Domain-Specific Keyphrase Extraction

Eibe Frank and Gordon W. Paynter and Ian H. Witten
Department of Computer Science
University of Waikato
Hamilton, New Zealand

Carl Gutwin
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada

Craig G. Nevill-Manning
Department of Computer Science
Rutgers University
Piscataway, New Jersey, USA

Abstract

Keyphrases are an important means of document summarization, clustering, and topic search. Only a small minority of documents have author-assigned keyphrases, and manually assigning keyphrases to existing documents is very laborious. Therefore it is highly desirable to automate the keyphrase extraction process. This paper shows that a simple procedure for keyphrase extraction based on the naive Bayes learning scheme performs comparably to the state of the art. It goes on to explain how this procedure's performance can be boosted by automatically tailoring the extraction process to the particular document collection at hand. Results on a large collection of technical reports in computer science show that the quality of the extracted keyphrases improves significantly when domain-specific information is exploited.

1 Introduction

Keyphrases give a high-level description of a document's contents that is intended to make it easy for prospective readers to decide whether or not it is relevant for them. But they have other applications too. Because keyphrases summarize documents very concisely, they can be used as a low-cost measure of similarity between documents, making it possible to cluster documents into groups by measuring overlap between the keyphrases they are assigned. A related application is topic search: upon entering a keyphrase into a search engine, all documents with this particular keyphrase attached are returned to the user. In summary, keyphrases provide a powerful means for sifting through large numbers of documents by focusing on those that are likely to be relevant.

Unfortunately, only a small fraction of documents have keyphrases assigned to than—mostly because authors only provide keyphrases when they are explicitly instructed to do so—and manually attaching keyphrases to existing documents is a very laborious task. Therefore, ways of automating this process using artificial intelligence—more specifically, machine learning techniques—are of interest. There are two different ways of approaching the problem: keyphrase *assignment* and keyphrase *extraction.* In keyphrase assignment, also known as text categorization [Dumais *et al.,* 1998], it is assumed that all potential kephrases appear in a predefined controlled vocabulary—the categories. The learning problem is to find a mapping from documents to categories using a set of training documents, which can be accomplished by training a classifier for each category, using documents that belong to it. as positive examples and the rest as negative ones. A new document is then processed by each of the classifiers and assigned to those categories whose classifiers identify it as a positive example. The second approach, keyphrase extraction, which we pursue in this paper, does not restrict the set of possible keyphrases to a selected vocabulary. On the contrary, *any* phrase in a new document can be identified—extracted—as a keyphrase. Using a set of training documents, machine learning is used to determine which properties distinguish phrases that are keyphrases from ones that are not.

Turney [1999] describes a system for keyphrase extraction, GenEx, based on a set of parametrized heuristic rules that are fine-tuned using a genetic algorithm. The genetic algorithm optimizes the number of correctly identified keyphrases in the training documents by adjusting the rules' parameters. Turney compares GenEx to the straightforward application of a standard madiine learning technique—bagged decision trees [Breiman, 1996]—and concludes that it gives superior performance. He also shows that GenEx generalizes well across collections: when trained on a collection of journal articles it successfully extracts keyphrases from web pages on a different topic. This is an important feature because training GenEx on a new collection is computationally very expensive.

This paper briefly summarizes the Kea keyphrase extraction algorithm, and goes cm to show that it generalizes as well as GenEx across collections. In con-

trast to GenEx, however, it does not employ a special-purpose genetic algorithm for training and keyphrase extraction: it is based on the well-known naive Bayes machine learning technique. Training is therefore much quicker. The main finding of this paper is that performance can be boosted significantly if Kea is trained on documents that are from the same domain as those from which keyphrases are to be extracted. This allows us to capitalize on speedy training, because deriving domain-specific models would be less practical with the original lengthy genetic algorithm approach.

Section 2 summarizes the Kea algorithm for keyphrase extraction, and shows that it performs comparably to GenEx if used in the same domain-independent setting. Section 3 explains a simple enhancement that enables Kea to exploit collection-specific information about keyphrases, and shows how this addition boosts performance on a large collection of computer science technical reports. The main findings of this paper are summarized in Section 4.

2 Keyphrase Extraction using Naive Bayes

Keyphrase extraction is a classification task: each phrase in a document is either a keyphrase or not, and the problem is to correctly classify a phrase into one of these two categories. Machine learning provides oif-the-shelf tools for this kind of situation. In machine learning terminology, the phrases in a document are "examples" and the learning problem is to find a mapping from the examples to the two classes "keyphrase" and "not-keyphrase". Machine learning techniques can automatically generate this mapping if they are provided with a set of training examples, that is, examples with class labels assigned to them. In our context, these are simply phrases which have been identified as either being keyphrases or not. Once the learning method has generated the mapping given the training data, it can be applied to unlabeled data, in other words, it can be used to extract keyphrases from new documents.

2.1 Generating Candidate Phrases

Not all phrases in a document are equally likely to be keyphrases *a priori* In order to facilitate the learning process, most phrases that appear can be eliminated from the set of examples that are presented to the learning scheme.

First, the input text is split up according to phrase boundaries (punctuation marks, dashes, brackets, and numbers). Non-alphanumeric characters (apart from internal periods) and all numbers are deleted. Kea takes all subsequences of these initial phrases up to length three as candidate phrases. It than eliminates those phrases that begin, or end, with a stopword- It also deletes phrases that consist merely of a proper noun. In the next step, all words are case-folded and stemmed using the iterated Lovins stemmer [Lovins, 1968], and stemmed phrases that occur only once in the document are removed.

2.2 Building the Model

So far we have shown how candidate phrases are generated. However, in conventional machine learning terms, phrases by themselves are useless—it is their properties, or "attributes," that are important. Several plausible attributes immediately spring to mind: the number of words in a phrase, the number of characters, the position of the phrase in the document, etc. However, in our experiments, only two attributes turned out to be useful in discriminating between keyphrases and non-keyphrases: the ***TF×IDF*** score of a phrase, and the *distance into* the document of the phrase's first appearance. In the following we explain how these attributes are computed and how a naive Bayes model [Domingos and Pazzani, 1997] is built from them.

The ***TF×IDF*** score of a phrase is a standard metric in information retrieval. It is designed to measure how specific a phrase *P* is to a given document *Dz*

$$TF{\times}IDF(P,D) = \Pr[\text{phrase in } D \text{ is } P] \times -\log \Pr[P \text{ in a document}].$$

The first probability in this equation is estimated by counting the number of times the phrase *P* occurs in the document D, and the second one by counting the number of documents in the training corpus that contain *P* (excluding *D).*[1]

The *distance* of a phrase from the beginning of a document is calculated as the number of words that precede its first appearance, divided by the number of words in the document. The resulting feature is a number between 0 and 1 that represents the proportion of the document preceding the phrase's first appearance.

Both these attributes are real numbers. The naive Bayes learning method can process numeric attributes by assuming, for example, that they are normally distributed. However, we obtained better results by discretizing the attributes prior to applying the learning scheme [Domingos and Pazzani, 1997]. This indicates that the normal distribution is not appropriate in this application. Discretization quantizes a numeric attribute into ranges so that the resulting new attribute can be treated as a nominal one: each value represents a range of values of the original numeric attribute. Kea uses Fayyad and Irani's [1993] discretization scheme, which is based on the Minimum Description Length principle. It recursively splits the attribute into intervals, at each stage minimizing the entropy of the class distribution. It stops splitting when the total cost for encoding both the discretization and the class distribution cannot be reduced further.

The naive Bayes learning scheme is a simple application of Bayes' formula. It assumes that the attributes—in this case *TFxIDF* and *distance*—are independent

[1]The counters are initialized to one to avoid taking the logarithm of zero.

given the class. Making this assumption, the probability that a phrase is a key phrase given that it has discretized *TFxIDF* value T and discretized *distance D* is:

$$\Pr[key|T, D] = \frac{\Pr[T|key] \times \Pr[D|key] \times \Pr[key]}{\Pr[T, D]},$$

where *Pr[T/key]* is the probability that a keyphrase has *TFxIDF* score T, *Pr[D\key]* the probability that it has *distance* D, Pr[key] the probability that a phrase is a keyphrase, and Pr*[T, D]* a normalization factor that makes Pr[key/ T, *D]* lie between zero and one. All these probabilities can be estimated reliably by counting the number of times the corresponding event occurs in the training data.[2]

It has been shown that naive Bayes can be a very accurate classification method even if the independence assumption is not correct [Domingos and Pazzani, 1997]. However, it can be argued that the two attributes we use, *TFxIDF* and *distance,* are close to being independent given the class. This implies that naive Bayes is close to being the optimum classification method for this application, and might be the reason why it performs better than all other learning methods that we have investigated. (In particular it performs better than bagged decision trees, as we show in Section 2.4.)

2.3 Extracting Keyphrases

Kea uses the procedure described above to generate a naive Bayes model from a set of training documents for which keyphrases are known (typically because the author provided them). The resulting model can then be applied to a new document from which keyphrases are to be extracted.

First, Kea computes *TFxIDF* scores and *distance* values for all phrases in the new document using the procedure described above, taking the discretization obtained from the training documents. (Both attributes can be computed without knowing whether a phrase is a keyphrase or not.) The naive Bayes model is then applied to each phrase, computing the estimated probability of it being a keyphrase. The result is a list of phrases ranked according to their associated probabilities. Assuming that the user wants to extract r keyphrases, Kea then outputs the r highest ranked phrases.

There are two special cases that have to be addressed in order to achieve optimum performance. First, if two phrases have equal probability—which is quite likely to happen due to the discretization—they are ranked according to their *TFxIDF* score (in its pre-discretized form). Second, if a phrase is a subphrase of another phrase, it is only accepted as a keyphrase if it is ranked higher; otherwise it is deleted from the list before the r top-ranking phrases are output.

2.4 Experimental Results

We have evaluated Kea on several different document collections with author-assigned keyphrases. Our criterion for success is the extent to which Kea produces the same stemmed phrases as authors do.[3] Because this method of evaluation is the same as used by Turney [1999], we can directly compare Kea's performance to his results.

Comparison to GenEx

We compared Kea and GenEx using two experimental settings from Turney's paper-[4] The first one involves training and testing on journal articles. In this setting, 55 articles are used for training (6 from the *Journal of the International Academy of Hospitality Research,* 2 from *The Neuroscientist,* 14 from the *Journal of Computer-Aided Molecular Design,* and 33 from *Behavioral and Brain Sciences),* and 20 for testing (all from *Psycoloquy).* In the second setting, the same documents are used for training but 35 *FIPS* web pages ate used for testing.

Table 1 shows the number of correctly identified author-provided keyphrases among the five and fifteen top-ranking phrases output by the extraction algorithms. Four extraction algorithms are represented: GenEx, fifty bagged C4.5 decision trees [Quinlan, 1992] as used by Turney, Kea, and Kea using fifty bagged C4.5 trees instead of the naive Bayes learning scheme. Results for the first two methods are from Turney's paper.[5] The third scheme is the standard Kea algorithm that we have described. In the fourth, bagged C4.5 trees were used instead of discretization and naive Bayes, with all the standard pre- and post-processing done by Kea. This variation of Kea is computationally much more expensive (by a factor of at least fifty).

Turney found bagged C4.5 trees to perform universally worse than GenEx, but in only one of the four experimental settings from Table 1, Journal/FIPS with cutoff of five, was the difference statistically significant. Kea sometimes performs worse than GenEx and sometimes better; the differences are not statistically significant (at the 5% level, according to a t-test). Moreover, Kea-C4.5 performs much better than Turney's C4.5 in the case where the latter does significantly worse than GenEx. We conclude that GenEx and Kea perform at about the same level, Kea-C4.5 seems slightly worse but the difference is not statistically significant on these datasets. The only statistically significant result is the poor performance that Turney observed in one case with C4.5.

The difference between Turney's findings for bagged C4.5 trees and ours deserves some explanation. Turney uses many more attributes—among them *distance-but* he does not use *TFxIDF.* Moreover, he performs no post-processing for C4.5—although he does for GenEx—

[2] The naive Bayes implementation used by Kea initializes all counts to one.

[3] Author-assigned keyphrases are, of course, deleted from the documents before they are given to Kea.

[4] We could not compare Kea on the other document collections used by Turney because we did not have access to his corpus of email messages, which contains confidential information.

[5] To get the number of correctly identified keyphrases, Turney's "precision" figures were multiplied by the cutoff employed (five or fifteen).

Experimental conditions		Turney [1999]'s results		Kea results	
Training/testing	Cutoff	GenEx	C4.5	Kea	Kea-C4.5
Journal/Journal	5	1.45±1.24	1.40±1.28	1.35±0.93	1.20±0.83
	15	2.65±1.95	2.55±1.70	2.75±1.25	2.70±1.38
Journal/FIPS	5	1.43±0.85	0.77±0.81	1.46±0.98	1.40±0.95
	15	2.46±1.17	2.12±0.99	2.20±1.35	2.26±1.32

Table 1: Experimental results for different extraction algorithms

whereas we remove subphrases if they do not perform better than their superphrases. These appear to be the main differences between his way of applying C4.5 and ours.

Changing the Amount of Training Data

An interesting question is how Kea's performance scales with the amount of training data available. In order to investigate this, we performed experiments with a large collection of computer science technical reports *(CSTR)* from the New Zealand Digital Library (www.nzdl.org). The documents in CSTR are fairly noisy, partly because the source files have been extracted automatically from PostScript. Also, they contain on average fewer keyphrases than the other collections. This makes keyphrase extraction in this domain more difficult than in the other corpuses.

There are two potential ways in which the corpus of documents that is available can influence Kea's performance on fresh data. First, training documents are used when computing both the discretization of the attributes, and the corresponding counts for the naive Bayes model. It is essential that these documents have keyphrases assigned to them because the learning method needs labeled examples. Second, the document corpus supports the learning process when each phrase's "document frequency" is calculated—this is used for deriving its *TFxIDF* score. In this case the documents need not be labeled. Our experiments showed that no further performance improvement was gained by increasing the number of documents used to compute the document frequencies beyond 50.

To illustrate the effect of training set size, Figure 1 shows Kea's performance on an independent set of 500 test documents. It plots the number of "correct" keyphrases, for both five and fifteen phrases extracted, against the number of documents used for training, from 1 through 130 files. The error bars give 99% confidence intervals derived by training Kea on ten different training sets of the same size. We used the same independent 100 documents for calculating the document frequencies throughout this particular experiment. It can be seen from Figure 1 that if more than twenty documents are used for training, little is gained by increasing the number further. With 50 documents, there is no further performance improvement.

These results show that Kea's performance is close to optimum if about 50 training documents are used; in other words, 50 labeled documents are sufficient to push performance to the limit. However, Section 3 demonstrates that this is not the case if domain-specific information is exploited in the learning and extraction process. In that case, much larger amounts of labeled training documents prove beneficial.

Subject Area of Training Documents

Now we investigate the extent to which models formed by Kea transfer from one subject domain to another. To this end we use the collection of journal articles described above, and two collections of web pages also used by Turney (1999], Aliweb, and NASA, all of which have keyphrases assigned. The basic procedure was to train on one of the collections and test on another, producing nine combinations. For each collection we chose 55 training documents at random and used the rest for testing, 20 for the journal articles, 35 for Aliweb, and 86 for NASA. The training documents were used to compute the document frequencies; thus the entire keyphrase assignment model was based on the training documents alone. For the journal articles, as well as the randomly-chosen test set, we ran experiments with the same training/testing division that Turney [1999] used, the test set comprising 20 articles in the journal *Psycoloquy.*

Figure 2 shows the average number of correct keyphrases returned when five keyphrases are retrieved, for twelve cases. The first nine represent every combination of training and testing sets drawn from one of the three collections, and the last represents the *Psycoloquy* test set with the same three training sets (except

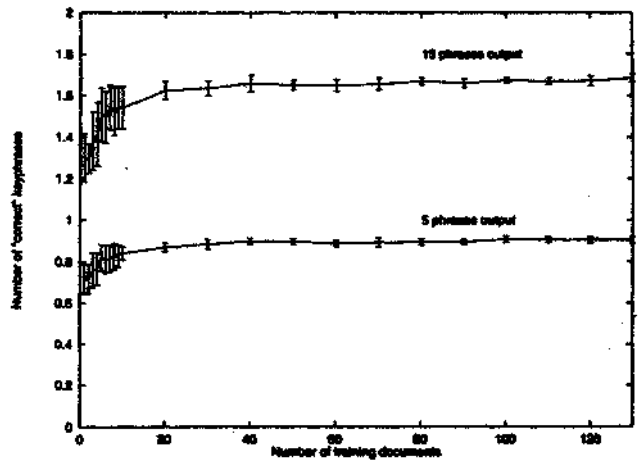

Figure 1: Performance on *CSTR* corpus for different numbers of training files (error bars show 99% confidence intervals)