METHODS FOR KEY PHRASE EXTRACTION FROM DOCUMENTS

Baljinder Kaur[1], Brahmaleen Kaur Sidhu[2]
Department of Computer Engineering
Punjabi University
Patiala, India.

Abstract: The word keyphrase implies two features: phraseness and informativeness. It is the process of obtaining the keyphrases which are available in the body of the text document. Single document keyphrase extraction usually make use of only the information contained in the specified document. It is used to extract most frequent words which are significant with respect to the applications. This technique is most frequently used in search engines for advertisement. This paper discusses various algorithms and tools for keyphrase extraction from documents. Application areas are also discussed. The last section of paper compares the pursued algorithms.

Keywords: keyphrase extraction, KEA, C4.5, and GenEx.

I. INTRODUCTION

In general, a phrase is a group of words acting as a single part of speech and not containing both a subject and a verb. It is a part of a sentence, and does not express a complete thought. This sentence contains five phrases: "of words", "acting as a single part of speech", "as a single part", "of speech", and "not containing both a subject and a verb". Technically a phrase is a unit of language larger than a word but smaller than a sentence. Keyphrase can be defined as a phrase of one to three words to capture the main topic. The word keyphrase implies two features: phraseness and informativeness. Phraseness is a somewhat abstract notion which describes the degree to which a given word sequence is considered to be a phrase. Informativeness refers to how well a phrase captures or illustrates the key ideas in a set of documents. Because informativeness is defined with respect to background information and new knowledge, users will have different perceptions of informativeness. A Keyphrase list usually consist of 5 to 15 keyphrases. There exist two types of concepts, keyphrase generation and keyphrase extraction where keyphrase generation is obtaining the keyphrases some of which are not available in the body of the text document and keyphrase extraction is obtaining the keyphrases which are available in the body of the text document.

Broadly speaking keyphrase extraction involve the following steps:

- **Input restructuring :** In this step required n-grams are chosen on the basis of parts of speech pattern.
- **Tokenization :** It is the process of chopping a character sequence in a document into pieces, called tokens and at the same time throwing away certain characters, such as punctuation.
- **Forming candidates :** n-grams chosen in previous steps are marked as keyphrase candidates.
- **Scoring :** A suitable score is associated with each keyphrase candidate.
- **Selection :** Above calculated score is responsible for document keyphrase selection.

Commonly used POS patterns [1]:

1. **POS patterns for 3-grams**
   (a) (N or named entity), (V or A or stop word), (N or named entity)
   (b) 3x (named entity) example: Tim Berners Lee
2. **POS patterns for 2-grams**
   (a) A (N or named entity)
   (b) 2x (named entity) example: Bill Gates

II. APPLICATION AREAS IN KEYPHRASE EXTRACTION

There are at least five general application areas for keyphrases [2]:

1. **Text Summarization**
   (a) Mini-summary : Automatic keyphrase extraction can provide a quick mini-summary for a long document. For example, it could be a feature in a web browser; just click the summarize button when browsing a long web page, and then a window pops up with the list of keyphrases.
   (b) Annotated Lists : Automatic keyphrase extraction can supply added information in a list or table of contents. For example, each item in the hit list generated by a web search engine could have a list of keyphrases in addition to the standard.
   (c) Labels : Keyphrases can supply quickly understood labels for documents in a user interface where there is a need to display a set of documents.
   (d) Author Assistance : Automatic keyphrase extraction can help an author or editor who wants to supply a list of keyphrases for a document. For example, the administrator of a web site might want to have a keyphrase list at the top of each web page. The automatically extracted phrases can be a starting point for further manual refinement by the author or editor.

(e) Text Compression : On a device with limited display capacity or limited bandwidth, keyphrases can be a substitute for the full text. For example, an email message could be reduced to a set of keyphrases for display on a pager; a web page could be reduced for display on a portable wireless web browser.

2. **Human-Readable Index**

(a) Journal Index : Automatic keyphrase extraction can be used to generate a human-readable index, for access to journal articles or magazine articles.

(b) Resource Access : It can provide automatic generation of a human-readable index for access to resources, such as a yellow pages, etc.

(c) Internet Access : It can provide automatic generation of a human-readable index for access to web pages, ftp, etc.

(d) On-the-fly Indexing: Automatic keyphrase extraction can generate a human-readable index for a dynamically generated cluster of documents

(e) Back-of-the-book Index : It can supply a human-readable index for a book or on-line documentation. Although a list of keyphrases typically contains less than ten items, whereas a back-of-the-book index may contain hundreds or thousands of items, an automatic keyphrase extraction algorithm could be used to generate a back-of-the-book index by breaking a long document into a series of short documents.

3. **Interactive Query Refinement**

(a) Narrow Hit List : Automatic keyphrase extraction can provide suggestions for improving a query. Often a query with a conventional search engine returns a huge list of matching documents. The user would like to narrow the list by adding new terms to the query, but it is not clear what terms should be added.

(b) Expand Hit List : New terms can be added to a query by disjunction, instead of conjunction, which will yield a longer hit list.

4. **Machine-Readable Index**

(a) Improved Precision : It can improve conventional search by improved weighting of terms and phrases in a conventional search engine index table, resulting in more precise retrieval of documents (i.e., fewer irrelevant documents).

(b) Compression : It can be used to compress the index tables that are used by conventional search engines, which can save memory space in constrained applications or very large applications .

5. **Keyphrases for Highlighting**

(a) When we skim a document, we scan for keyphrases, to quickly determine the topic of the document. Highlighting is the practice of emphasizing keyphrases and key passages (e.g., sentences or paragraphs) by underlining the key text, using a special font, or marking the key text with a special color. The purpose of highlighting is to facilitate skimming. Some software tools for document management now support skimming by automatically highlighting keyphrases.

6. **Keyphrases for Indexing**

(a) An alphabetical list of keyphrases, taken from a collection of documents or from parts of a single long document (e.g., chapters in a book), can serve as an index. The alphabetical list of keyphrases was generated using GenEx.

7. **Keyphrases for Interactive Query Refinement**

(a) 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. The left frame shows the matching documents and the right frame lists suggestions for narrowing the original query. These suggestions are keyphrases extracted by GenEx from the documents that are listed in the left frame. The query terms are combined by conjunction, so the hit list becomes smaller with each iteration. However, adding the suggested terms will never result in an empty hit list, because the terms necessarily appear in at least one of the documents in the hit list.

III. TOOLS USED IN KEYPHRASE EXTRACTION

Four basic tools are discussed in this paper :

1. **Carrot2**
Carrot2 is a library and a set of supporting applications you can use to build a search results clustering engine [3]. Such an engine will organize your search results into topics, fully automatically and without external knowledge such as taxonomies or preclassified content. Carrot2 contains two document clustering algorithms designed specifically for search results clustering: Suffix Tree Clustering and Lingo. It is a great tool for key phrase extraction. It uses two algorithm STC and lingo. Lingo search complete key phrase with some other constraints key phrase. STC is kind of Suffix Trie.Carrot2 also contains components for fetching search results from several search engines, such as Bing, Google, but it also supports other sources of documents like Lucene or Solr indexes. Carrot2 is not a search engine itself, it does not have a crawler and indexer. There is a number of Open Source projects you can use to crawl (Nutch), index and search (Lucene, Solr) your content, which can then be queried and clustered

by Carrot2 .Carrot2 comes with a suite of tools and APIs that we can use to quickly set up clustering on our own data, tune clustering results, call Carrot2 clustering from your Java or C code or access Carrot2 clustering as a remote service.Carrot2 Document Clustering Server (DCS) exposes Carrot2 clustering as a REST service. It can cluster documents from an external source (e.g. a search engine) or documents provided directly as an XML stream and returns results in XML or JSON formats. Carrot2 Web Application exposes Carrot2 clustering as a web application for end users. It allows users to browse clusters using a conven

2. **Maui**
   It is basic KEA tool but also gives options to boost taxonomy from Wikipedia. Maui automatically identifies main topics in text documents. Depending on the task, topics are tags, keywords, keyphrases, vocabulary terms, descriptors, index terms or titles of Wikipedia articles [4]. Maui performs the following tasks:
   - Term assignment with a controlled vocabulary (or thesaurus)
   - Subject indexing
   - Topic indexing with terms from Wikipedia
   - Keyphrase extraction
   - Terminology extraction
   - Automatic tagging
3. **WikiFier**
   Enrichment of text documents with links to Wikipedia's pages has become an extremely popular task. This task is called wikifieation [5]. Wikifieation is necessary for intelligent systems that use knowledge extracted from Wikipedia for different purposes. Showing wikified documents to reader of blogs or news feed is common as well. Enrichment text with links to Wikipedia usually consists of two steps: extraction of key terms from a document and associating these terms with Wikipedia pages. Lexical ambiguity of language presents a main difficulty for automatic wikifieation. Therefore, word sense disambiguation (WSD) is a necessary step for the automatic wikifiers. User selects by mouse some part of the text to mark up a term there. It's very important to accurately select the term boundaries, so we had implemented several techniques that help users to do that. To make the test creating process more easy we provide automatic preprocessing feature which uses wikifier described in to automatically detect terms in documents, assign them right meanings and select key concepts.
4. **Mallet**
   Mallet is a Java-based package for statistical natural language processing, document classification, clustering, topic modeling, information extraction, and other machine learning applications to text [6]. Mallet includes sophisticated tools for document classification: efficient routines for converting text to "features", a wide variety of algorithms, and code for evaluating classifier performance using several commonly used metrics. n addition to classification, Mallet includes tools for sequence tagging for applications such as named-entity extraction from text. Algorithms include Hidden Markov Models, Maximum Entropy Markov Models, and Conditional Random Fields. These methods are implemented in an extensible system for finite state transducers. Many of the algorithms in Mallet depend on numerical optimization. Mallet includes an efficient implementation of Limited Memory BFGS, among many other optimization methods.

IV. ALGORITHMS

Three algorithms are discussed in this paper :

1. **KEA : Key Extraction Algorithms**
   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 [7]. 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; TF*IDF(term frequency * inverse document frequency) 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.

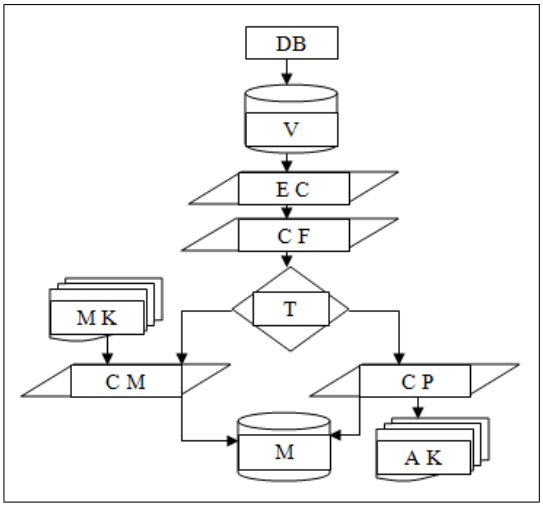

Figure 1

All the components included in Figure 1 are explained as under :

- DB - Database
- V - Vocabulary
- E C - Extracting Candidates
- C F - Calculating Features
- T - Training

- CM - Compute Model
- CP - Compute Probabilities
- M - Model
- MK - Manual Keyphrases
- AK - Automatic Keyphrases

(a) Database : Kea gets a directory name and processes all documents in this directory that have the extension ".txt".

(b) Vocabulary - If a vocabulary is provided, Kea matches the documents' phrases against this file.

(c) Extracting Candidates : Here Kea extracts n-grams of a predefined length that do not start or end with a stopword.

(d) Features : For each candidate phrase Kea computes 4 feature values: TF * IDF is a measure describing the specificity of a term for this document under consideration, compared to all other documents in the corpus.

i. First occurrence is computed as the percentage of the document preceding the first occurrence of the term in the document.

ii. Length of a phrase is the number of its component words. Two-word phrases are usually preferred by human indexers.

iii. Node degree of a candidate phrase is the number of phrases in the candidate set that are semantically related to this phrase

(e) Building the model : Before being able to extract keyphrases from new documents, Kea first needs to create a model that learns the extraction strategy from manually indexed documents. This means, for each document in the input directory there must be a file with the extension ".key" and the same name as the corresponding document. This file should contain manually assigned keyphrases, one per line.

(f) Extracting keyphrases : When extracting keyphrases from new documents, Kea takes the model and feature values for each candidate phrase and computes its probability of being a keyphrase. Phrases with the highest probabilities are selected into the final set of keyphrases. The user can specify the number of keyphrases that need to be selected.

2. **C4.5**

C4.5 is a decision tree induction algorithm that classify phrases as positive or negative examples of keyphrases [7]. For keyphrase extraction, a case is a candidate phrase, which we wish to classify as a positive or negative example of a keyphrase. We classify a case by examining its features. A feature can be any property of a case that is relevant for determining the class of the case. C4.5 can handle real-valued features, integer-valued features, and features with values that range over an arbitrary, fixed set of symbols. C4.5 takes as input a set of training data, in which cases are represented as feature vectors. C4.5 generates as output a decision tree that models the relationships among the features and the classes. Let C1, C2,âĂę, Ck be classes and T be the set of training samples in the given node of decision tree, then;

- T contains one or more samples, all belonging to a single class Cj.
- T contains no samples.
- T contains samples that belong to a mixture of classes.
- Entropy test is carried out as follows:

$$Info(S) = -\Sigma((freq(C_i, S)/|S|) - log2(freq(C_i, S)/|S|)) \quad (1)$$

We convert a document into a set of feature vectors by first making a list of all phrases of one, two, or three consecutive non-stop words that appear in the given document. (Stop words are words such as "the", "of", "and".) It uses Iterated Lovins stemmer to find the stemmed form of each of these phrases. The Lovins stemmer is more likely to map two words to the same stem, but it is also more likely to make mistakes. For example, the Lovins stemmer correctly maps "psychology" and "psychologist" to the same stem, "psycholog. We have found that aggressive stemming is better for keyphrase extraction than conservative stemming. In our experiments, we have used an aggressive stemming algorithm that we call the Iterated Lovins stemmer. C4.5 uses 12 feature vectors as given below:

- **stemmed-phrase :** The stemmed form of a phrase used for matching with human-generated phrases.
- **whole-phrase :** The most frequent whole phase Corresponding to the given stemmed phrase for output and for calculating features 8 to 11.
- **num-words-phrase :** The number of words in the stemmed phrase ,ranging from is one to three.
- **first-occur-phrase :** The first occurrence of the stemmed phrase which is normalized by dividing by the number of words in the document (including stop words).
- **first-occur-word :** The first occurrence of the earliest occurring single stemmed word in the stemmed phrase normalized by dividing by the number of words in the document (including stop words).
- **freq-phrase :** The frequency of the stemmed phrase, normalized by dividing by the number of words in the document (including stop words).
- **freq-word :** Frequency of the most frequent single stemmed word in the stemmed phrase normalized by dividing by the number of words in the document (including stop words).
- **relative-length :** The relative length of the most frequent whole phrase - the number of characters

in the whole phrase, normalized by dividing by the average number of characters in all candidate phrases.

- **proper-noun :** Is the whole phrase a proper noun? It is based on the most frequent whole phrase.
- **final-adjective :** Does the whole phrase end in a final adjective? It is based on the most frequent whole phrase.
- **common-verb :** Does the whole phrase contain a common verb?
- **Class :** Is the stemmed phrase a keyphrase ? It is based on match with stemmed form of human generated keyphrases.

C4.5 has access to nine features (features 3 to 11) when building a decision tree. The leaves of the tree attempt to predict class (feature 12). When a decision tree predicts that the class of a vector is 1, then the phrase whole-phrase is a keyphrase, according to the tree. This phrase is suitable for output for a human reader. We used the stemmed form of the phrase, stemmed-phrase, for evaluating the performance of the tree.

3. **GenEx : Generator Extractor**
GenEx is a Hybrid Genetic Algorithm for Keyphrase Extraction. GenEx has two components, the Genitor genetic algorithm and the Extractor keyphrase extraction algorithm. Extractor takes a document as input and produces a list of keyphrases as output [7]. Extractor has twelve parameters that determine how it processes the input text. In GenEx, the parameters of Extractor are tuned by the Genitor genetic algorithm to maximize performance on training data. Genitor is used to tune Extractor, but Genitor is no longer needed once the training process is complete.

1. **Extractor**
What follows is a conceptual description of the Extractor algorithm. For clarity, we describe Extractor at an abstract level that ignores efficiency considerations. That is, the actual Extractor software is essentially an efficient implementation of the following algorithm. There are ten steps to the Extractor algorithm. Figure below summarizes the ten steps. Steps 4 and 5 are conceptually independent of steps 1, 2, and 3, so they are represented as a separate sequence. (For efficiency reasons, in the actual implementation of the algorithm, several steps are interleaved.) Table 2 is a list of the 12 parameters of Extractor, with a brief description of each of them. The meaning of the parameters should become clear as the algorithm is described. Figure 2 below specifies ten steps to the extractor algorithm

(a) Find Single Stems : Make a list of all of the words in the input text. Drop words with less than three characters. Drop stop words, using a given stop word list. Convert all remaining words to lower case. Stem the words by truncating them at STEM-LENGTH characters.

(b) 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.Extractor has 12 parameters:

- NUM-PHRASES
- NUM-WORKING
- FACTOR-TWO-ONE
- FACTOR-THREE-ONE
- MIN-LENGTH-LOW-RANK
- MIN-RANK-LOW-LENGTH
- FIRST-LOW-THRESH
- FIRST-HIGH-THRESH
- FIRST-LOW-FACTOR
- FIRST-HIGH-FACTOR
- STEM-LENGTH
- SUPPRESS-PROPER

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 . Typically FIRST-LOW-FACTOR is greater than one and FIRST-HIGH-FACTOR is less than one. Thus, early, frequent stems receive a high score and late, rare stems receive a low score.

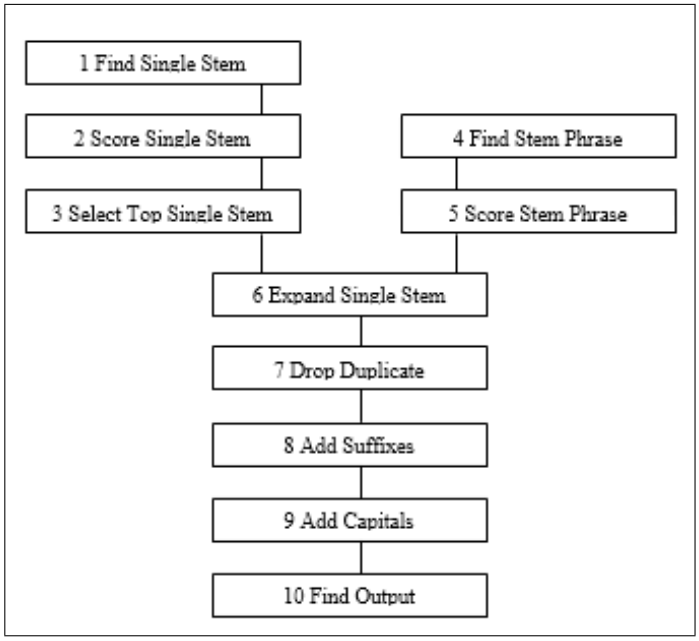

Figure 2

(c) Select Top Single Stems : Rank the stems in order of decreasing score and make a list of the top NUM-WORKING single stems.

(d) Find Stem Phrases : Make a list of all phrases in the input text. A phrase is defined as a sequence of one, two, or three words that appear consecutively in the text, with no intervening stop words or punctuation. Stem each phrase by truncating each word in the phrase at STEM-LENGTH characters.

(e) 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, using the parameters,FIRST-LOW-FACTOR,FIRST-LOW-THRESH, FIRST-HIGH-FACTOR, and FIRST-HIGH-THRESH. Then make an adjustment to each score, based on the number of stems in the phrase. If there is only one stem in the phrase, do nothing. If there are two stems in the phrase, multiply the score by FACTOR-TWO-ONE. If there are three stems in the phrase, multiply the score by FACTOR-THREE-ONE. Typically FACTOR-TWO-ONE and FACTOR-THREE-ONE are greater than one, so this adjustment will increase the score of longer phrases. A phrase of two or three stems is necessarily never more frequent than the most frequent single stem contained in the phrase.

(f) Expand Single Stems : For each stem in the list of the top NUM-WORKING single stems, find the highest scoring stem phrase of one, two, or three stems that contains the given single stem. The result is a list of NUM-WORKING stem phrases. Keep this list ordered by the scores calculated in step 2. Now that the single stems have been expanded to stem phrases, we no longer need the scores that were calculated in step 5.

(g) Drop Duplicates : The list of the top NUM-WORKING stem phrases may contain duplicates. For example, two single stems may expand to the same two-word stem phrase. Delete duplicates from the ranked list of NUM-WORKING stem phrases, preserving the highest ranked phrase.

(h) Add Suffixes : For each of the remaining stem phrases, find the most frequent corresponding whole phrase in the input text. When counting the frequency of whole phrases, if a phrase has an ending that indicates a possible adjective, then the frequency for that whole phrase is set to zero.

(i) Add Capitals : For each of the whole phrases (phrases with suffixes added), find the best capitalization, where best is defined as follows. For each word in a phrase, find the capitalization with the least number of capitals.

(j) Final Output : We now have an ordered list of mixed-case (upper and lower case, if appropriate) phrases with suffixes added. The list is ordered by the scores calculated in step 2. That is, the score of each whole phrase is based on the score of the highest scoring single stem that appears in the phrase. The length of the

2. **Genitor**
A genetic algorithm may be viewed as a method for optimizing a string of bits, using techniques that are inspired by biological evolution. A genetic algorithm works with a set of bitstrings, called a population of individuals. The initial population is usually randomly generated. New individuals (new bit strings) are created by randomly changing existing individuals (this operation is called mutation) and by combining substrings from parents to make new children (this operation is called crossover). Each individual is assigned a score (called its fitness) based on some measure of the quality of the bit string, with respect to a given task. Fitter individuals get to have more children than less fit individuals. As the genetic algorithm runs, new individuals tend to be increasingly fit, up to some asymptote.

3. **GenEx : Generator plus Extractor**
The algorithm is tuned with a dataset, consisting of documents paired with target lists of keyphrases. The dataset is divided into training and testing subsets. The learning process involves adjusting the parameters to maximize the match between the output of Extractor and the target keyphrase lists, using the training data. The success of the learning process is measured by examining the match using the testing data. We assume that the user sets the value of NUM-PHRASES, the desired number of phrases, to a value between five and fifteen. We then set NUM-WORKING too. The remaining ten parameters are set by Genitor. Genitor uses a binary string of 72 bits to represent the ten parameters, as shown in Table 11. We run Genitor with a population size of 50 for 1050 trials (these are default settings). Each trial consists of running Extractor with the parameter settings specified in the given binary string, processing the entire training set. The final output of Genitor is the highest scoring binary string. Ties are broken by choosing the earlier string. We first tried to use the average precision on the training set as the fitness measure, but GenEx discovered that it could achieve high average precision by adjusting the parameters so that less than NUM-PHRASES phrases were output. This is not desirable, so we modified the fitness measure to penalize GenEx when less than NUM-PHRASES phrases were output:

$$total - matches = total\ number\ of\ matches\ between\ GenEx\ and\ human \quad (1)$$

$$total - machinephrases = total\ number\ of\ phrases\ output\ by\ GenEx \quad (2)$$

$$precision = total - matches/total - machine - phrases \quad (3)$$

$$num - docs = number\ of\ documents\ in\ training\ set \quad (4)$$

$$total - desired = num - docsNUM - PHRASES \quad (5)$$

$$penalty = (total - machine - phrases/total - desired \quad (6)$$

$$fitness = precisionpenalty \quad (7)$$

The penalty factor varies between 0 and 1. It has no effect (i.e., it is 1) when the number of phrases output by GenEx equals the desired number of phrases. The penalty grows (i.e., it approaches 0) with the square of the gap between the desired number of phrases and the actual number of phrases.

V. COMPARISION OF C4.5 WITH GENEX

A comparison of Extractor with the feature vectors we used with C4.5 shows that GenEx and C4.5 are learning with essentially the same feature sets. The two algorithms have access to the same information, but they learn different kinds of models of keyphrases. This section lists some of the more significant differences between GenEx and C4.5 .

1. Given a set of phrases with a shared single-word stem (for example, the set of phrases "learning","machine learning", "learnability" shares the single-word stem "learn"), GenEx tends to choose the best member of the set, rather than choosing the whole set. GenEx first identifies the shared single-word stem (steps 1 to 3) and then looks for the best representative phrase in the set (steps 4 to 6). C4.5 tends to choose several members from the set, if it chooses any of them.
2. GenEx can adjust the aggressiveness of the stemming, by adjusting STEM-LENGTH. C4.5 must take the stems that are given in the training data.
3. C4.5 is designed to yield high accuracy. GenEx is designed to yield high precision for a given NUM-PHRASES. High precision does not necessarily correspond to high accuracy.
4. C4.5 uses the same model (the same set of decision trees) for all values of NUM-PHRASES. With C4.5, We select the top NUM-PHRASES most probable feature vectors, but our estimate of probability is not sensitive to the value of NUM-PHRASES. On the other hand, Genitor tunes Extractor differently for each desired value of NUM-PHRASES.
5. GenEx might output less than the desired number of phrases, NUM-PHRASES, but C4.5 (as we use it here) always generates exactly NUM-PHRASES phrases. Therefore, in the following experiments, performance is measured by the average precision, where precision is defined by equation (8), not by equation (9). Equation (8) ensures that GenEx cannot spuriously boost its score by generating fewer phrases than the user requests.

$$precision = number\ of\ matches / desired\ number\ of\ machine-generated phrases \quad (8)$$

$$precision = number\ of\ matches / actual\ number\ of\ machine-generated\ phrases \quad (9)$$

References

[1] Martin Dostal and Karel Jezek. Automatic Keyphrase Extraction based on NLP and Statistical Methods. *Department of Computer Science and Engineering, Faculty of Applied Sciences University of West Bohemia*, 2011.

[2] P. Turney. Extraction of Keyphrases from Text:Evaluation of Four Algorithms. *National Research Council, Institute for Information Technology*, 1999.

[3] Peter D. Turney. Learning Algorithms For Keyphrase Extraction. *National Research Council, Institute for Information Technology*, 1999.

[4] `http://code.google.com/p/maui-indexer/`.

[5] `http://wikifier.labs.exalead.com/`.

[6] `http://mallet.cs.umass.edu/topics.php/`.

[7] `http://project.carrot2.org/`.

