Query-Free News Search

Monika Henzinger
Google Inc.
2400 Bayshore Parkway
Mountain View, CA 94043
USA
monika@google.com

Bay-Wei Chang
Google Inc.
2400 Bayshore Parkway
Mountain View, CA 94043
USA
bay@google.com

Brian Milch
UC Berkeley
Computer Science Division
Berkeley, CA 94720-1776
USA
milch@cs.berkeley.edu

Sergey Brin
Google Inc.
2400 Bayshore Parkway
Mountain View, CA 94043
USA
sergey@google.com

ABSTRACT

Many daily activities present information in the form of a stream of text, and often people can benefit from additional information on the topic discussed. TV broadcast news can be treated as one such stream of text; in this paper we discuss finding news articles on the web that are relevant to news currently being broadcast.

We evaluated a variety of algorithms for this problem, looking at the impact of inverse document frequency, stemming, compounds, history, and query length on the relevance and coverage of news articles returned in real time during a broadcast. We also evaluated several postprocessing techniques for improving the precision, including reranking using additional terms, reranking by document similarity, and filtering on document similarity. For the best algorithm, 84%-91% of the articles found were relevant, with at least 64% of the articles being on the exact topic of the broadcast. In addition, a relevant article was found for at least 70% of the topics.

1. INTRODUCTION

Many daily activities present information using a written or spoken stream of words: television, radio, telephone calls, meetings, face-to-face conversations with others. Often people can benefit from additional information about the topics that are being discussed. Supplementing television broadcasts is particularly attractive because of the passive nature of TV watching. Interaction is severely constrained, usually limited to just changing the channel; there is no way to more finely direct what kind of information will be presented.

Indeed, several companies have explored suggesting web pages to viewers as they watch TV:

- The Intercast system, developed by Intel, allows entire HTML pages to be broadcast in unused portions of the TV signal. A user watching TV on a computer with a compatible TV tuner card can then view these pages, even without an Internet connection. NBC transmitted pages via Intercast during their coverage of the 1996 Summer Olympics.

- The Interactive TV Links system, developed by VITAC (a closed captioning company) and WebTV (now a division of Microsoft), broadcasts URLs in an alternative data channel interleaved with closed caption data [17, 2]. When a WebTV box detects one of these URLs, it displays an icon on the screen; if the user chooses to view the page, the WebTV box fetches it over the Internet.

For both of these systems the producer of a program (or commercial) chooses relevant documents by hand. In fact, the producer often creates new documents specifically to be accessed by TV viewers. To our knowledge, there has been no previous work on *automatically* selecting web pages that a user might want to see while watching a TV program.

In this paper we study the problem of finding news articles on the web relevant to the ongoing stream of TV *broadcast news*. We restrict our attention to broadcast news since it is very popular and information-oriented (as supposed to entertainment-oriented).

Our general approach is to extract queries from the ongoing stream of closed captions, issue the queries in real time to a news search engine on the web, and postprocess the top results to determine the news articles that we show to the user. We evaluated a variety of algorithms for this problem, looking at the impact of inverse document frequency, stemming, compounds, history, and query length on the relevance and coverage of news articles returned in real time during a broadcast. We also evaluated several postprocessing techniques for improving the precision, including reranking using additional terms, reranking by document similarity, and filtering on document similarity. The best algorithm achieves a precision of 91% on one data set and 84% on a second data set and finds a relevant article for at least 70% of the topics in the data sets.

In general, we find that it is more important to concentrate on a good postprocessing step than on a good query generation step. The difference in precision between the best and the worst query generation algorithm is at most 10 percentage points, while our best postprocessing step improves precision by 20 percentage points or more. To reduce the impact of postprocessing on the total number of relevant articles retrieved, we simply increased the number of queries.

To be precise, the best algorithm uses a combination of techniques. Our evaluation indicates that the most important features for its success are a "history feature" and a postprocessing step that

WWW2003, May 20–24, 2003, Budapest, Hungary.
ACM xxx.

filters out irrelevant articles. Many of the other features that we added to improve the query generation do not seem to have a clearly beneficial impact on precision. The "history feature" enables the algorithm to consider all terms since the start of the current topic when generating a query. It tries to detect when a topic changes and maintains a data structure that represents all terms in the current topic, weighted by age. The filtering step discards articles that seem too dissimilar to each other or too dissimilar to the current topic. We also experimented with other postprocessing techniques but they had only a slight impact on precision.

Our algorithms are basically trying to extract keywords from a stream of text so that the keywords represent the "current" piece of the text. Using existing terminology this can be called *time-based keyword extraction*. There is a large body of research on topic detection and text summarization. Recently, time-based summarization has also been studied [1], but to the best of our knowledge there is no prior work on time-based keyword extraction.

The remainder of this paper is organized as follows: Section 2 describes the different query generation algorithms and the different postprocessing steps. Section 3 presents the evaluation. Section 4 discusses related work. We conclude in Section 5.

2. OUR APPROACH

Our approach to finding articles that are related to a stream of text is to create queries based on the text and to issue the queries to a search engine. Then we postprocess the answers returned to find the most relevant ones. In our case the text consists of closed captioning of TV news, and we are looking for relevant news articles on the web. Thus we issue the queries to a news search engine.

We first describe the algorithms we use to create queries and then the techniques we use for postprocessing the answers.

2.1 Query Generation

We are interested in showing relevant articles at a regular rate during the news broadcast. As a result the query generation algorithm needs to issue a query periodically, i.e., every s seconds. It cannot wait for the end of a topic. We chose $s = 15$ for two reasons: (1) Empirically we determined that showing an article every 10-15 seconds allows the user to read the title and scan the first paragraph. The actual user interface may allow the user to pause and read the current article more thoroughly. (2) A caption text of 15 seconds corresponds to roughly three sentences or roughly 50 words. This should be enough text to generate a well-specified query.

Because postprocessing may eliminate some of the candidate articles, we return two articles for each query. We also tested at $s = 7$, thus allowing up to half of the candidate articles to be discarded while maintaining the same or better coverage as $s = 15$.

The query generation algorithm is given the *text segment* T since the last query generation. It also keeps information about the previous stream of text. We consider seven different query generation algorithms, described in the following sections. All but the last query generation algorithm issue 2-term queries. A *term* is either a word or a 2-word compound like *New York*. Two-term queries are used because experiments on a test set (different from the evaluation set used in this paper) showed that 1-term queries are too vague and return many irrelevant results. On the other hand, roughly half of the time 3-term queries are too specific and do not return any results (because we are requiring all terms to appear in the search results). The last query generation algorithm uses a combination of 3- and 2-term queries to explore whether the 2-term limit hurts performance.

As is common in the IR literature [18] the *inverse document frequency* idf of a term is a function of the frequency f of the term in the collection and the number N of documents in the collection. Specifically, we use the function $\log(N/(f+1))$. Since we do not have a large amount of closed caption data available, we used Google's web collection to compute the idf of the terms. This means N was over 2 billion, and f was the frequency of a term in this collection. Unfortunately, there is a difference in word use in written web pages and spoken TV broadcasts. As a result we built a small set of words that are common in captions but rare in the web data. Examples of such words are *reporter* and *analyst*. All of the algorithms below ignore the terms on this stopword list.

2.1.1 The baseline algorithm A1-BASE

Our baseline algorithm is a simple $tf \cdot idf$ based algorithm. It weights each term by $tf \cdot idf$, where tf is the frequency of the term in the text segment T. This results in larger weights for terms that appear more frequently in T, and larger weights for more unusual terms. This is useful since doing a search with the more distinctive terms of the news story is more likely to find articles related to the story. The baseline algorithm returns the two terms with largest weight as the query.

2.1.2 The $tf \cdot idf^2$ algorithm A2-IDF2

This is the same algorithm as the baseline algorithm, but a term is weighted by $tf \cdot idf^2$. The motivation is that rare words, like named entities, are particularly important for issuing focussed queries. Thus, the idf component is more important than tf.

2.1.3 The simple stemming algorithm A3-STEM

In the previous two algorithms each term is assigned a weight. Algorithm A3-STEM assigns instead a weight to each *stem*. The *stem* of a word is approximated by taking the first 5 letters of the word. For example, *congress* and *congressional* would share the same stem, *congr*. The intention is to aggregate the weight of terms that describe the same entity. We use this simple method of determining stems instead of a more precise method because our algorithm must be real-time.

For each stem we store all the terms that generated the stem and their weight. The weight of a term is $c \cdot tf \cdot idf^2$, where $c = 1$ if the term was a noun and $c = 0.5$ otherwise. (Nouns are determined using the publicly available Brill tagger [3].) We use this weighting scheme since nouns are often more useful in queries than other parts of speech. The weight of a stem is the sum of the weights of its terms.

To issue a query the algorithm determines the two top-weighted stems and finds the top-weighted term for each of these stems. These two terms form the query.

2.1.4 The stemming algorithm with compounds, algorithm A4-COMP

Algorithm A4-COMP consists of algorithm A3-STEM extended by two-word compounds. Specifically, we build stems not only for one-word terms, but also for two-word compounds. For this we use a list of allowed compounds compiled from Google's corpus of web data. Stems are computed by stemming both words in the compound, i.e., the stem for the compound *veterans administration* is *veter-admin*. Compounds are considered to be terms and are weighted as before. Queries are issued as for algorithm A3-STEM, i.e., it finds the top-weighted term for the two top-weighted stems. Since a term can now consists of a two-word compound, a query can now in fact consist of two, three, or four words.

2.1.5 The history algorithm A5-HIST

Algorithm A5-HIST is algorithm A4-COMP with a "history feature". All previous algorithms generated the query terms solely on the basis of the text segment T that was read since the last query generation. Algorithm A5-HIST uses terms from previous text segments to aid in generating a query for the current text segment, the notion being that the context leading up to the current text may contain terms that are still valuable in generating the query.

It does this by keeping a data structure, called the *stem vector*, which represents the previously seen text, i.e., the history. It combines this information with the information produced by algorithm A4-COMP for the current text segment T and finds the top weighted stems.

To be precise, for each stem the stem vector keeps a weight and a list of terms that generated the stem, each with its individual weight. The stem vector keeps the stems of all words that were seen between the last reset and the current text segment. A *reset* simply sets the stem vector to be the empty vector; it occurs when the topic in a text segment changes substantially from the previous text segment (see below).

When algorithm A5-HIST receives text segment T it builds a second stem vector for it using algorithm A4-COMP. Then it checks how similar T is to the text represented in the old stem vector by computing a similarity score sim. To do this we keep a stem vector for each of the last three text segments. (Each text segment consists of the text between two query generations, i.e., it consists of the text of the last s seconds.) We add these vectors and compute the dot-product of this sum with the vector for T, only considering the weights of the terms and ignoring the weights of the stems. If the similarity score is above a threshold a_1, then T is *similar* to the earlier text. If the similarity score is above a_2 but below a_1, then T is *somewhat similar* to the earlier text. Otherwise T is *dissimilar* from the earlier text.

If text segment T is similar to the earlier text, the old stem vector is *aged* by multiplying every weight by 0.9 and then the two vectors are added. To add the two vectors, both vectors are expanded to have the same stems by suitably adding stems of weight 0. Also the set of terms stored for each stem is expanded to consist of the same set by adding terms of weight 0. Then the two vectors are added by adding the corresponding weights of the stems and of the terms.

If text segment T is very dissimilar from the earlier text, then the old stem vector is reset and is replaced by the new stem vector. To put it another way, when the current text is very different than the previous text, it means that the topic has changed, so previous history should be discarded in deciding what query to issue.

If text segment T is somewhat similar to the earlier text, then the stem vector is not reset, but the weights in the old stem vector are decreased by multiplying them with a weight that decreases with the similarity score sim. Afterwards the old stem vector and the new stem vector are added. So even though the topic has not completely changed, previous terms are given less weight to allow for topic drift.

We used a test data set (different from the evaluation data sets) to choose values for a_1 and a_2 in the sim calculation. In our implementation, $a_1 = 0.001$ and $a_2 = 0.0003$. When T is somewhat similar, we use the weight multiplier $a = 0.9^{2-1000 \cdot sim}$, which was chosen so that $a \leq 0.9$, i.e., the weights are more decreased than in the case that T is similar to the early text.

In the resulting stem vector the top two terms are found in the same way as in algorithm A4-COMP.

2.1.6 The query shortening algorithm A6-3W

To verify our choice of query length 2 we experimented with a query shortening algorithm, which issues a multiple term query, and shortens the query until results are returned from the news search engine. Earlier experiments showed that reducing the query to one term hurt precision. Therefore we kept two terms as the minimum query length. The query shortening algorithm A6-3W is identical to A5-HIST, but begins with three-term queries, reissuing the query with the two top-weighted terms if there are no results.

2.1.7 Algorithm A7-IDF

Algorithm A7-IDF is identical to algorithm A5-HIST with idf^2 replaced by idf.

(Note that each increasing algorithm A1-A6 adds one additional feature to the previous. A7-IDF does not fit this pattern; we created it in order to test the specific contribution of idf^2 to A5-HIST's performance.)

2.2 Postprocessing

After generating the search queries we issue them to a news search engine and retrieve the top at most 15 results. Note that each result contains exactly one news article. Because we want to retrieve articles that are about the current news item, we restricted the search to articles published on the day of the broadcast or the day before.

We applied several ways of improving upon these search results, described in the sections below, and then selected the top two results to show to the user as news articles related to the broadcast news story.

Since several queries will be issued on the same topic, they may yield similar result sets and many identical or near identical articles may end up being shown to the user. In fact, in the data sets used for the evaluation (see 3.1), queried at both $s = 7$ and $s = 15$, an average of 40% of articles returned would be near-duplicates. Such a large number of duplicates would lead to a poor user experience, so we employed a near-duplicate backoff strategy across all the algorithms. If an article is deemed a near-duplicate of one that has already been presented, the next article in the ranking is selected. If all articles in the result set are exhausted in this manner, the first article in the result set is returned (even though it was deemed a near-duplicate). This reduces the number of repeated highly similar articles to an average of 14% in the evaluation data sets.

To detect duplicates without spending time fetching each article, we looked at the titles and summaries of the articles returned by the search engine. We compared these titles and summaries to a cache of article titles and summaries that have already been displayed during the broadcast. A similarity metric of more than 20% word overlap in the title, or more than 30% word overlap in the summary, was successful in identifying exact matches (e.g., the same article returned in the results for a different query) and slight variants of the same article, as is common for news wires to issue as the story develops over time.

The postprocessing steps we used were boosting, similarity reranking, and filtering.

2.2.1 Boosting

The news search engine gets a two-term query and does not know anything else about the stream of text. The idea behind boosting is to use additional high-weighted terms to select from the search results the most relevant articles. To implement this idea the query generation algorithm returns along with the query associated *boost terms* and *boost values*. The boost terms are simply the top five terms found in the same way as the query terms. The boost values are the IDF values of these terms.

The boosting algorithm then reranks the results returned from the search by computing a weight for each result using the boost terms. For a boost term which has IDF idf and occurs tf times in the text summary returned with the result, the weight is incremented by the value $idf \cdot 4tf/(tf + 3)$, which is a $tf \cdot idf$-like formula that limits the influence of the tf part to 4. For boost terms in the title, the weight is increased by twice that value. Finally, to favor more recent articles, the weight is divided by $d + 1$, where d is the number of days since the article was published. Since we restrict articles to the current date and the day before, the weight is divided by either 1 or 2. The results are then reordered according to their weight; non-boosted results or ties are kept in their original order.

2.2.2 Similarity reranking

A second way of reranking is to compute for each of the results returned by the search engine its similarity to the text segment T and to rerank the search results according to the similarity score. To implement this idea we built a $tf \cdot idf$-weighted term vector for both the text segment T and the text of the article and compute the normalized cosine similarity score. (The first 500 characters of the article are used.) This filtering step requires first fetching the articles, which can be time-expensive.

2.2.3 Filtering

The idea behind filtering is to discard articles that are very dissimilar to the caption. Additionally, when the issued query is too vague, then the top two search results often are very dissimilar. (Indeed, all the results returned by vague queries are often very different from one another.) So whenever we find two candidate articles and they are dissimilar, we suspect a vague query and irrelevant results. So we discard each of the articles unless it is itself highly similar to the caption.

We again used the $tf \cdot idf$-weighted term vector for the text segment T and the text of the article and computed the normalized cosine similarity score as in the similarity reranking, above. Whenever the page-T similarity score is below a threshold b the article is discarded (*Rule F1*). If there are two search results we compute their similarity score and discard the articles if the score is below a threshold p (*Rule F2*)– but allowing each article to be retained if its page-T similarity score is above a threshold g (*Rule F3*).

We analyzed a test data set (different from the evaluation data sets) to determine appropriate thresholds. In our implementation, $b = 0.1$, $g = 0.3$, and $p = 0.35$.

3. EVALUATION

To evaluate different algorithms on the same data set the evaluators worked off-line. They were supplied with two browser windows. One browser window contained the article to be evaluated. The article was annotated with an input box so that the score for the article could simply be input into the box. The other browser window contained the part of the closed caption text for which the article was generated. The evaluators were instructed as follows:

You will be reading a transcript of a television news broadcast. What you will be evaluating will be the relevance of articles that we provide periodically during the broadcast. For each displayed article consider whether the article is relevant to at least one of the topics being discussed in the newscast for this article. Use the following scoring system to decide when a article is relevant to a topic:

- *0 - if the article is not on the topic*
- *1 - if the article is about the topic in general, but not the exact story*
- *2 - if the article is about the exact news story that is being discussed*

For example, if the news story is about the results of the presidential election, then a article about a tax bill in congress would score a 0; a article about the candidates' stands on the environment would score a 1; a article about the winner's victory speech would score a 2.

Don't worry if two articles seem very similar, or if you've seen the article previously. Just score them normally. The "current topic" of the newscast can be any topic discussed since the last article was seen. So if the article is relevant to any of those topics, score it as relevant. If the article is not relevant to those recent topics, but is relevant to a previous segment of the transcript, it is considered not relevant; give it a 0.

We count an article as "relevant" (R) if it was given a score of 1 or 2 by the human evaluator. We count it as "very relevant" (R+) if it was given a score of 2.

To compare the algorithms we use *precision*, i.e., the percentage of relevant articles out of all returned articles. *Recall* is usually defined as the percentage of returned relevant articles out of all relevant articles that exist. However, this is very hard to measure on the web, since it is very difficult to determine all articles on a given topic. In addition, our algorithms are not designed to return all relevant documents, but instead a steady stream of relevant documents. Thus, we define the *relative recall* to be the percentage of returned relevant articles out of all relevant articles pooled from *all of the query generation algorithms with all postprocessing variants.* We use relative recall instead of the number of relevant documents to enable comparison over different data sets. Additionally, we measure *topic coverage*, which is the percentage of topics (defined below) that have at least one relevant article.

To understand the relationship of the different algorithms we compute their overlap, both in terms of issued queries and in terms of articles returned. Since filtering is such a powerful technique we study its effectiveness in more detail.

3.1 Data sets

We evaluated all these approaches using the following two data sets:

- **HN:** three 30-minute sessions of CNN Headline News, each taken from a different day , and
- **CNN:** one hour of Wolf Blitzer Reports on CNN from one day and 30 mins from another day.

The Headline News sessions ("HN") consists of many, relatively short, news stories. The Wolf Blitzer Reports ("CNN") consists of fewer news stories discussed for longer and in greater depth.

Both data sets contain *news stories* and *meta-text*. Meta-text consists of the text between news stories, like "and now to you Tom" or "thank you very much for this report". For evaluating the performance of our algorithms we manually decomposed the news stories into *topics*, ignoring all the meta-text. (This manual segmentation is not an input to the algorithms; it was used strictly for evaluation purposes.) Each topic consists of at least 3 sentences on the same theme; we do not count 1-2 sentence long "teasers" for upcoming stories as topics. The shortest topic in our data sets is 10 seconds long, the longest is 426 seconds long. The average length of a topic in the HN data set is 51 seconds and the median is 27 seconds. The topics comprise a total of 4181 seconds (70 mins) out of the 90 mins long caption. In the CNN data set the average topic length is 107 seconds and the median is 49 seconds. The topics comprise a total of 3854 seconds (64 mins).

Table 1: HN data set: Precision p and relative recall r.

Technique	s	Postprocessing			
		None		Boost+ Filter	
		p	r	p	r
A1-BASE	7	58%	37%	86%	31%
A2-IDF2	7	58%	37%	87%	31%
A3-STEM	7	64%	32%	88%	29%
A4-COMP	7	64%	32%	88%	28%
A5-HIST	7	64%	36%	**91%**	30%
A6-THREE	7	72%	33%	89%	28%
A7-IDF	7	61%	38%	89%	31%
A1-BASE	15	63%	20%	**91%**	17%
A2-IDF2	15	62%	20%	**91%**	18%
A3-STEM	15	69%	25%	88%	24%
A4-COMP	15	70%	26%	90%	25%
A5-HIST	15	67%	26%	89%	24%
A6-THREE	15	**75%**	24%	**91%**	22%
A7-IDF	15	59%	26%	**91%**	24%

Table 2: CNN data set: Precision p and relative recall r.

Technique	s	Postprocessing			
		None		Boost+ Filter	
		p	r	p	r
A1-BASE	7	43%	27%	77%	21%
A2-IDF2	7	46%	27%	75%	18%
A3-STEM	7	43%	23%	76%	18%
A4-COMP	7	44%	23%	76%	17%
A5-HIST	7	55%	32%	84%	23%
A6-THREE	7	60%	30%	86%	23%
A7-IDF	7	52%	25%	82%	23%
A1-BASE	15	48%	17%	83%	14%
A2-IDF2	15	60%	16%	85%	13%
A3-STEM	15	54%	17%	76%	14%
A4-COMP	15	59%	18%	82%	15%
A5-HIST	15	61%	25%	**88%**	20%
A6-THREE	15	**71%**	23%	83%	21%
A7-IDF	15	56%	25%	82%	21%

3.2 Evaluation of the Query Generation Algorithms

We first evaluated all the baseline algorithms with two different ways of postprocessing, namely no postprocessing and postprocessing by both boosting and filtering. The CNN data set consists of 3854 seconds, and thus an algorithm that issues a query every 15 seconds issues 257 queries. We return the top two articles for each query so that a maximum of 514 relevant articles could be returned for this data set when $s = 15$. For the HN data set the corresponding number is 557.

The pool of all relevant documents found by any of the algorithms for the HN data set is 846, and for the CNN data set is 816. Thus the relative recall for each algorithm is calculated by dividing the number of relevant documents it found by these numbers. Note that for $s = 15$ no algorithm can return more than 557 (for HN) or 514 (for CNN) relevant articles, so in those cases the maximum possible relative recall would be $557/846 = 66\%$ (HN) or $514/816 = 63\%$ (CNN).

The pooled relative recall numbers are appropriate for comparing performance among the different algorithms, but not useful as an absolute measure of an algorithm's recall performance, since no algorithm would be able to achieve 100% relative recall. This is because when a query is issued at a text segment, an algorithm is limited to returning a maximum of two articles. However, pooling usually identifies more than two articles as relevant for a given text segment.

Table 1 presents the precision and relative recall for all the different query generation algorithms for the HN data set. Table 2 shows the corresponding numbers for the CNN data set. It leads to a few observations:

- All algorithms perform statistically significantly[1] better with a p-value of < 0.003 when postprocessed with boosting and filtering than without postprocessing. Depending on the algorithm the postprocessing seems to increase the precision by 20-35 percentage points.

- For both data sets the highest precision numbers are achieved with postprocessing and $s = 15$. However, the largest relative recall is achieved without postprocessing and $s = 7$. This is no surprise: Filtering reduces not only the number of non-relevant articles that are returned, but also the number of relevant ones. The impact of postprocessing on the number of relevant articles that are returned varies greatly between algorithms. The maximum change is 71 articles (A1-BASE with $s = 7$ on HN), and the minimum change is 10 articles (A3-STEM with $s = 7$ on HN). Also, reducing s increases the number of queries issued and thus one expects the number of returned articles to increase, both the relevant ones as well as the non-relevant ones. Thus relative recall increases as well.

- Precision on the CNN data set is lower than precision on the HN data set. This is somewhat surprising as longer topics might be expected to lead to higher precision. The reason is that since we issue more queries on the same topic, we reach further down in the result sets to avoid duplicates and end up returning less appropriate articles.

- Algorithm A5-HIST with $s = 7$ and with postprocessing performs well in both precision and relative recall. For the HN data set, it achieves a precision of 91% with 257 relevant articles returned, for the CNN data set it achieves a precision of 84% with 190 relevant articles returned. This means it returns a relevant article every 16 seconds and every 20 seconds, respectively, on the average. The performance of algorithm A6-3W is very similar to algorithm A5-HIST. None of the other algorithms achieves precision of at least 90% and relative recall of at least 30%. For example, algorithms A1-BASE and A2-IDF2 with $s = 15$ have precision 91% on the HN data set but they return roughly 100 articles fewer that A5-HIST with $s = 7$, which corresponds to a drop of relative recall by 13 percentage points (A1-BASE) and 12 percentage points (A2-IDF2).

  Without postprocessing the difference in precision between A5-HIST and A1-BASE, A2-IDF2, A3-STEM, and A4-COMP is statistically significant on the CNN data set for $s = 7$. For $s = 15$ the difference between A5-HIST and A1-BASE is

[1]To determine statistically significance we used the rank-sum test and the t-test. If a p-value is given, it is the p-value of the rank-sum test, as it is more conservative. If no p-value is given, the p-value of the rank-sum test is less than 0.05.

significant with a p-value of < 0.004.

- Without postprocessing the precision of the baseline algorithm A1-BASE is statistically significantly worse than most of the other algorithms on the CNN data set. Also algorithm A6-3W is statistically significantly better than most of the other algorithms. However, these differences disappear or are no longer statistically significant when filtering and boosting is applied.

We also discuss the contribution of different techniques.

- *idf versus idf^2*: The baseline algorithm A1-BASE and algorithm A2-IDF2 differ only in the use of idf^2 versus idf. For $s = 15$ and no postprocessing, A2-IDF2 gives a statistically significant improvement over A1-BASE on the CNN data set. In all the other cases their performance is very similar.

  Algorithms A5-HIST and A7-IDF also differ only in the use of idf^2 versus idf. Without postprocessing A5-HIST outperforms A7-IDF in precision on both data sets. The difference are statistically significant for $s = 7$ on the CNN data set and for $s = 15$ on the HN data set. With postprocessing their performance is either very similar or the difference is not statistically significant. Altogether, idf^2 seems to work slightly better than idf.

- *Stemming:* Adding stemming to algorithm A2-IDF2 gives algorithm A3-STEM. On the HN data set stemming gives an improvement without postprocessing but with postprocessing and $s = 15$ stemming gives slightly worse performance. On the CNN data set stemming hurts precision. Stemming is often used to improve recall. It does increase relative recall over A3-STEM for $s = 15$, but it has no positive impact on relative recall for $s = 7$. Overall, our experiments are inconclusive with regard to the benefits of stemming.

- *Compounds:* Algorithm A4-COMP consists of algorithm A3-STEM with 2-word compounding added, i.e., we only evaluated compounding for algorithms that use stemming. Their performance is very similar. The precision of A4-COMP is larger than the precision of A3-STEM for $s = 15$ on the CNN data set but it is not statistically significant. However, for $s = 15$ and no postprocessing, A4-COMP gives a statistically significant improvement (p-value < 0.02) over A1-BASE on the CNN data set. Overall, adding compounds does not seem to significantly improve precision.

- *History:* Adding a "history feature" to algorithm A4-COMP gives algorithm A5-HIST. The history gives a small improvement in precision for $s = 7$ on the HN data set, while it seems to slightly hurt for $s = 15$. On the CNN data set, A5-HIST clearly outperforms A4-COMP, both in precision and in relative recall; the difference is statistically significant with p-value < 0.004 for $s = 7$ and no postprocessing.

  This is not surprising. For longer topics (as the CNN data set has) it becomes valuable to have a history feature, especially if queries are issued every 7 seconds. Each text segment may not on its own contain highly relevant text that can be used as a query in finding similar stories. Shorter text segments suffer even more from this problem. The history rectifies this by effectively extending the length of the text segment in a time-aged manner.

  For example, for one of the data sets three shootings were in the news: one in Arizona, one in Oklahoma, and one in Jordan. The algorithms without history sometimes returned non-relevant articles about shootings different than the one being discussed in the broadcast because the current text segment did not mention the location. Algorithm A5-HIST never made this mistake. Altogether, we recommend adding a history feature to a query generation algorithm.

- *Query shortening:* Algorithm A6-3W first issues a three-word query and "backs off" to a two-word query if no results were found. This happens for about 60% of the queries. Without postprocessing, its precision is statistically significantly better than all of the other algorithms with $s = 15$ on the CNN data set and for most of the other algorithms for $s = 7$ and also for the HN data set. With boosting and filtering A6-3W is very similar to algorithm A5-HIST. Relative recall decreases slightly when compared to A5-HIST. The reason is that three-word queries might return only one result where two-word query would return at least two results. Thus, trying out three-word queries is helpful without postprocessing, but with postprocessing it does not lead to an improvement.

Table 3: HN data set: Precision and relative recall in parenthesis.

Technique	s	Postprocessing					
		None	Boost	Filter-	Boost + Filter	Sim. Rerank	Sim. Rerank + Filter
A2-IDF2	7	58% (37%)	58% (37%)	88% (32%)	87% (31%)	60% (38%)	84% (34%)
A4-COMP	7	64% (32%)	66% (33%)	86% (27%)	88% (28%)	68% (34%)	86% (32%)
A5-HIST	7	64% (36%)	64% (36%)	91% (29%)	91% (30%)	64% (36%)	88% (31%)
A2-IDF2	15	62% (20%)	64% (20%)	89% (17%)	91% (18%)	66% (21%)	92% (20%)
A4-COMP	15	70% (26%)	72% (27%)	93% (23%)	90% (25%)	74% (27%)	91% (25%)
A5-HIST	15	67% (26%)	69% (26%)	92% (22%)	89% (24%)	71% (26%)	92% (25%)

Table 9 and Table 10 in the appendix give the percentage of articles exactly on topic (R+: given a score of 2 by the evaluator) together with the actual number of such articles found by each algorithm. They confirm the above observations.

In conclusion, postprocessing and the "history feature" give the largest improvement in search precision, namely 20-35 percentage points for postprocessing and about 5 percentage points for history. Postprocessing reduces relative recall by about 6 percentage points, while the history feature has negligible effect on relative recall. A query generation algorithm should have both, a way to include the history and a postprocessing step that filters out irrelevant documents. None of the other features seem clearly beneficial.

3.3 Postprocessing

As we saw in the previous section postprocessing using boosting and filtering gives a big improvement in precision without decreasing relative recall much. The obvious question is what contributed most to the improvement, boosting or filtering. A second question is whether postprocessing by similarity reranking performs better than postprocessing by boosting.

Since the improvement was unanimous among algorithms and data sets, we evaluated only the HN data set for 3 algorithms. Table 3 shows the details. In all six cases the improvement is clearly achieved by the filtering step, the boosting step only giving a small improvement. All of the differences between boosting alone and filtering and boosting are statistically significant. Also, all of the differences between boosting alone and filtering alone are statistically significant. In some cases filtering alone gives even higher precision than filtering and boosting together.

Similarity reranking seems to give a slightly higher gain in precision than boosting. However, combined with filtering it does not perform better than boosting and filtering combined. None of the differences between boosting alone and similarity reranking alone and between boosting with filtering and similarity reranking with filtering are statistically significant.

Note however, that similarity reranking and filtering together always has better relative recall than boosting and filtering, which in turn has better relative recall than filtering alone.

The results when analyzing the articles with score R+ and the data for the CNN data sets (both omitted in this paper) confirm the above findings.

To summarize, filtering gives a large precision improvement: about 20-30 percentage points with a decrease of 6 percentage points in relative recall. Filtering and similarity reranking together achieve the same precision but return roughly 10% more relevant articles than filtering alone.

3.4 Query Overlap and URL Overlap

Given a postprocessing step the performance of the different query selection algorithms is very similar. An obvious question to ask is whether the reason for this similarity is that the algorithms issue very similar queries. To answer this question we compute the similarity between the queries issued by the different query selection algorithms, i.e., we compare the ith query issued by one algorithm with the ith query issued by another algorithm. Table 4 gives the percentage of queries that have identical terms (though not necessarily ordered identically) for $s = 7$ and the HN data set. Note that we are looking at all generated queries, i.e., the queries *before* the postprocessing step.

The table shows that nearly all queries are identical for related algorithms like A1-BASE and A2-IDF2. However, for algorithms A1-BASE and A5-HIST for example, only 10% of the queries are identical. Table 13 give the corresponding data for $s = 15$. It can be found in the appendix.

Even if the queries are quite different, there could still be a large overlap in the URLs returned at a given point in the stream of text. However, that is also not the case as table 5 shows for the HN data set and $s = 7$. The results for $s = 15$ are similar. Thus it might be possible to improve precision by combining the algorithms in the right way.

To summarize, the overlap both in queries and in articles is high between A1-BASE and A2-IDF2 and is high between A3-STEM and A4-COMP but is low otherwise. Thus, even though the algorithms have similar performance when used with postprocessing, it is in general not due to the same queries being issued or the same URLs being returned.

3.5 Topic Coverage

Another question to ask is how many of the topics receive at least one relevant article. In the HN data set there were a total of 82 topics. In Table 6 we show the percentage of topics with at least one relevant article for the HN data set and also the percentage of topics with at least one article rated R+ for the HN data set. Not surprisingly, these percentages are strongly correlated with relative

Table 4: HN data set with $s = 7$: Percentage of queries that are identical when sorted lexicographically.

	A1-BASE	A2-IDF2	A3-STEM	A4-COMP	A5-HIST	A6-3W	A7-IDF
A1-BASE		94%	27%	25%	10%	6%	10%
A2-IDF2	94%		30%	27%	12%	7%	10%
A3-STEM	27%	30%		87%	31%	19%	28%
A4-COMP	25%	27%	87%		38%	19%	34%
A5-HIST	10%	12%	31%	38%		40%	63%
A6-3W	6%	7%	19%	19%	40%		30%
A7-IDF	10%	10%	28%	34%	63%	30%	

Table 5: HN data set with $s = 7$: Percentage of URLs of algorithm A that are also returned by algorithm B, where the choice of A determines the row and the choice of B determines the column. Since different algorithms return a different number of URLs the table is not symmetric.

	A1-BASE	A2-IDF2	A3-STEM	A4-COMP	A5-HIST	A6-3W	A7-IDF
A1-BASE		93%	36%	33%	15%	11%	13%
A2-IDF2	96%		37%	36%	17%	13%	15%
A3-STEM	41%	41%		83%	36%	23%	21%
A4-COMP	36%	38%	80%		42%	24%	28%
A5-HIST	16%	18%	35%	42%		39%	40%
A6-3W	13%	15%	23%	26%	43%		38%
A7-IDF	15%	17%	22%	30%	43%	38%	

Table 6: HN data set: Percentage of topics with at least one relevant article and percentage of topics with at least one article rated R+.

Technique	s	Score R		Score R+	
		None	Boost Filter	None	Boost Filter
A1-BASE	7	78%	73%	76%	70%
A2-IDF2	7	79%	76%	76%	72%
A3-STEM	7	74%	70%	70%	67%
A4-COMP	7	76%	72%	70%	68%
A5-HIST	7	77%	70%	73%	67%
A6-3W	7	73%	70%	70%	68%
A7-IDF	7	73%	73%	72%	70%
A1-BASE	15	63%	59%	60%	56%
A2-IDF2	15	63%	61%	60%	60%
A3-STEM	15	72%	67%	70%	67%
A4-COMP	15	76%	72%	73%	71%
A5-HIST	15	72%	65%	68%	65%
A6-3W	15	71%	66%	66%	63%
A7-IDF	15	71%	69%	70%	63%

recall. They are the highest for $s = 7$ with no postprocessing and the lowest for $s = 15$ with postprocessing. It is interesting to note that the numbers are not much lower for the percentage of topics with score R+ than for score R. Said differently, if a topic has a relevant article it most likely also has a topic rated R+.

Table 11 and Table 12 in the appendix give the corresponding percentages for the CNN data set. The values are higher as we would expect since the topics are longer. However, there is also more variation in these numbers as there are only 36 topics in the CNN data set.

We also analyzed longer and shorter topics. Both are equally well covered, i.e., the length is not the distinguishing factor of whether a topic is covered or not. Instead there seem to be topics for which it is "hard" to find relevant articles and others for which it easy. For example, it is easy to find articles for Winona Ryder's shoplifting trial: Her name is rare and thus had high *idf*, and she is not mentioned in other news for that day. For other topics it is hard to find related news stories, mostly because they fall into the category of "unusual" news. Examples include a story about a beauty pageant for women in Lithuania's prisons, a story about a new invention that uses recycled water from showers and baths to flush toilets, and a story about garbage trucks giving English lessons over loudspeakers in Singapore.

In summary, roughly 70% of the topics have at least one article rated relevant, and almost as many have at least one article rated very relevant (R+). The length of the topic does not seem to be a factor in determining whether a relevant article can be found for it.

3.6 Filtering Effectiveness

The filtering technique is very powerful in improving precision. Recall that there can be two reasons why an article is filtered out: *F1:* Its similarity with text segment T is below threshold b. *F2:* Its similarity with text segment T is below threshold g *and* there are two search results and their similarity score is below a third threshold. (Recall that $b < g$.) Note that it is possible that both rules apply. We analyzed which of the two rules filters out more articles. Table 7 shows the percentage of articles that each filtering rule filtered on the HN data set. The percentage can add up to over 100% since both rules can apply. It clearly shows that F2 filters out most of the articles.

Table 7: HN data set: For each filtering rule the percentage of filtered articles that are filtered by the technique. The percentages for a given algorithm can add up to over 100% since both filtering rules can apply.

Technique	s	# filtered articles	% filtered by F1	% filtered by F2
A1-BASE	7	218	39%	97%
A2-IDF2	7	202	38%	96%
A3-STEM	7	139	27%	98%
A4-COMP	7	127	30%	98%
A5-HIST	7	175	54%	86%
A6-3W	7	209	48%	86%
A7-IDF	7	130	54%	78%
A1-BASE	15	126	24%	98%
A2-IDF2	15	85	29%	96%
A3-STEM	15	76	24%	93%
A4-COMP	15	76	22%	93%
A5-HIST	15	95	32%	92%
A6-3W	15	130	26%	97%
A7-IDF	15	36	33%	81%

Table 8: HN data set and $s = 7$: The error rate for each filtering rule.

Technique	F1	F2	F3
A1-BASE	12%	6%	7%
A2-IDF2	15%	9%	6%
A3-STEM	22%	9%	3%
A4-COMP	22%	6%	3%
A5-HIST	32%	33%	11%
A6-3W	29%	28%	2%
A7-IDF	23%	25%	3%

Finally, we wanted to evaluate for each filtering rule how often it makes the wrong decision. For F1 and F2 this means that they discard a relevant article. Rule F3 requires that an article is kept if its similarity to the caption text is above a threshold g. It makes the wrong decision if it keeps an irrelevant article. Table 8 gives the error rate for each filtering rule. For F1 and F2, the *error rate* is the percentage of relevant articles out of all articles filtered by the technique. For F3, it is the percentage of irrelevant articles out of all articles whose similarity with text T is above the threshold g. The error rates range from very low to, in one instance, nearly a third. For F3, which excludes highly similar articles from being filtered, the mostly low error rate indicates that few irrelevant articles escape filtering through this technique. The higher error rates for F1 and F2 indicate that relevant pages are being suppressed; but we can tolerate this since we are aggressively querying for two results every 7 or 15 seconds.

4. RELATED WORK

4.1 Query-free search

To our knowledge, there has been no previous work on automatically selecting documents that a user might want to see while watching a TV program. However, there is a significant literature on the broader problem of query-free information retrieval: finding documents that are relevant to a user's current activity, without requiring an explicit query. The different systems differ in what stream of text they consider as input and what genre of related documents they return. We will use the "Input—Output" notation below.

- *Web pages—web pages*: The Letizia system [10] observes a user browsing the web, and suggests other web pages the user may find interesting. Rather than searching an index of web pages, it "surfs ahead" of the user, following hyperlinks from the page the user is currently viewing. Similarly, commercial browser assistants such as Autonomy Kenjin and PurpleYogi (both no longer available) suggested related web pages based on the content of web pages the user has been viewing.

- *Problem report—repair manual*: Another early query-free IR system is FIXIT [8], which helps technicians as they use an expert system to diagnose and repair copiers. FIXIT identifies the currently reported symptoms and the faults it considers likely, then maps these symptoms and faults to keywords, and retrieves sections of the copier documentation that match these words.

- *User behavior—personal files*: The just-in-time IR project at MIT [15, 14] has focused on retrieving personal files – such as notes and archived email messages – that a user would currently find useful. This project first produced the Remembrance Agent, which looks at a document the user is editing in Emacs and matches fragments of this document (such as the last 50 words) against a corpus of personal files. The followup Margin Notes system performs a similar task, but observes the web pages that a user views in a web browser. Finally, the Jimminy system runs on a wearable computer. Jimminy bases its suggestions on what the user is reading or writing on the heads-up display, as well as on Global Positioning System data and active badge data indicating what other people are nearby. All these systems use a common information retrieval backend based on the Okapi similarity metric [16].

  The XLibris pen-based document reader [13] allowed users to mark up documents as they are reading. The system would derive queries from the passages of text that were marked, and search over a local corpus for relevant documents to present to the user.

- *User behavior—News and stock quotes*: The SUITOR system [11] tracks user behavior like what applications are running and what text the user currently writes to build a model of the user's current interest. It uses this model to find information that is interesting to the user like news headlines and stock quotes.

- *Open documents in editor or browser—web pages*: The system most similar in purpose to our own is Watson [5], which suggests web pages to a computer user based on the documents currently open in a word processor or web browser. Watson uses a variety of heuristics to construct queries from the text of the documents, then sends these queries to the AltaVista search engine.

- *Email—web pages*: Our work is also related to a small prototype system that constructed queries from email messages and sent them to an early version of the Google search engine [4].

4.2 Text Summarization and Keyword Extraction

In the Information Retrieval literature there has been a plethora of work on topic detection and text summarization. Recently, the problem of time-based summarization has been studied. See [1] for an excellent overview of the area. Our work is different in two ways:

(1) It doesn't need to identify topics; it only needs to detect whether the current topic is different from the previous topic. If a later topic is very similar to a topic discussed much earlier, the system does not need to recognize this.

(2) The system does not need to construct a summary; it extracts keyphrases that can be used to formulate a search query.

The research on keyphrase extraction, see, e.g., [9, 12, 19, 7], and specifically the algorithm by [20], is the most related to our work. The main difference to our work is that we study the time-based variant of the problem, which also includes topic change detection.

5. CONCLUSION

This paper evaluated seven algorithms and three postprocessing techniques for finding news articles on the web relevant to news broadcasts. For this genre of television show, the best algorithm finds a relevant page every 16-20 seconds on average, achieves a precision of 84-91%, and finds a relevant article for about 70% of the topics. Our experiments clearly show that filtering articles by similarity to the caption text and similarity with each other gives a large improvement in precision. It would be interesting future work to refine and improve upon the filtering technique presented in this paper. It would also be interesting to experiment with different ways of using the history for query generation.

The news search engine we used restricted us to using Boolean retrieval. It is an interesting open question whether a weighted term-vector retrieval would have improved the search quality sufficiently to make postfiltering redundant.

The framework of the system is not limited to news, however; we have considered simple methods of detecting other genres (such as sports, weather, and "general" topics) and sending such queries to appropriate web information sources. The genres could be identified by using machine learning on a labelled corpus of television captions; an even simpler way would be to use television schedules and their associated metadata to categorize the current show into a genre.

Finally, as voice recognition systems improve, the same kind of topic finding and query generation algorithms described in this paper could be applied to conversations, providing relevant information immediately upon demand.

5.1 Acknowledgments

We would like to thank Shahid Choudhry for providing us with closed caption transcripts for our experiments.

6. REFERENCES

[1] J. Allan, R. Gupta, and V. Khandelwal. Temporal summaries of news topics. In *Research and Development in Information Retrieval*, pages 10–18, 2001.

[2] Electronic Industries Alliance. EIA-746-A: Transport of internet uniform resource locator (url) information using text-2 (t-2) service. Technical report, 1998.

[3] E. Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computation Linguistics*, 21(4):543-565, 1995.

[4] S. Brin, R. Motwani, L. Page, and T. Winograd. What can you do with a web in your pocket? *Data Engineering Bulletin*, 21(2):37–47, 1998.

[5] J. Budzik, K. Hammond, and L. Birnbaum. Information access in context. *Knowledge based systems*, 14(1-2):37–53, 2001.

[6] J. Davis. Intercast dying of neglext. CNET News, January 29, 1997.

[7] E. Frank, G. W. Paynter, I. H. Witten, C. Gutwin, and C. G. Nevill-Manning. Domain-specific keyphrase extraction. In *IJCAI*, pages 668–673, 1999.

[8] P. Hart and J. Graham. Query-free information retrieval. *IEEE Expert*, 12(5):32–37, 1997.

[9] B. Krulwich and C. Burkey. Learning user information interests through the extraction of semantically significant phrases. In *AAAI 1996 Spring Symposium on Machine Learning in Information Access*, 1996.

[10] H. Lieberman. Letizia: An agent that assists web browsing. In C. S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 924–929, Montreal, Quebec, Canada, 1995. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[11] P. Maglio, R. Barrett, C. Campbell, and T. Selker. Suitor: An attentive information system. In *International Conference on Intelligent User Interfaces*, 2000.

[12] A. Munoz. Compound key word generation from document databases using a hierarchical clustering art model. *Intelligent Data Analysis*, 1(1), 1997.

[13] M. N. Price, G. Golovchinsky, and B. N. Schilit. Linking by inking: Trailblazing in a paper-like hypertext. In *Hypertext '98*, pages 30–39, 1998.

[14] B. Rhodes and P. Maes. Just-in-time information retrieval agents. *IBM Systems Journal*, 39(3-4), 2000.

[15] B. J. Rhodes. *Just-In-Time Information Retrieval*. PhD thesis, MIT Media Laboratory, Cambridge, MA, May 2000.

[16] S. Robertson, S. Walker, and M. Beaulieu. Okapi at TREC-7: automatic ad hoc, filtering, VLC and interactive track. In *Proceedings of the 7th International Text Retrieval Conference (TREC)*, pages 253–264, 1999.

[17] G. D. Robson. Closed captions, V-chip, and other VBI data. *Nuts and Volts*, 2000.

[18] G. Salton. *The SMART System – Experiments in Automatic Document Processing.* Prentice Hall, 1971.

[19] A. M. Steier and R. K. Belew. Exporting phrases: A statistical analysis of topical language. In *Second Symposium on Document Analysis and Information Retrieval*, pages 179–190, 1993.

[20] P. D. Turney. Learning algorithms for keyphrase extraction. *Information Retrieval*, 2(4):303–336, 2000.

APPENDIX

A. MORE EVALUATION DATA

Table 9: HN data set: Percentage p of articles with score R+ out of all returned articles and percentage r of articles with score R+ out of all articles with score R+.

Technique	s	Postprocessing			
		None		Boost+ Filter	
		p	r	p	r
A1-BASE	7	44%	28%	69%	25%
A2-IDF2	7	45%	29%	70%	25%
A3-STEM	7	49%	25%	73%	24 %
A4-COMP	7	50%	25%	72%	23 %
A5-HIST	7	47%	26%	76%	25 %
A6-THREE	7	56%	26%	75%	23 %
A7-IDF	7	46%	29%	74%	26 %
A1-BASE	15	53%	16%	81%	15%
A2-IDF2	15	51%	16%	78%	15%
A3-STEM	15	54%	20%	75%	20%
A4-COMP	15	51%	19%	72%	20%
A5-HIST	15	52%	20%	71%	19%
A6-THREE	15	59%	19%	75%	18%
A7-IDF	15	46%	20%	75%	20%

Table 10: CNN data set: Percentage p of articles with score R+ out of all returned articles and percentage r of articles with score R+ out of all articles with score R+.

Technique	s	Postprocessing			
		None		Boost+ Filter	
		p	r	p	r
A1-BASE	7	30%	19%	61%	16%
A2-IDF2	7	31%	18%	59%	14%
A3-STEM	7	31%	16%	59%	14%
A4-COMP	7	31%	16%	59%	13%
A5-HIST	7	36%	21%	64%	18%
A6-THREE	7	40%	20%	61%	17%
A7-IDF	7	37%	18%	65%	18%
A1-BASE	15	35%	12%	66%	11%
A2-IDF2	15	43%	12%	67%	10%
A3-STEM	15	37%	12%	51%	9%
A4-COMP	15	39%	12%	58%	10%
A5-HIST	15	40%	16%	60%	14%
A6-THREE	15	49%	16%	59%	15%
A7-IDF	15	36%	16%	56%	14%

Table 11: CNN data set: Percentage of topics with at least one relevant document.

Technique	s	None	Boost Filter
A1-BASE	7	86%	81%
A2-IDF2	7	83%	81%
A3-STEM	7	83%	72%
A4-COMP	7	83%	75%
A5-HIST	7	89%	72%
A7-IDF	7	92%	69%
A1-BASE	15	81%	78%
A2-IDF2	15	75%	72%
A3-STEM	15	69%	64%
A4-COMP	15	72%	67%
A5-HIST	15	78%	75%
A7-IDF	15	78%	75%

Table 12: CNN data set: Percentage of topics with at least one document rated R+.

Technique	s	None	Boost Filter
A1-BASE	7	83%	81%
A2-IDF2	7	81%	75%
A3-STEM	7	72%	69%
A4-COMP	7	78%	69%
A5-HIST	7	81%	72%
A7-IDF	7	78%	67%
A1-BASE	15	72%	72%
A2-IDF2	15	67%	61%
A3-STEM	15	64%	61%
A4-COMP	15	64%	64%
A5-HIST	15	69%	67%
A7-IDF	15	64%	69%

Table 13: HN data set with $s = 15$: Percentage of queries that are identical when sorted lexicographically.

	A1-BASE	A2-IDF2	A3-STEM	A4-COMP	A5-HIST	A6-3W	A7-IDF
A1-BASE		75%	34%	27%	11%	6%	14%
A2-IDF2	75%		40%	32%	13%	9%	11%
A3-STEM	34%	40%		82%	33%	21%	25%
A4-COMP	27%	32%	82%		45%	21%	32%
A5-HIST	11%	13%	33%	45%		38%	57%
A6-3W	6%	9%	21%	21%	38%		25%
A7-IDF	14%	11%	25%	32%	57%	25%	