. Extracting thousands upon thousands of features and putting them into such a menu would be counter-productive, since it would *complicate* rather than simplify the process of finding relevant products. Instead, the feature extraction algorithm must extract a *small* number of features that would help users quickly find appropriate products.

Therefore, a primary metric for the algorithm's success is whether end users perceive a short list of feature terms to be helpful guideposts when searching for products.

4.1 Survey Overview

Survey respondents were classmates in a Carnegie Mellon University e-Commerce class. They were a mixture of undergraduates and graduate students and thus represented a key online market segment: well-educated 20-to-30-year olds who are highly likely to purchase products online.

The survey instructed respondents to imagine that they were shopping at Amazon.com for a birthday gift for a friend. The questions asked whether searching for each feature term would be helpful for finding products. For example, in the case of digital cameras, I asked whether searching for each term "*might* help you pick out the right DIGITAL CAMERA for a friend." Respondents could score each term using the following scale:

- Definitely Helpful (coded with a value of 3)
- Probably Helpful (coded with a value of 2)
- Probably Not Helpful (coded with a value of 1)
- Definitely Not Helpful (coded with a value of 0)

When interpreting the meaning of these responses, note that different respondents have different friends, who in turn need different features. Consequently, it is certainly not reasonable to expect that terms would score well with *all* respondents. However, terms scoring *well with many* respondents would probably prove helpful to a range of similar customers at an e-Commerce site. Conversely, terms that score uniformly poorly would probably not prove useful with a wide range of customers. The goal, therefore, is to find the algorithm that generates higher-scoring terms.

[3] Future versions may use an information-theoretic approach to removing terms that do not help distinguish among products.

4.2 Term Extraction and Data Details

The survey evaluated terms for each of 7 categories shown in Table 1. I selected these categories because students are likely to be familiar with such products. (Note that I downloaded a subset of all products and reviews, rather than the entirety of the Amazon.com catalog for each product category.)

Table 1: The 7 product categories covered by the survey

Category	Number of Products in Category	Number of Reviews in Category	Number of Feature Terms
Digital Cameras	110	1690	10
DVD Players	100	808	11
E-Commerce Books	110	321	10
Grills	15	208	12
PC Video Games	110	1277	8
PS2 Video Games	110	594	9
Watches	110	108	10

For each category, I used the support-based algorithm and the probability-based algorithm to extract 6 features each. Thus, although the term list for each product category could conceivably include up to 12 terms, each term could be identified by both algorithms, so some product categories had fewer than 12 (distinct) terms. For example, the survey's question related to digital cameras presented the 10 terms shown in Table 2.

Table 2: The digital camera features evaluated in the survey

Term	Produced by Support-Based Algorithm	Produced by Probability-Based Algo-rithm
Battery	Yes	No
Battery life	No	Yes
Feature	Yes	No
Image quality	Yes	No
Lens	No	Yes
Memory card	Yes	Yes
Photos	No	Yes
Picture	Yes	No
Picture quality	Yes	Yes
Shots	No	Yes

I placed the terms for each category into alphabetical order so there was no indication to respondents that the terms originated from different algorithms. I recruited respondents by email and presented the survey through a web site to help prevent bias through face-to-face contact.

Although 12 people took the survey (a response rate of 41%), 1 respondent entered "Definitely Helpful" for every single feature, and another respondent's answers correlated very poorly with those of the remaining 10 respondents (specifically, a correlation of 0.18 with the mean of the other 10 respondents' answers). Consequently, I dropped the data from these 2 respondents. The scores from the remaining 10 respondents were quite consistent (demonstrating a standardized Cronbach alpha of 0.89 overall).

4.3 Survey Results

4.3.1 Support- vs. Probability-Based Algorithm

To assess the relative utility of the algorithms, I used SAS 9.1.3 SP2 to compare the scores of features generated by each algorithm. Of the 70 features, 28 were generated only by the support-based algorithm, and 28 were generated only by the probability-based algorithm. (The remaining 14 were generated by both.) Since 10 people rated each feature, this generated 560 ratings of features generated by only one algorithm.

The probability-based algorithm performed better than the support-based algorithm in terms of generating features with high perceived utility. Specifically, averaging the 280 scores for features generated only by the probability-based algorithm yielded a mean score of 1.69 (S=1.07), whereas averaging the 280 scores for features generated by the support-based algorithm yielded a mean score of only 1.40 (sample standard deviation S=1.03). This difference was statistically significant under a t-test ($t=3.17$, $df=558$, $P<0.001$).

Of course, as shown in Table 3, including the 14 features generated by both algorithms, for a total of 420 scores per algorithm, also yielded a better average score for features extracted by the probability-based compared to those extracted by the support-based algorithm. However, it is not acceptable in this case to apply t-tests to check for statistical significance, since including the 14 features generated by both algorithms would violate the t-test's assumption of independent populations.

Table 3: Findings concerning perceived feature helpfulness

Feature	Mean Score	Sample Std Dev (S)	Number of Scores (N)
Features generated *ONLY* by support-based algorithm	1.40	1.03	280
Features generated *ONLY* by probability-based algorithm	1.69	1.07	280
ALL features generated by support-based algorithm	1.65	1.05	420
ALL features generated by probability-based algorithm	1.79	1.07	420

The 140 scores for features identified by both algorithms had a mean score of 2.01 (S=1.04). This is quite a bit higher than the scores for features generated by either algorithm alone. Consequently, the ideal algorithm might select for features that have a high support and also a high probability-based score.

4.3.2 Single-Noun vs. Compound Features

To assess the relative utility of generating features comprising compound nouns, I compared the scores of features containing one noun to the scores of features containing multiple nouns. Of the 70 features, 44 contained one noun, 24 contained two nouns, and 2 contained three nouns. Since 10 individuals rated each feature, the survey thus generated a total of 440 scores for single-noun features and 260 scores for compound features.

The compound features generally received higher scores of perceived utility. Specifically, averaging the 440 scores for compound features yielded a score of 1.98 (S=1.02), whereas averaging the 260 scores for single-noun features yielded a mean score of only 1.43 (S=1.05). This difference was statistically significant under a t-test ($t=6.73$, $df=698$, $P<0.001$).

This suggests that the ideal algorithm might preferentially select for feature terms that are compound nouns.

5. Evaluation 0f Efficiency

In order to evaluate whether the probability-based algorithm's improved utility comes at the cost of unbearably higher computation time, I ran it on several collections of reviews, ranging from 100 to 32000 reviews. Timing measurements revealed that it runs orders of magnitude faster than the POS-tagger that is used to pre-process input prior to running the feature-extraction algorithm. This implies that even if POS-tagging becomes significantly faster, then the probability-based feature-extraction algorithm would still be unlikely to become a bottleneck. Moreover, because the algorithm's asymptotic computational complexity is linear, it is likely to perform adequately even on still larger collections of reviews.

5.1 Execution Time

I tested execution times on a commodity desktop machine, an Optiplex GX270 equipped with 1GB of RAM and a single Pentium 4 CPU running at 3 GHz. The operating system was Windows XP Professional SP 2. The database server was MySQL 4.1.14 running the InnoDB engine. Applications ran on Sun's Java runtime environment, standard edition, version 1.4.2_09-b05, with HotSpot enabled and a heap size of 250 MB.

On this configuration, running the MontyLingua POS-tagger on 1690 digital camera reviews took an average of 301 ms per review. Repeating this measurement using other review sets of comparable size produced some variability in the time to tag and lemmatize text; however, in no case did the execution time fall below an average of 250 ms per review.

In contrast, the probability-based algorithm processed hundreds of reviews per second. Specifically, I ran it on each of the 7 product categories listed in Table 1. The probability-based algorithm processed the 108 reviews of the smallest category in 3 seconds; it processed the 1690 reviews of the largest category in 7 seconds. For each category, Red Opal averaged less than 30 ms per review.

To assess the algorithm's throughput for still higher numbers of reviews, I downloaded 32000 reviews of paperback fiction books from Amazon.com and ran the algorithm on increasingly larger subsets of reviews. Specifically, I ran the algorithm 4 times for each of the following 7 collection sizes: 1000, 2000, 4000, 8000, 16000, 24000, and 32000 reviews. I averaged each set of four measurements to produce one of the 7 points in Figure 1, which depicts how execution time increases with number of reviews.

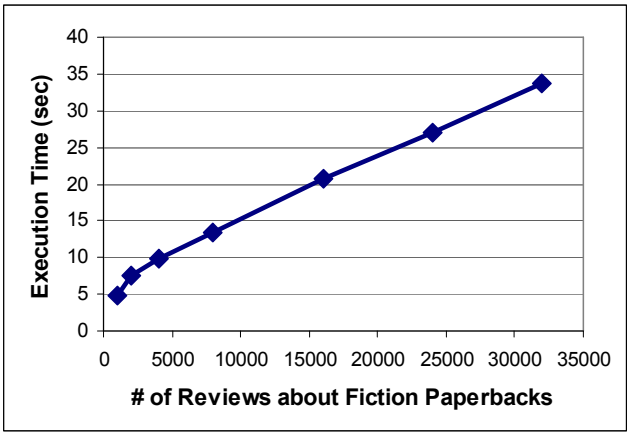

Figure 1: Empirical Execution Time vs # of Reviews

As shown in Figure 1, although the curve in Figure 1 exhibits nonlinearity for small collections, it becomes linear for larger collections of reviews. (I examine the reason for this in Section 5.3.) Using SAS 9.1.3 to fit a linear equation to the 28 data points reveals that the execution time in milliseconds approximately equals $5545 + 0.899 \cdot n$, where n is the number of reviews (R^2=0.988, F=2227, df=1, P<0.001). At about 0.9 ms per review, this algorithm is approximately 300 times as fast as the POS pre-processor, so Red Opal's feature-extraction algorithm is unlikely to become a bottleneck when run on tens of thousands of reviews.

5.2 Related Statistics on Occurrence Counts

Processing these reviews provides an opportunity to collect statistics on how the number of reviews relates to the number of lemmas, bigrams, and lemma sets in the corpus. These numbers are extremely valuable since they can help designers of new feature-extraction algorithms make decisions about how to optimize their algorithms to ensure high performance. (For example, deciding whether to implement a cache in memory or on disk depends on the cache's projected size.)

It is clear that the total number of noun lemma occurrences (N in Section 3.3.2) should increase linearly with respect to the number of reviews. It is also clear that the total number of noun bigram occurrences (H in Section 3.3.4) and the total number of 1-, 2-, and 3-noun lemma set occurrences should likewise rise linearly.

What is more interesting is how the number of *distinct* noun lemmas, noun bigrams, and lemma sets each rise with respect to the number of reviews. It is known that the number of distinct lemmas rises sub-linearly with respect to corpus size, as described by Herds' Law [8] in Equation 8. Here, N is the total number of lemmas, n is the total number of documents, and c and γ are each constants (with $0 < \gamma < 1$).

For many types of stories written in English, γ is around 0.79 (and c varies depending on the average document length) [8].

$$N = c \cdot n^{\gamma} \qquad \textit{Eq 8}$$

Surprisingly, γ is somewhat smaller for reviews on Amazon.com, meaning that the number of distinct terms does not grow as quickly as would be expected for English in general. Specifically, using the paperback reviews described in Section 5.1 and letting n range from 1 through 32000, I tracked the number of distinct noun lemmas as a function of the number of reviews,

as depicted in Figure 2. Using SAS to fit this data to Equation 8 revealed that c=121, γ=0.506 (F=2.55e7, df=2, P<0.001). This value of γ is notably smaller than the value for stories written in English [8].

Reviewing the data revealed that the total number of distinct noun bigrams and the total number of lemma sets also fit Herds' Law extremely well. Specifically, fitting these data to Equation 8 revealed that c=7.79, γ=0.797 when counting the number of distinct noun bigrams (F=1.52e7, df=2, P<0.001), and c=304, γ=0.923 when counting distinct lemma sets (F=1.56e7, df=2, P<0.001).

In other words, the number of distinct lemma sets almost grew linearly with respect to the number of reviews. Thus, designers may need to take care to avoid scalability problems when designing algorithms (such as Hu and Liu's support-based algorithm) that will operate on lemma sets over tens of thousands of reviews.

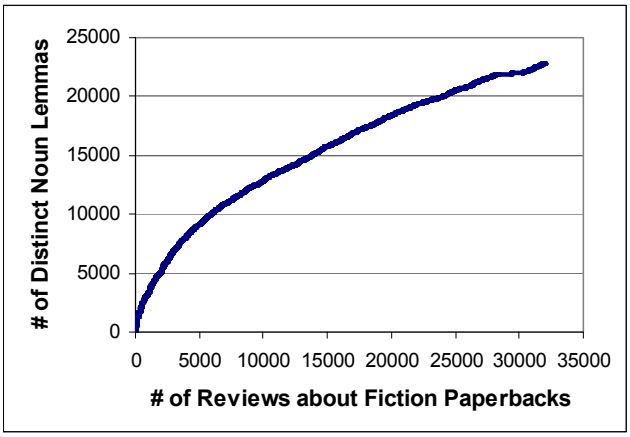

Figure 2: # of Distinct Lemma Nouns vs # of Reviews

5.3 Analysis of Computational Complexity

Existing implementations of the support-based approach generally can run in linear time, with respect to the number of reviews. For example, in [13], Hu and Liu use a version of the Apriori algorithm, which loops through all lemma sets containing a single noun and generates lemma sets containing two nouns; it then loops through sets containing two nouns and generates those containing three nouns. At each step, a lemma sets are discarded unless they have some minimal support at that step; for example, the implementation in [13] requires lemma sets to appear in at least 1% of all sentences. This threshold could be lowered if needed so that the algorithm takes an acceptable amount of time. However, as noted in Section 5.2, the number distinct noun lemma sets rises linearly with respect to the number of reviews, so the support-based algorithm can run in linear time.

Red Opal's probability-based algorithm also runs in linear time with respect to the number of reviews. Its main operation is to loop through each noun lemma *y* and increment the corresponding occurrence count n_y. If *y* immediately follows another noun lemma *x* in the review text, then Red Opal also increments the occurrence count η_{xy} for the bigram <*x*, *y*>. Thus, the algorithm visits each lemma occurrence exactly once.

Then in the next step, Red Opal calculates the probability described in Section 3.3 for each distinct lemma and each distinct bigram that occurs in the review corpus; as noted in Section 5.2, this number of distinct lemmas and bigrams rises sublinearly with respect to the number of reviews. So overall, for very large numbers of reviews such as the hundreds of thousands in the Amazon.com database, the execution time of Red Opal increases linearly.[4]

In summary, the average amount of time to process each review is roughly a constant, regardless of the total number of reviews that the algorithm processes. Thus, the throughput demonstrated in Section 5.2 would not degrade due to non-linearity as the number of reviews increases. As a result, Red Opal's feature-extraction algorithm would likely be able to keep up with the POS-tagging pre-processor even for much larger sets of reviews.

[4] Note that similar arguments apply to usage of other computer resources, including memory and bandwidth.

6. Conclusion and Future Work

Repositories of product reviews represent a valuable asset, since online retailers could conceivably leverage them to guide shoppers automatically to products with key features. The first step toward achieving this goal is to extract feature terms from these reviews—for this, Red Opal's probability-based algorithm has demonstrated improved utility compared to a support-based algorithm, without sacrificing scalability.

Yet four steps remain before the ultimate goal can be obtained.

First, Red Opal's probability-based algorithm should be evaluated against promising information-theoretic algorithms such as that of [22]. The ideal algorithm may be a hybrid of the two, offering high quality feature-extraction without requiring any supervised machine learning.

Second, existing document retrieval research could be tapped for ideas on how to match reviews optimally with product features. Existing TFIDF or "bag of words" algorithms may need alteration to accommodate traits particularly common to reviews, such as implicit references to features.

Third, current research on extracting and weighting opinion words requires extension and generalization. As in the case of feature-extraction, the ideal algorithm would accurately identify the most useful terms and assign appropriate weights without requiring supervised machine learning.

Finally, there are a wide variety of options for presenting the final search results, including charts, snippets, natural language summaries, and raw numbers (for use by shopbots). At this point, it is not clear which interface would be preferred by end users for various shopping scenarios. As with the other tasks mentioned above, significant innovation and evaluation are required in order to identify the optimal design.

7. Acknowledgments

Over one dozen colleagues helped pre-test early versions of the survey. Norman Sadeh and George Duncan provided valuable input concerning the study's design and analysis, respectively. Mary Shaw provided advice on the preparation of this report. This work has been funded in part by the EUSES Consortium via the National Science Foundation (ITR-0325273) and by the National Science Foundation under Grant CCF-0438929. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the sponsors.

8. References

[1] Barzilay, R., and Elhadad, M. Using Lexical Chains for Text Summarization. *ACL'97/EACL'97: Proceedings of the Workshop on Intelligent Scalable Text Summarization*, 1997.

[2] Boguraev, B., and Kennedy, C. Salience-based Content Characterization of Text Documents. *ACL'97/EACL'97: Proceedings of the Workshop on Intelligent Scalable Text Summarization*, 1997.

[3] Church, K., and Hanks, P. Word Association Norms, Mutual Information, and Lexicography. *Computational Linguistics*, Vol. 16, No. 1, 1990, 22-29.

[4] Clark, D. Shopbots Become Agents for Business Change. *Computer*, Vol. 33, No. 2, Feb. 2000, 18-21.

[5] Damerau, F. An Experiment in Automatic Indexing. *American Documentation*, Vol. 16, No. 4, Oct 1965, 283-289.

[6] Dave, K., Lawrence, S., and Pennock, D. Mining the Peanut Gallery: Opinion Extraction and Semantic Classification of Product Reviews. *WWW '03: Proceedings of the 12th International Conference on World Wide Web*, ACM Press, 2003, 519-528.

[7] Edmundson, H., and Wyllys, R. Automatic Abstracting and Indexing—Survey and Recommendations. *Communications of ACM*, Vol. 4, No. 5, 1961, 226-234.

[8] Gelbukh, A., and Sidorov, G. Zipf and Heaps Laws' Coefficients Depend on Language. *CICLing '01: Proceedings of the Second International Conference on Computational Linguistics and Intelligent Text Processing*, Springer-Verlag, 2001, 332-335.

[9] Glance, N., Hurst, M., and Tomokiyo, T. BlogPulse: Automated Trend Discovery for Weblogs. *WWW 2004 Workshop on the Weblogging Ecosystem: Aggregation, Analysis and Dynamics*, 2004.

[10] Harter, S. *A Probabilistic Approach to Automatic Keyword Indexing. Part I: On the Distribution of Specialty Words in a Technical Literature*. Journal of the ASIS, Vol. 26, 1975, 197-206.

[11] Hovy, E., and Lin, C. Automated Text Summarization in SUMMARIST. *Advances in Automated Text Summarization* (I. Mani and M. Maybury, eds.), MIT Press, 1999.

[12] Hu, M., and Liu, B. Mining and Summarizing Customer Reviews. *KDD '04: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM Press, 2004, 168-177.

[13] Hu, M., and Liu, B. Mining Opinion Features in Customer Reviews. *AAAI'04: Proceedings of the 9th National Conference on Artificial Intelligence*, 2004, 755-760.

[14] Jones, G., Foote, J., Jones, K., and Young, S. Retrieving Spoken Documents by Combining Multiple Index Sources. *SIGIR '96: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM Press, 1996, 30-38.

[15] Kageura, K., and Umino, B. Methods of Automatic Term Recognition: A Review. *Terminology*, Vol. 3, No. 2, 1996, 259-289.

[16] Kan, K., and McKeown, K. Information Extraction and Summarization: Domain Independence through Focus Types. Technical Report CUCS-030-99, Columbia University, 1999.

[17] Leech, G., Rayson, P., and Wilson, A. *Word Frequencies in Written and Spoken English: Based on the British National Corpus*, Longman, London, United Kingdom, 2001. Baseline lemma-frequency tables are available for download at http://www.comp.lancs.ac.uk/ucrel/bncfreq/

[18] Liu, B., Hu, M., and Cheng, J. Opinion Observer: Analyzing and Comparing Opinions on the Web. *WWW '05: Proceedings of the 14th International Conference on World Wide Web*, ACM Press, 2005, 342-351.

[19] Liu, H. *MontyLingua: An End-To-End Natural Language Processor with Common Sense*, Available at: http://web.media.mit.edu/~hugo/montylingua.

[20] Mani, I., and Bloedorn, E. Multi-Document Summarization By Graph Search and Matching. Proceedings of the 14th National Conference on Artificial Intelligence and the 9th Annual Conference on Innovative Applications of Artificial Intelligence, AAAI Press, 1997, 622-628.

[21] Matsuo, Y., and Ishizuka, M. Keyword Extraction from a Single Document Using Word Co-Occurrence Statistical Information. *FLAIRS'03: Proceedings of the 16th International Florida AI Research Society*, 2003, 392-396.

[22] Morinaga, S., Yamanishi, K., Tateishi, K., and Fukushima, T. Mining Product Reputations on the Web. *KDD '02: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM Press, 2002, 341-349.

[23] Nakagawa, H., and Mori, T. A Simple but Powerful Automatic Term Extraction Method. *COLING'02: The 19th International Conference on Computational Linguistics*, Morgan Kaufmann Press, 2002, 29-35.

[24] Nigam, K. and Hurst, M. Towards a Robust Metric of Opinion. AAAI-EAAT'04: AAAI Spring Symposium on Exploring Attitude and Affect in Text, 2004.

[25] Noreault, T., McGill, M., and Koll, M. A Performance Evaluation of Similarity Measures, Document Term Weighting Schemes and Representations in a Boolean Environment. *SIGIR '80: Proceedings of the 3rd Annual ACM Conference on Research and Development in Information Retrieval*, Butterworth & Co., 1981, 57-76.

[26] Radev, D., and McKeown, K. Generating Natural Language Summaries from Multiple On-Line Sources. *Computational Linguistics*, Vol. 24, No. 3, 1998, 470-500.

[27] Robertson, S., van Rijsbergen, C., and Porter, M. Probabilistic Models of Indexing and Searching. *SIGIR '80: Proceedings of the 3rd Annual ACM Conference on Research and Development in Information Retrieval*, Butterworth & Co., 1981, 35-56.

[28] Salton, G., Wong, A., and Yang, A. A Vector Space Model for Automatic Indexing. *Communications of ACM*, Vol. 18, No. 11, 1975, 613-620.

[29] Sista, S., and Srinivasan, S. Polarized Lexicon for Review Classification. MLMTA'04: Proceedings of the International Conference on Machine Learning; Models, Technologies & Applications, CSREA Press, 2004.

[30] Smith, M. The Impact of Shopbots on Electronic Markets. Journal of the Academy of Marketing Science, Vol. 30, No. 4, Fall 2002, 442-450.

[31] Stone, D., and Rubinoff, M. Statistical Generation of a Technical Vocabulary. *American Documentation*, Vol. 19, No. 4, 1968, 411–412.

[32] Turney, P. Thumbs Up Or Thumbs Down?: Semantic Orientation Applied To Unsupervised Classification of Reviews. *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, Association for Computational Linguistics, 2001, 417-424.

[33] Wu, H., and Salton, G. A Comparison of Search Term Weighting: Term Relevance vs. Inverse Document Frequency. *SIGIR '81: Proceedings of the 4th Annual International ACM SIGIR Conference on Information Storage and Retrieval*, ACM Press, 1981, 30-39.