CSCI 4152—Natural Language Processing

Project 2000

A comparison of Four Keyphrase extraction Algorithms

Atreya Basu

April 14, 2000

Abstract

Keyphrase extraction is becoming an important topic in computing. Keyphrases are not only useful in searching for documents but in also for cataloguing documents and they are important for Machine Learning tasks. Instead of generating Keyphrases by hand it would be ideal to automate this process. There are numerous methods of keyword and keyphrase extraction, and most perform well. In this report I examine the performance of four Keyphrase extraction algorithms; KEA, Microsoft Word 2000 AutoSummarize, NRC Extractor and the Lazy Heading Search. The performance of the algorithms is measured using the 'recall', 'precision', 'F-measure' and 'fallout'. The training data for the experiment was the last 8 chapters of the book, 'Boilers and Burners—Design and theory' (Basu et. al. 1999b). The test data was the first 10 chapters of the same book. Only the KEA and NRC Extractor algorithms were learning algorithms and needed training. The experiment showed that on the test data the non-learning algorithms, specifically Microsoft Word 2000's AutoSummarize and the Lazy Heading Search, gave the overall best extractions.

Table of Contents

Introduction

A person searching for a document in a library often does not know specific details about their target. That is, a person could be searching for documents related to ‘Gardening’ may not know the exact title of the document, its author, or the subject/category it is located or filed under. What they do know is that the body of the document contains information about Gardening. This is the case for most of our document searches; we know the content we are searching for but nothing else. It is unrealistic for searcher to know anything but the content they are looking for—However it is also impractical for cataloguing systems to search the body of a document to see if its content matches the subject or contents in question.

Document keywords are a solution to the above dilemma. Cataloguing systems simply need to keep a list of keywords for every document; where the keywords imply the content of the document.

The problem with keywords is two fold. Keywords are arduous to encode by hand, and a bag of keywords do not always give precise matching. Take our previous query as an example. I wish to find documents relating to Gardening—As in home gardening. A search of keywords in a catalogue produces many documents including some with the following titles, ‘Ancient Babylonian Gardens’, ‘The rise and fall of Gardenia, a corporate story’, ‘Home Gardening, a how-to book’, and ‘Louis Garden, comic genius or bad social taste’. Although fictions this example illustrates a likely scenario. Only one of the returned documents is of use to us, ‘Home Gardening, a how-to book’. Of course the search could have been more precise if we added the search-word, ‘home’ to ‘gardening’. However to narrow the search scope the searcher must have some knowledge about the documents they are searching for; more precisely the searcher must know which keywords could occur in ‘other’ documents.

Our problem, therefore, is two fold; Automatic generation of keywords and a precise way of searching for documents.

An obvious solution to the problem of search results outside your domain of interest is to match more keywords. For better results we could match a series of keywords called a keyphrase. A keyphrase has the advantage of containing not just keywords but also word order. A keyphrase is simply a phrase such as, ‘Home networking’, or ‘Efficiency of coal burning boilers’.

The problem of automatically generating keywords is now the problem of automatically generating keyphrases.

Keyphrase extraction is becoming an important topic in computing. Usually an author provides a list of keyphrases for their document. A keyphrase captures the main topics discussed in a document. Although not an overly laborious task, it would be useful to have an automated process. This has been one of the promises of computers—Message understanding. While other promises, such as speech understanding or even speech reorganization has yet to bear true fruitarian keyphrase extraction appears to be very promising.

Keyphrases are not only useful in searching for documents but in also for cataloguing documents and message understanding in Machine Learning tasks.

Instead of generating Keyphrases by hand it would be ideal to automate this process. There are numerous methods of keyword and keyphrase extraction, and most perform well. In this report I examine the performance of four Keyphrase extraction algorithms; KEA, Microsoft Word 2000 AutoSummarize, NRC Extractor and the Lazy Heading Search.

The performance of the algorithms is measured using the 'recall', 'precision', 'F-measure', and 'fallout.' The training and test data for the experiment was the book, 'Boilers and Burners—Design and theory' (Basu et. al. 1999b). Each algorithm was tested with one chapter at a time against the hand created keyphrases. The KEA and NRC Extractor algorithms were learning algorithms and could perform better after being trained. To accommodate these algorithms the last eight chapters were used as training data. This provided sufficiently large training copra while leaving enough material to give statistically valid test data. The last eight chapters were used because they were narrower in scope leading to better extraction. The testing data was the first ten chapters—as these chapters were more general, they should be more difficult extraction.

Applications of keyphrases

It is important to know the motivation behind the research. For that reason this section looks at some applications of Keyphrase extraction[1].

Keyphrases for Metatdata

The growth of the Internet and corporate intranets has created a document management problem. Many researchers believe that Metadata is essential to address this problem. Metadata is meta-information about a document or set of documents. There are several standards for document metadata, including the Dublin Core Metadata Element Set (championed by the US Online Computer Library Center), the MARC (Machine-Readable Cataloging) format (maintained by the US Library of Congress), the GILS (Government Information Locator Service) standard (from the US Office of Social and Economic Data Analysis), and the CSDGM (Content Standards for Digital Geospatial Metadata) standard (from the US Federal Geographic Data Committee). All of these standards include a field for keyphrases (although they have different names for this field).

Keyphrases for Highlighting

Keyphrases can be used to highlight interesting passages in a document automatically. This helps human readers skim the document.

Keyphrases for Indexing

An alphabetical list of keyphrases, taken from a collection of documents or from parts of a single long document, can serve as an index.

[1] Turney 1999.

Keyphrases for Interactive Query Refinement

Using a search engine is often an iterative process. The user enters a query, examines the resulting hit list, modifies the query, then tries again. Most search engines do not have any special features that support the iterative aspect of searching however.

Keyphrases for Web Log Analysis

Web site managers often want to know what visitors to their site are seeking. Most web servers have log files that record information about visitors, including the internet address of the client machine, the file that was requested by the client, and the date and time of the request. There are several commercial products that analyze these logs for web site managers. Typically these tools will give a summary of the general traffic patterns and produce an ordered list of the most popular files on the web site.

Criteria

The algorithms in this project are evaluated on four criteria, precision, recall, fallout and F-measure (Rijsbergen. 1979:174).

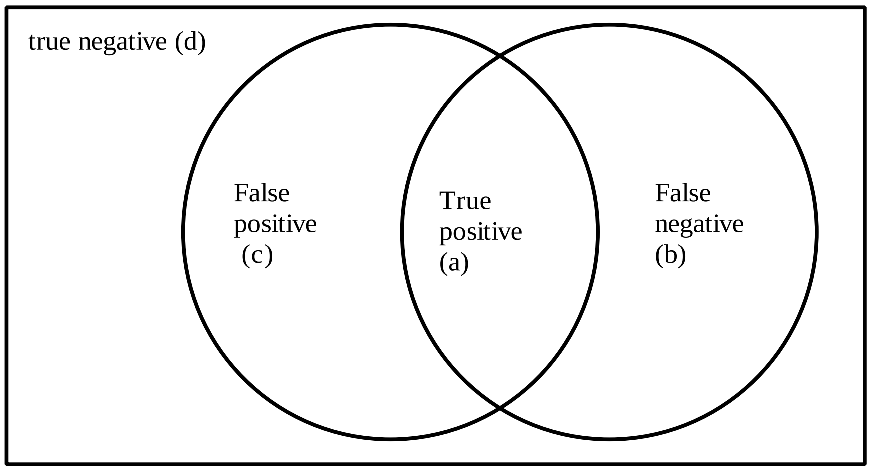


Table 1: The confusion matrix for keyphrase classification.

	Classification as a Keyphrase by the Human	Classified as Not a Keyphrase by the Human
Classified as a Keyphrase by the Machine	a	b
Classified as Not a Keyphrase by the Machine	c	d

$$\text{precision} = \frac{a}{a ? b} \qquad (1)$$

Precision is an estimate of the probability that, a keyphrase (as chosen by the computer) will be an actual keyphrase (as chosen by a human). I can also be thought of as the proportion of the selected items that the system got right.

$$\text{recall} = \frac{a}{a ? c} \qquad (2)$$

Recall is an estimate of the probability that, a keyphrase for a document (as chosen by the human) will be classified as such by the set of keyphrases chosen by the computer. It can also be thought of as the proportion of the target items that the system selected.

$$\text{fallout} = \frac{b}{b \mathbin{?} d} \tag{3}$$

Fallout is the measure of falsely identified keyphrases. In our case fallout is an important measure because the extraction must produce keyphrases that represent the document. Getting a wrong keyphrase might have dire consequences for an application. In the case of classification, incorrect keyphrases would lead to improper classification of a document.

$$\text{F-measure} = \frac{2 * \text{precision} * \text{recall}}{\text{precision} \mathbin{?} \text{recall}} \mathbin{?} \frac{2a}{2a \mathbin{?} b \mathbin{?} c} \tag{4}$$

The F-measure can be thought of as the combination of precision and recall into a single measure of overall performance. The F-measure seems to give preference to true positive results, and in our case this is exactly what we are interested in.

The use of these measures is justifiable criteria because of the following reasons (Manning, Schutze, 1999)

?? Accuracy figures are not very sensitive to the small, but interesting numbers a, b, and c, where as precision and recall are. You can get high accuracy simply by selecting nothing.

?? Other things being equal, the F measure prefers results with more true positives, whereas accuracy is sensitive only to the number of errors. This bias normally reflects our intuitions: We are interested in finding things, even at the cost of also returning some junk.

?? Using precision and recall, one can give a different cost to missing target items versus selecting junk.

These four criteria are used to rate the performance of the four algorithms.

Corpora

The corpora for the project is, ‘Boilers and Burners—Design and theory’ by Basu, Jestin and Kefa. The book is composed of eighteen chapters. The book is a textbook on fossil burning boilers and their burners. This book was chosen for testing because the one of authors (Basu) could build the keyphrases for each chapter.

The first ten chapters of the book were used to test each algorithm. This is because the beginning chapters are more general than the later chapters. A general chapter should be more difficult to extract keyphrases from, as there is a greater number of possible keyphrases for the chapter. The last eight chapters were used as training data for the learning algorithms, KEA and NRC’s Extractor. The last eight chapters were on specific topics, and therefore should be relatively easy to extract keyphrases from. That is the reason they were used for training.

Both training and testing consisted of extracting one chapter at a time. The extracted keyphrases for a chapter was compared against what the author gave as the keyphrase for the same chapter.

Algorithms

KEA

The KEA algorithm, named after one of New Zealand's native parrots, is a learning algorithm. KEA automatically extracts keyphrases form the full text of documents. The set of all candidate-parses in a document are identified using rudimentary lexical processing. Features are computed for each candidate, and machine learning is used to generate a classifier that determines which candidate should be assigned as keyphrases. Two features are used in the standard algorithm[i]; TF.IDF and position of the first occurrence. The TF.IDF require a corpus of text from which document frequencies can be calculated; the machine-learning phrase requires a set of training documents with keyphrases assigned.

The tables below show the titles and keyphrases for three computer science technical reports. Keyphrases extracted by Kea are listed, along with those assigned by the author. Phrases that both the Author and Kea chose are in italics. Generally, the author phrases look a lot better. Kea occasionally chose simple phases like *cut* and *gauge* that are not really appropriate. Kea assigns the keyphrase *garbage* to the third paper, a classification the author is unlikely to agree with.

Protocols for secure, atomic transaction execution in electronic commerce		Neural multigrid for gauge theories and other disordered systems		Proof nets, garbage, and computations	
Author	Kea	Author	Kea	Author	Kea
anonymity	*atomicity*	disordered	disordered	*cut-*	cut

atomicity *auction* *electronic commerce* privacy real-time *security* *transaction*	*auction* customer *electronic commerce* intruder merchant protocol *security* third party *transaction*	systems *gauge fields* *multigrid* neural multigrid neural networks	gauge *gauge fields* interpolation kernels length scale *multigrid* smooth	*elimination* linear logic *proof nets* sharing graphs typed lambda-calculus	*cut elimination* garbage *proof net* weakening

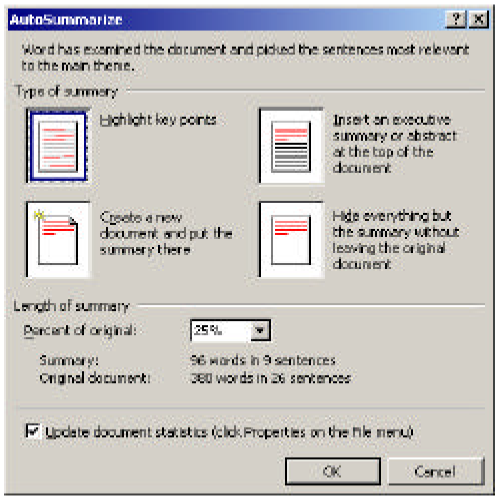
KEA was available for download from the author's site and installed on *Borg*. The program consists of a Perl program that performs some pre-parsing, a C implementation of the Lovins stemmer and finally a Java implementation of the algorithm. Some slight modifications were needed to the Perl code and the Lovins stemmer had to be compiled for Solaris.

To run the program I first had to covert all of the chapters to text files, as the authors suggest gives the best performance. KEA is capable of batch processing, so for training, I simply provided, on the command-line, the names of the chapter files and KEA produced the keywords. Similarly, for testing I provided the chapter filenames on the command-line and KEA produced the keyphrases in a separate file for each chapter.

Microsoft Word 2000 AutoSummarize

Very little can be said about MS Word's AutoSummarize function. There is no literature discussing the algorithm Microsoft uses for keyphrase extraction.

Word's AutoSummarize feature was the most convenient of all the algorithms to use. This is because the corpus was written in Word. To perform keyphrase extraction I had to select Tools ✍ AutoSummarize and a dialog with extraction options was presented.



AutoSummarize

Word has examined the document and picked the sentences most relevant to the main theme.

Type of summary

[✓ Highlight key points]

Insert an executive summary or abstract at the top of the document

Create a new document and put the summary there

Hide everything but the summary without leaving the original document

Length of summary

Percent of original: 25%

Summary: 96 words in 9 sentences

Original document: 380 words in 26 sentences

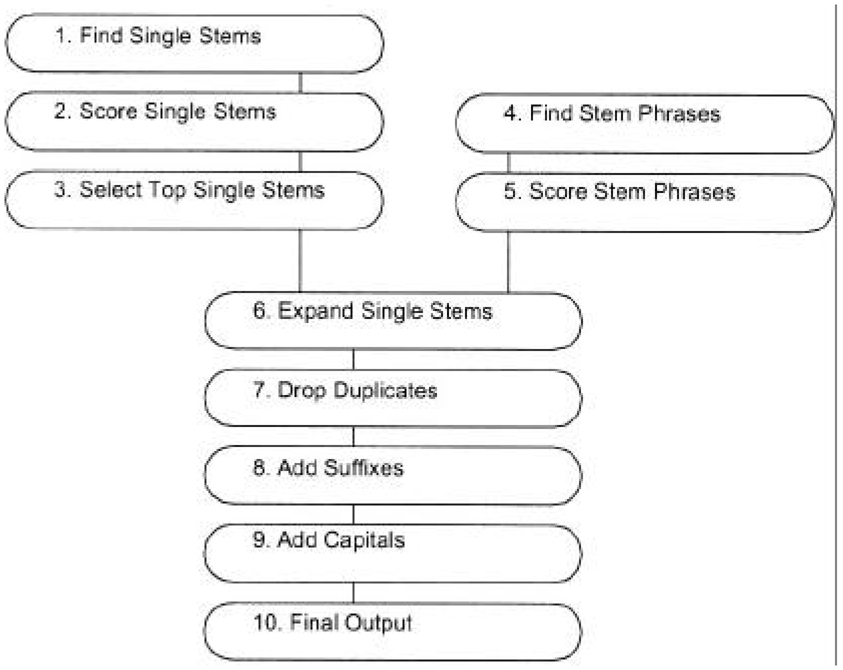
☑ Update document statistics (click Properties on the file menu)

I chose to display the extracted keyphrases on a separate document. Word also has the option of highlighting keyphrases in the document.

It should be noted that although Word's AutoSummarize feature was the easiest to use it was the most inconsistent. Word would extract table rows for example. All other algorithms ignored tables. This is especially troubling since the original format of the corpus was native to Word.

NRC's Extractor

The National Research Council of Canada's Extractor algorithm was available on their web site. Extractor works in the following manner:



1. **Find Single Stems**: Make a list of all the words in the input text. Drop words with less than three characters. Drop stop words. Convert all remaining words to lower case.
2. **Score Single Stems**: For each unique stem, count how often the stem appears in the text and note when it first appears. Assign a score to each stem. The score is the number of times the stem appears in the text, multiplied by a factor. If the stem first appears before FIRST_LOW_THRESH, then multiply the frequency by FIRST_LOW_FACTOR. If the stem first appears after FIRST_HIGH_THRESH, then multiply the frequency by FIRST_HIGH_FACTOR.
3. **Select Top Single Stems:** Rank the stems in order of decreasing score and make a list of the top NUM_WORKING single stems.
4. **Find Stem Phrases**: Make a list of all phrases in the input text. Stem each phrase by truncating each word in the phrase at STEM_LENGTH characters.
5. **Score Stem Phrases**: For each stem phrase, count how often the stem phrase appears in the text and note when it first appears. Assign a score to each phrase, exactly as in step 2.
6. **Expand Single Stems**: For each stem in the list of the top NUM_WORKING single stems, find the highest scoring stem phrase of one, tow or three stems that contain the given single stem. The result is a list of NUM_WORKING stem phrases. Keep this list ordered by the score calculate din step 2.
7. **Drop Duplicates**: The list of the top NUM_WORKING stem phrases may contain duplicates. Delete duplicates form the ranked list of NUM_WORKING stem phrases, preserving the highest ranked phrase.
8. **Add Suffixes**: For each of the remaining stem phrases, find the most frequent corresponding whole phrase in the input text.
9. **Add Capitals**: For each of the whole phrases find the best capitalization.

10. **Final Output**: We now have an ordered list of mixed-case phrases with suffixes added. The score calculated in step 2 orders the list. The top phrases in the list are returned.

Lazy Heading Search

The lazy heading search algorithm is of my own design. It works on the assumption that the headings in a document give some information about the content of text beneath it. The Lazy Heading Search algorithm looks at the headings in an HTML document and ranks them using the method described below.

The program searches for an occurrence of h1 tags. The region between two h1 tags, or between a h1 tag and the bottom of a document (in the case of the last tag) is considered a frame. The frame includes the text of the h1 tag on top of it but not below it because we are interested in a title and t he text under it. The text in the frame has all stop words removed and the remaining words stemmed. Then a score is attached to the title by looking at the words remaining in the title's text and calculating their frequency over the entire frame. If the frequency is high a higher score is given to the title.

Pre-processing of this process was more difficult then first considered, as I had to manually ensure that each heading title was properly placed in the h1 tag and sub headings were in the h2 tag etc. I also needed to ensure that the body text was in the 'normal' format; otherwise my program would ignore it. This problem is caused by the conversion of a chapter from Word format to HTML. For example, the author for a chapter might have defined his own style for some text and when the chapter is converted, that style is stored in a stylesheet (because it wasn't one of the default styles). My program was unable to read stylesheets to parse a document.

Results

Raw numbers

		University of Waikato Kea	NRC Extractor	Microsoft AutoSummarize	Lazy heading search
Chapter 1	Recall	0.70	0.50	1.00	1.00
	Precision	0.70	0.57	1.00	0.77
	Fmeasure	0.70	0.53	1.00	0.87
Chapter 2	Recall	0.20	0.63	0.80	0.80
	Precision	0.25	0.50	0.73	0.62
	Fmeasure	0.22	0.56	0.76	0.70
Chapter 3	Recall	0.20	0.20	0.60	1.00
	Precision	0.29	0.29	0.67	0.67
	Fmeasure	0.24	0.24	0.63	0.80
Chapter 4	Recall	0.30	0.63	0.80	0.90
	Precision	0.60	0.71	1.00	0.82
	Fmeasure	0.40	0.67	0.89	0.86
Chapter 5	Recall	0.20	0.75	0.30	0.80
	Precision	0.33	0.75	0.43	0.80
	Fmeasure	0.25	0.75	0.35	0.80
Chapter 6	Recall	0.40	0.75	0.30	0.80
	Precision	0.44	0.60	0.50	0.67
	Fmeasure	0.42	0.67	0.38	0.73
Chapter 7	Recall	0.40	0.50	0.30	1.00
	Precision	0.36	0.44	0.43	0.67
	Fmeasure	0.38	0.47	0.35	0.80
Chapter 8	Recall	0.40	0.25	0.40	1.00
	Precision	0.57	0.33	0.57	0.71
	Fmeasure	0.41	0.29	0.47	0.83
Chapter 9	Recall	0.70	0.25	0.30	0.60
	Precision	0.64	0.29	0.50	0.55
	Fmeasure	0.67	0.27	0.38	0.57
Chapter 10	Recall	0.50	0.38	0.50	1.00
	Precision	0.56	0.43	0.56	0.71
	Fmeasure	0.53	0.40	0.53	0.83

Remark on the results

I was quite surprised at the quality of the extracted keyphrases in general. With the exception of Microsoft Word's AutoSummarize every algorithm extracted sensible keyphrases. That is keyphrases that could arguably be valid for a chapter. NRC's

Extractor algorithm would have preformed much better had it not given out as many keywords. The Extractor algorithm often would produce keywords instead of proper keyphrases, especially for the first few testing chapters.

Performance of the algorithms

Surprisingly, NRC's Extractor algorithm was the best performer in terms of speed. This is surprising because the Extractor program was web based. Microsoft Word's AutoSummarize was the second best performer in terms of speed. My algorithm was relatively slow, mostly due to the reading operation. I suspect that if I had used some sort of buffered input stream the algorithm would have preformed faster. The actual parsing and extraction was relatively fast because I used Sun Microsystem's **javax.swing.text.HTMLEditorKit** parser, which is a relatively fast parser. KEA preformed the slowest, even though it was run locally on Borg. Although I was unsure of the load on Borg at the time, I doubt that is too much of a factor as the Perl pre-processing probably caused most of the slow down.

The affect of corpus size

NRC's extractor algorithm and my Lazy heading search algorithm performed well the regardless of corpus size. Microsoft's AutoSummarize was the most affected by size. The larger the corpus the worse the results were using AutoSummarize. Similarly KEA performed worse when the corpus size was large.

Conclusions

Of the four algorithms evaluated the non-learning algorithms preformed best overall. This is odd considering the learning algorithms were trained on the same corpus. The Extractor algorithm was the most consistent in producing good keyphrases, although the AutoSummarize produced more correct keyphrases.

The AutoSummarize algorithm was the most interesting for two reasons, its inconsistency and the fact that I knew nothing of the way the algorithm worked.. AutoSummarize would produce wonder keyphrases in one instance then return a table row. This is very strange, considering that the original document was in Word's native format. The inconsistency of AutoSummarize discourages me from using it for future work.

The KEA algorithm was a bit of a disappointment also. Even though it ran on a local machine it was slow and preformed in the middle of the pack in terms of extracted keyphrases. The KEA algorithm is currently being used at the New Zealand Digital Library.

My Lazy Heading Search algorithm worked well for this specific corpus. I ran it using a Ethics paper on software piracy and although it produced some intelligible keyphrases, it was the worst performer. The other three algorithms extracted more sensible keyphrases. The reason is obvious. Because my algorithm assumes that the heading of a section will be the keyphrase for the preceding text—something valid only for technical documents and not always that too. If I restrict the domain of extraction to technical documents, then this makes a more valid algorithm. Currently there are some additions to the algorithm that could possibly make it better. One is to rate a heading text higher if there are sub-headings under it. This works with the assumption that important sections will probably have more sub-sections.

Overall, keyphrase extraction seems very promising as a field of research. It appears to be close to fulfilling a part of the promise of computer message understanding. I suspect in the near future keyphrase creation will be a wholly automatic process.

References

Turney, P. (1999). *Learning to Extract Keyphrases from Text.* Internal paper, Institute for Information Technology, National Research Council Canada, NRC-41622

Turney, P., (1999). *Learning Algorithms for Keyphrase Extraction*. Submitted to Information Retrieval – INRT 34-99

Turney P., (1997). *Extraction of Keyphrases from Text: Evaluation of Four Algorithms*. Institue for Information Technology, National Research Council Canada, ERB-1051.

Belew, R., Amy, M. *Exporting Phrases: A statistical Analysis of Topical Language*. Cognitive Computer Science Research Group, Computer Science & Engr. Dept. (0114) University of California – San Diego

Frank E., Paynter, G., Witten, I., Gutwin, C., and Nevill-Manning, C. (1999*) Domain-Specific Keyphrase Extraction. Proc. DL '99*, pp. 254-256. (Poster presentation.)

[i] Witten I.H., Paynter G.W., Frank E., Gutwin C. and Nevill-Manning C.G. (1999) “KEA: Practical automatic keyphrase extraction”