Significant Phrases Detection

Mikhail Bautin
Department of Computer Science
Stony Brook University
Stony Brook, NY 11790
mbautin@cs.sunysb.edu

Michael Hart
Department of Computer Science
Stony Brook University
Stony Brook, NY 11790
mahart@cs.sunysb.edu

Abstract

The problem of determining key words and phases which best characterize a text document has important applications such as building a compact index for a large-scale text processing system, or using a keyword set for summarization and topic detection. We approached this problem from two perspectives. Our knowledge-poor approach is based on statistical collocation detection using the t-test and likelihood ratio, and applying latent semantic analysis to identify terms important in a particular document. The knowledge-rich approach addresses the problem using noun phrase chunking and coreference resolution. Both approaches use a decision tree classifier to answer whether a given phrase is a key word looking at the set of calculated features. We have built prototypes and compared results of these two approaches.

1 Introduction

Amazon.com, one of the premier bookselling sites on the Internet, provides extensive information to enrich the user's shopping experience. One of the features of the book listings is called "Statistically Improbable Phrases" (SIPs). These phrases are occur significantly more often in the text than they do relative to other books. In some cases, the SIPs are representative of the main ideas, topics and plot elements of a particular piece. We find, however, that SIPs are not infallible and may produce phrases that have no bearing on the content in general. Therefore, it is clear that there are phrases in text that are significant inasmuch that they signify the content of the document. We pose this research question: by selecting and displaying significant phrases, are we able to give users a sense of the general ideas, better understanding, and increased search power of the text? What properties do signicant phrases posses and how can we identify them?

We will approach this problem from two perspectives: Knowledge Poor and Knowledge Rich. Knowledge Poor techniques rely on using shallow text processing which primarily utilizes the information about word and collocation frequencies. From the Knowledge Rich perspective, we hope to use many computational linguistic techniques to intelligently parse documents and rank words to discover meaningful phrases. We have compared these two approaches in selecting significant phrases, and found that they should be combined to augment each other. The knowledge poor approach is robust and fast, but the knowledge rich approach has the advantage of tackling phrases relevant to the contents more precisely.

2 Related Work

Automatic keyword generation is an important commercial application, and it is contingent on detection of content words and collocations in text. Amazon.com and their "Search Inside!" program was useful for structuring our approach. From the theory perspective, the most prominent paper on the topic was by Katz (Katz, 1996) who worked on creating

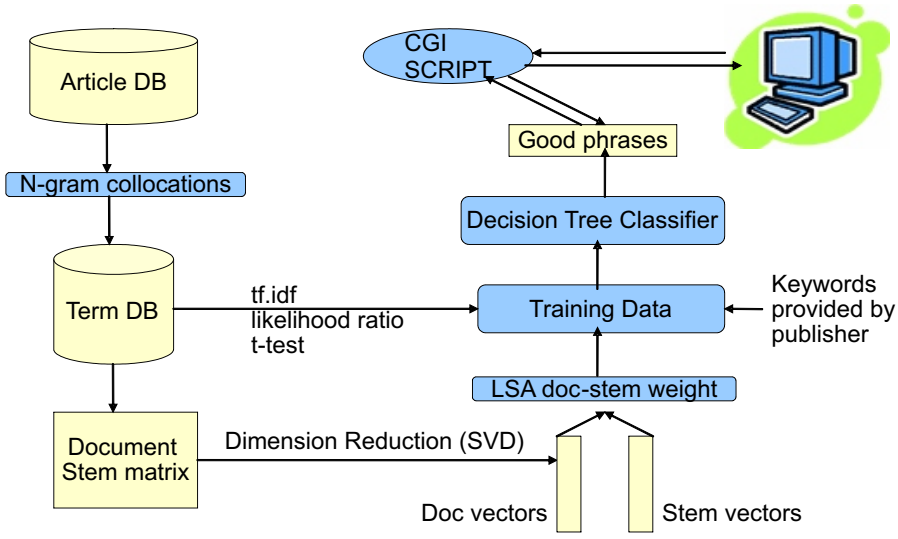

Figure 1: Knowledge-poor approach pipeline

relationships between text and topic words. One paper discusses using significant words for ontology extraction (Damle and Uren, 2005). Another approach found content-bearing words using serial clustering (Bookstein et al., 1995). We found work done on text summarization and topic tracking, which are somewhat related, more common. We considered this work because it dealt with deriving meaning from text and could be exploited for certain properties. We could weight significant phrases by considering if they occurred in summary sentences. Text summarization techniques focused on selecting sentences from the source that were representative of the whole. Many different approaches are considered, we found that Maximal Marginal Relevance was very effective (Carbonell and Goldstein, 1998). A good source for additional information on text summarization techniques is found in the SUMMAC conference (Mani et al., 1999).

3 Design Outline

3.1 Knowledge-Poor Approach

A scheme of the knowledge-poor approach pipeline is shown in figure 1. We collect unigrams, bigrams and trigrams satisfying the predefined noun-phrase part-of-speech patterns (table 1) and put them into a database along with their counts, document frequencies and statistical measures (t-test, likelihood ratio).

Also, we stem all the words in all documents, excluding stop-words, and store the frequency of every stem in every document. In addition, we store the total count and document frequency for each stem. We use this information to apply the Latent Semantic Analysis technique, as described in section 4.7.2, which gives us modified stem-document weights. Those weights, along with t-test, likelihood ratio, total and document frequency serve as features for the C4.5 decision tree training algorithm.

The class variable is 'yes' or 'no' depending on whether a collocation is a good keyword for a specified article. The training data comes from the publisher-provided keywords that are available in certain online newspapers (e.g. New York Times). Many of those keywords are proper nouns, so exact matches of those keywords with collocations found by our algorithm are rare. This is why we add even a collocation that partially matches some of the given keywords as a good keyword in the training data. Also, we treat document titles in the same way as publisher-provided keywords. This allows to do training even on a bigger corpus.

Finally, we provide a front-end CGI script allowing to run the pre-trained classifier on a specific articles and displaying the identified significant terms. This allows to easily compare them to the results of knowledge-rich approach and the publisher-provided keywords.

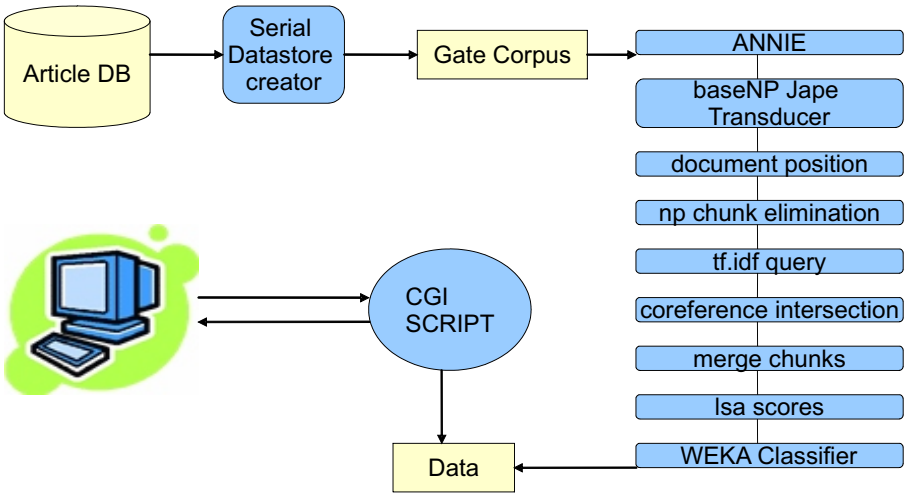

Figure 2: Knowledge-rich approach pipeline

3.2 Knowledge-Rich approach

Figure 2 shows the stages of the Knowledge Rich Pipeline. The guiding principle of the Knowledge Rich Pipeline is that significant phrases should act like the ”content words” of Katz (Katz, 1996). That is, there are certain properties of these words and phrases that denote the content they bear. Though we will analyze the claims later later, Katz argued that content words had a significant difference in document frequency and the actual number of documents it appeared in. Therefore, we look for words that would give high tf.idf scores. Next, Katz discusses the concept of ”burstiness”. Content words tend to occur in consecutive sentences, contributing to a burst behavior. One issue to be discussed later, it was clear that in many of our documents entities are not refered to by the same lexical phrase, such as ”President George Bush” and ”The President”. To handle this, we investigate ways of finding coreferential entities and merging attributes of the lexical counterparts together. Thus, coreference resolution became pivotal in order to detect bursts and possibly the document count for specific entites.

4 Knowledge Poor Approach Implementation

4.1 Collocation Detection Overview

The first step of knowledge-poor approach is detection of statistically improbable collocations over the whole corpus. We focused our work on bigrams and trigrams. The collocations interesting to us were noun phrases, so we used the part-of-speech patterns provided in (Manning and Schutze, 2003) to detect these collocations (table 1).

Tag Pattern	Example
A N	linear function
N N	regression coefficients
A A N	Gaussian random variable
A N N	cumulative distribution function
N A N	mean squared error
N N N	class probability function
N P N	degrees of freedom

Table 1: Patterns of part-of-speech tags used to filter out noun phrases by the knowledge-poor approach, courtesy (Manning and Schutze, 2003).

For ranking collocations obtained this way we chose to use the likelihood ratio and the t-test.

4.2 Log-likelihood Ratio

Log-likelihood ratio of an n-gram is calculated as a logarithm of the ratio between the probability of getting the observed counts under the hypothesis that the words in an n-gram are dependent on each other related to the probability of observed data under the hypothesis of independence. For trigrams, however, this calculation is complicated by the fact that there is more than one possible model of independence

between words in a trigram. Indeed, any two of them can be dependent but independent of the third one, or all three of them can be independent. Each of these hypotheses needs to be rejected for a truly meaningful collocation, because we are interested in trigrams where no word is accidental. We used the solution described in (McInnes, 2004) which is based on 'model fitting', i.e. assigning the score equal to the minimum of likelihood ratios of the four models to each particular trigram.

However, as we eventually found out, the likelihood ratio test alone was not sufficient because it sometimes gave high score to collocations that appeared less frequently than expected. In particular, those were trigrams composed of two strong bigrams, such as 'lot of course' or 'couple of dollars'. In this case, every hypothesis of partial independence of independence of one of these words from two others was rejected with high confidence. So, we needed another statistical test to filter out those terms and the t-test turned out to help.

4.3 The t-test

The t-test (Manning and Schutze, 2003) is calculated as follows:

$$t = \frac{\bar{x} - \mu}{\sqrt{\frac{s^2}{N}}} \tag{1}$$

where $\bar{x}$ is the observed frequency of trigram occurence, s^2 is the observed sample variance, N is the corpus size in words, μ is the expected number of occurences. The value of t-test indicates how likely the sample is to be drawn from a distribution with mean μ. E.g. for a trigram appearing c_{123} times $\bar{x} = \frac{c_{123}}{N}$, $s^2 = \frac{c_{123}}{N}(1 - \frac{c_{123}}{N}) \approx \frac{c_{123}}{N}$ (see section 2.1.9 of (Manning and Schutze, 2003)), $\mu = \frac{c_1 c_2 c_3}{N^3}$.

4.4 Likelihood Ratio Calculation for Bigrams

For a given bigram $w_1 w_2$ we count the number c_{12} of bigram occurences and numbers c_1 and c_2 of occurences of w_1 and w_2 respectively. Let B be the total number of bigrams, including those that do not match our part-of-speech requirements. Then the *contingency table* of this bigram is defined in the following way:

	w_2	$\neg w_2$
w_1	n_{11}	n_{12}
$\neg w_1$	n_{21}	n_{22}

where

$$\begin{pmatrix} n_{11} & n_{12} \\ n_{21} & n_{22} \end{pmatrix} = \begin{pmatrix} c_{12} & c_1 - c_{12} \\ c_2 - c_{12} & B - c_1 - c_2 + c_{12} \end{pmatrix}$$

Along with observed counts n_{ij} we will consider the expected counts m_{ij}, $i, j \in \{1, 2\}$ calculated based on the hypothesis of independence of w_1 and w_2. If we let $n_{ip} = \sum_{j=1,2} n_{ij}$ and $n_{pj} = \sum_{i=1,2} n_{ij}$, then $m_{ij} = \frac{n_{ip} n_{pj}}{B}$. The log-likelihood ratio of a bigram has then two equivalent (Moore, 2004) expressions:

$$G^2 = \log L(c_{12}, c_1, p) + \log L(c_2 - c_{12}, N - c_1, p) - \log L(c_{12}, c_1, p_1) - \log L(c_2 - c_{12}, N - c_1, p_2), \tag{2}$$

where $L(k, n, \chi) = \chi^k (1 - \chi)^{n-k}$, and

$$G^2 = 2 \sum_{i,j} n_{ij} \log(\frac{n_{ij}}{m_{ij}}). \tag{3}$$

We use the latter one because it is easily generalizable to the case of trigrams

4.5 Trigram Collocation Detection

The generalization of formula (3) for trigrams is

$$G^2 = 2 \sum_{i,j,k} n_{ijk} \log(\frac{n_{ijk}}{m_{ijk}}), \tag{4}$$

where n_{ijk} and m_{ijk} are the observed and expected values accordingly for the counts in table 5.

$i = 1$	$j = 1$	$k = 1$	$C(w_1, w_2, w_3)$
$i = 1$	$j = 1$	$k = 2$	$C(w_1, w_2, \neg w_3)$
$i = 1$	$j = 2$	$k = 1$	$C(w_1, \neg w_2, w_3)$
$i = 1$	$j = 2$	$k = 2$	$C(w_1, \neg w_2, \neg w_3)$
$i = 2$	$j = 1$	$k = 1$	$C(\neg w_1, w_2, w_3)$
$i = 2$	$j = 1$	$k = 2$	$C(\neg w_1, w_2, \neg w_3)$
$i = 2$	$j = 2$	$k = 1$	$C(\neg w_1, \neg w_2, w_3)$
$i = 2$	$j = 2$	$k = 2$	$C(\neg w_1, \neg w_2, \neg w_3)$

Table 5: Contingency table for trigrams

Calculation of expected counts for the model of complete independence of the three words is as follows (McInnes, 2004):

$$m_{ijk} = \frac{n_{ipp} n_{pjp} n_{ppk}}{n_{ppp}}, \tag{5}$$

Model	Expected count values
w_2 and w_3 dependent but independent of w_1	$m_{ijk} = n_{pjk}n_{ipp}/n_{ppp}$
w_1 and w_3 dependent but independent of w_2	$m_{ijk} = n_{ipk}n_{pjp}/n_{ppp}$
w_1 and w_2 dependent but independent of w_3	$m_{ijk} = n_{ijp}n_{ppk}/n_{ppp}$

Table 2: Formulas for calculating expected counts for trigrams (McInnes, 2004)

Bigram	LLR	c1	c2	c12
last year	20349.31	8874	17452	2519
last week	10953.55	8874	5245	1222
high school	10760.26	2343	6123	969
vice president	7240.635	570	2654	481
last month	6180.07	8874	4019	738
real estate	5555.987	1170	447	339
last season	4686.203	8874	6539	661
little bit	4139.569	2141	808	317
health care	3929.346	1412	1168	303
news conference	3815.63	1563	1273	304
executive director	3764.972	584	2264	280
law enforcement	3643.588	2602	403	260
locker room	3512.098	257	1689	220
free throw	3474.606	2718	316	240
powder machine	3425.675	293	674	200
general manager	3392.192	792	1729	260
field goals	3356.676	1949	2279	310
regular season	3262.687	719	6539	300
bird flu	3244.954	694	442	204
board member	2924.258	1944	3540	298
cell phones	2720.526	458	996	185
gold medal	2590.139	579	531	169
next week	2578.615	3360	5245	322
spring training	2557.995	1118	1006	200
same time	2481.963	2553	10182	338

Table 3: Bigrams for a 120MB corpus of Lydia news articles

where p denotes a summation over two possible values of the corresponding index. For the other three models m_{ijk} are calculated as shown in the table 2.

Then, for every trigram, the resulting score is the minimum of four values of (4) calculated for different models.

The way of computation of the n_{ijk} values for a trigram, however, is not obvious. We need to count

- the number c_{123} of occurences of the trigram;
- numbers c_{12}, c_{13}, c_{23} of occurences of every pair of words. c_{12} and c_{23} can be counted using the same hash map, and c_{13} has to be counted using a different hash map storing the number occurences for every pair of words with one word between them;
- numbers c_1, c_2, c_3 of occurences of every word of the trigram;
- total number T of trigrams which is rougly equal to the number of words.

Based on these values, for a particular trigram we can calculate

$$\begin{aligned} n_{111} &= c_{123} \\ n_{112} &= c_{12} - c_{123} \\ n_{121} &= c_{13} - c_{123} \\ n_{122} &= c_1 - c_{12} - n_{121} \\ n_{211} &= c_{23} - c_{123} \\ n_{212} &= c_2 - c_{12} - n_{211} \\ n_{221} &= c_3 - c_{13} - n_{211} \\ n_{222} &= T - c_1 - n_{211} - n_{212} - n_{221}. \end{aligned} \quad (6)$$

Trigram	LLR score	c1	c2	c3	count
millions of dollars	2252.286	403	128981	746	134
years in prison	2033.748	17452	114438	1026	184
past few years	1767.856	2416	2060	17452	40
lot of people	1748.485	3245	128981	7779	190
national championship game	1669.084	1796	1116	10779	16
thousands of dollars	1628.506	683	128981	746	69
next few years	1531.278	3360	2060	17452	30
school board member	1379.756	6123	1944	3540	40
mental health care	1364.802	242	1412	1168	14
hundreds of thousands	1312.636	477	128981	683	70
last few years	1214.283	8874	2060	17452	28
tens of thousands	1207.882	120	128981	683	82
number of people	1125.308	2361	128981	7779	70
thousands of people	1066.448	683	128981	7779	45
chief executive officer	1050.979	781	584	3806	69
group of people	1011.54	3302	128981	7779	35
possession of marijuana	952.4012	626	128981	225	51
life in prison	947.1451	2159	114438	1026	73
millions of people	928.5314	403	128981	7779	14
hundreds of people	925.0369	477	128981	7779	15
dozens of people	922.4656	663	128981	7779	11
next few days	915.6978	3360	2060	6032	25
3-point field goal	910.1331	489	1949	2279	63
next few weeks	894.8951	3360	2060	5245	24
time of year	854.1218	10182	128981	17452	92

Table 4: Trigrams for a 120MB corpus of Lydia news articles

4.6 Term Filtering

Of all bigrams and trigrams satisfying the noun phrase pattern we are only interested in those having a sufficiently high score. The problem is that we have multiple different parameters that can be used as score, such as term frequency in a particular document (tf), document frequency (df, the number of documents where the term occurs). First we attempted to identify a threshold for each of these parameters by hand, using the tables of χ^2 and t critical values. Table 7 indicates the percentage of terms removed according to these thresholds. As it shows, the t-test threshold was too high, and this turned likelihood ratio (G^2) useless in this case. Finally, we decided to treat likelihood ratio and t-test in the similar way as other features of a decision tree classifier, leaving threshold calculation to the training algorithm.

Another condition we used for terms filtering was stop words. For example, we removed collocations containing 'that', 'if', 'a.m.', 'p.m.', 'such', 'per', 'be' etc. This helped protect against the part-of-speech tagger inaccuracies and some limitations of the approach.

4.7 Determining Relevant Terms for an Article

4.7.1 Using tf.idf

One approach to detecting terms relevant to a particular article is the *tf.idf* weighting scheme (Manning and Schutze, 2003). If tf_{ij} is the number of occurences of the term t_i in the document d_j and df_{ij} is the number of documents that t_i occurs in, the weight of term t_i relative to document d_j is defined as

$$\mathrm{weight}(i,j) = \begin{cases} (1+\log(\mathrm{tf}_{ij}))\log\frac{N}{\mathrm{df}_i} & \text{if } \mathrm{tf}_{ij} \geq 1, \\ 0 & \text{if } \mathrm{tf}_{ij} = 0 \end{cases} \quad (7)$$

The problem with this scheme, however, is that it favors terms that only appear in one document and most times (for the Lydia corpus) these terms appear in the document only once. Clearly, these terms cannot be regarded as meaningful collocations (see the previous section).

However, the *tf.idf* scheme is very useful combined with other approaches that are described below. We include the *tf.idf* value into the set of features for classifier training.

4.7.2 Using Latent Semantic Analysis

The second approach we employ is to use Latent Semantic Analysis based on the technique called Singular Value Decomposition. The key idea of LSA is to reduce the dimensionality of the document-term matrix by choosing the best approximation of this matrix with a matrix of a lower rank, which is claimed to capture hidden semantic relations between terms and documents.

SVD works by decomposing document-term matrix as shown (see (Manning and Schutze, 2003)):

$$A_{t\times d} = T_{t\times n} S_{n\times n} (D_{d\times n})^T \tag{8}$$

where t is the number of terms, d is the number of documents and $n = \min(t, d)$ (we show dimension by subscript).

In our case, in the rows of document-term matrix we decided not to use all the terms (unigrams, bigrams and trigrams) obtained in previous steps, because the matrix would be incredibly large and sparse in this case. We decided to use single words instead, and furthermore, remove stop words from them and strip them to stems, so our A is essentially a *document-stem* matrix.

To choose documents and stems to participate in the SVD process we used the following rule: we chose n documents with the largest number of different stems in them (i.e. roughly largest documents), and among all the stems that happened to occur in these documents we chose n with the largest document frequency. Hardware and software available to us allow to handle SVD for an $n \times n$ matrix with $n = 10000$. The matrix $S_{n\times n}$ in (8) is diagonal and contains singular values of matrix A.

Dimensionality reduction is done by leaving only first k of these values (replacing others with zero) and recalculating A from (8). The resulting table of weights presumably takes into account co-occurence patterns, unlike the unprocessed A matrix. In the evaluation section, we present some results to illustrate this.

To handle more than n documents, we fold additional documents into our k-dimensional LSA space by using the property of SVD that $T_{t\times n}$ has orthogonal columns: $T^T_{n\times t} T_{t\times n} = I_{n\times n}$. Therefore, from (8) we have

$$T^T_{n\times t} A_{t\times d} = S_{n\times n} D^T_{n\times d} \tag{9}$$

So, if $x_{t\times 1}$ is a column-vector of stem frequencies of a document outside our n-document collection initially used for LSA, we can fold it in our n-dimensional LSA space as follows:

$$y_{n\times 1} = T^T_{n\times t} x_{t\times 1} \tag{10}$$

Further, we can discard the last $n - k$ components of y, thus obtaining the vector $\tilde{y}$ with reduced dimension, and calculate the 'smoothed' vector $\tilde{x}$ of weights of stems in our document by multiplying (10) by $T_{t\times n}$ on the left:

$$\tilde{x}_{t\times 1} = T_{t\times k} \tilde{y}_{k\times 1} = (T_{t\times k} T^T_{k\times t}) x \tag{11}$$

where $T_{t\times k}$ is the matrix of the first k columns of $T_{t\times n}$. In practice, for online operation, it makes sense to store the k-dimensional vector $\tilde{y}$ for each document and perform only the part of the $T_{t\times k}\tilde{y}_{k\times 1}$ matrix multiplication relevant to the stems we are interested in.

There is a question of what information should be used as the initial document-stem matrix for LSA – stem frequencies in documents or their *tf.idf* values. One more issue that comes up here is whether we need to normalize weights of stems in each document to make them (or their squares) to sum up to one. We address these question in the evaluation section 5.2.

Another question that arises is how to obtain a weight of a term (bigram or trigram) knowing weights of every particular word in it. The possible answers are: sum, product, or minimum value – to restrict spurious collocations with only one significant word. We considered these possibilities and decided to include weights of all the stems into the training feature set. We did it in such a way that the weight of the last stem is represented by the same feature for unigrams, bigrams and trigrams, because the last word in a phrase is frequently the head word. The weight of the last but one stem corresponds to another feature in case of bigrams and trigrams.

4.7.3 Machine Learning

We used key phrases provided by New York Times to identify the value of the class attribute for machine learning: whether or not a term is important for an article. We considered a term to be a keyword if it differs in no more than one word from

Top Stems after LSA			Top Stems by Frequency		
Stem	Weight	Freq	Stem	Weight	Freq
women	22.475	4	femal	3.494	12
studi	20.812	10	stress	1.479	12
research	10.674	3	studi	20.812	10
heart	10.281	5	rat	0.726	10
diseas	9.790	9	diseas	9.790	9
risk	8.674	2	respons	4.607	8
human	7.936	1	male	2.143	7
person	7.707	1	men	6.943	6
anim	7.468	1	differ	5.527	6
men	6.943	6	effect	4.491	6
cancer	6.470	1	social	2.058	6
health	5.821	2	isol	0.268	6
differ	5.527	6	inflammatori	0.129	5
report	4.914	1	immun	1.000	5
respons	4.607	8	heart	10.281	5
system	4.503	2	bodi	3.594	4
effect	4.491	6	women	22.475	4
new	4.199	1	offspr	0.360	4

Table 6: Top-ranked stems in a document after applying LSA and before (using stem frequency)

a subset of some of the key phrases provided by the publisher. To predict the class attribute value, we used the following attributes:

- count - total number of occurences of the term in the corpus
- t-test
- likelihood ratio
- document frequency (number of documents containing the term)
- number of words in the term (one, two or three)
- frequency of the term in the document
- tf.idf
- 'frequency lift' - the ratio of the relative frequency of the term in the document to its relative frequency in the whole corpus
- $LSAweight_{1,2,3}$ - weights of stems of individual words according to LSA

For machine learning we used the C4.5 decision tree learning algorithm. We ranked significant phrases in the output based on the class attribute and confidence value provided by the classifier.

Number of Terms	Why removed
146 (0%)	stop-words
3215 (1%)	stop-words, t-test
125117 (48%)	t-test
4880 (2%)	stop-words, t-test, G^2
101781 (39%)	t-test, G^2

Table 7: Percentage of terms filtered out according to various criteria with required confidence for G^2 and t-test of 0.999.

5 Evaluation

5.1 Bigrams and Trigrams Scoring and Filtering

We have run bigram and trigram noun-phrase detection and ranking on a 120MB news to ascertain the relevance of our collocation scores. The results are shown in tables 3 and 4.

5.2 Ranking Terms Relative to a Document

Tables 8 and 9 show top-ranked terms in a document according to two schemes: *tf.idf* and weight obtained by summing LSA weights of stems of words in a collocation without normalizing weights prior to running SVD. The general character of terms favored by LSA suggests that 'smoothing' done by LSA on stem frequencies favors stems that are frequent over the whole corpus rather than specific

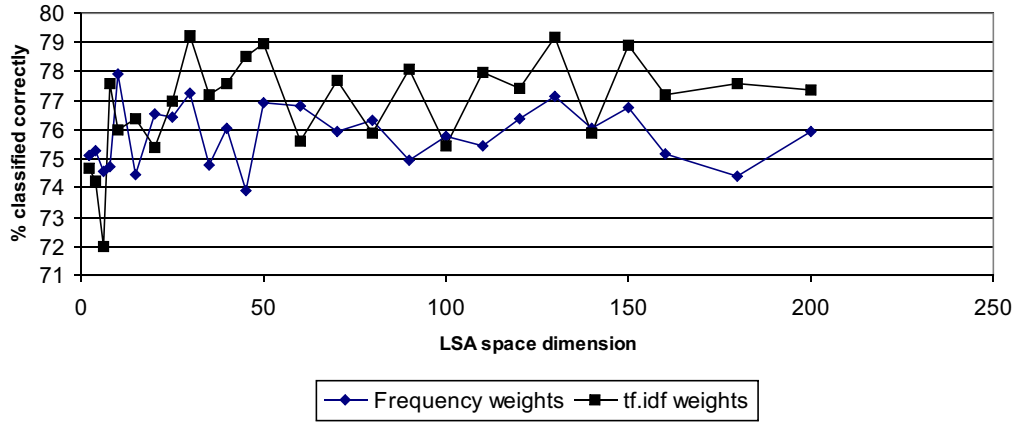

Figure 3: The use of term frequences compared to tf.idf in calculating LSA weights, dependent on the LSA space dimension

stems. This is why all the subsequent experiments we run with mandatory normalization of weights in the initial document-stem table.

Also we experimented with different dimensions of the LSA space. Figure 3 shows the percentage of correctly classified terms obtained via stratified cross-validation, when varying number of dimensions of the LSA space. We can see that using tf.idf as weights for LSA is generally better than using frequencies. We also see that choosing the best dimension of LSA space is not an easy task and more research is needed to come up with an algorithm of doing this.

Maximum precision and recall were achieved for a 30-dimensional LSA space and they both equal 79.2%.

We have set up a simple information retrieval system performing search based on collocations and listing significant phrases generated by both (knowledge-rich and knowledge-poor) approaches. While browsing this system, many times we were able to tell article topic by looking at the top significant phrases.

Phrase	Score	TF	DF
inflammatory response	26.5	5	5
sex differences	20.6	3	7
rat	19.91	10	309
inflammatory	18.43	5	110
female rats	17.19	2	5
isolation	16.48	6	351
physical stress	16.4	2	8
social isolation	16.4	2	8
offspring	16.01	4	157
stress	15.68	10	1113
immunity	14.03	5	593
males	13.74	5	665

Table 8: Terms most relevant to a document according to tf.idf

6 Knowledge Rich Approach Implementation

The software to run the KR pipeline is written as GATE processing resources. Our goal here was to be able to utilize other NLP tools that can be run on the platform. We also used the GATE API in order to create stand-alone applications to create corpora. To run our pipeline, first, documents must be loaded into the articles database. Next, the documents must be processed using ANNIE (with defaults) with additional processing resources Noun Chunking and Pronoumial Coreference Resolution. This is followed by the baseNP Jape Transducer. Next, the Knowledge Rich approach plugin (KR_SP) is run. After this, the Knowledge Rich Training phase (KR_Train) is executed and the pipeline is finished.

Phrase	Score	TF	DF
new study	26.01	1	329
women	23.48	4	9195
study	21.81	10	5779
heart disease	21.07	3	485
risk of heart	19.95	1	210
disease in men	17.73	1	2
differences in heart	16.81	1	4
research team	14.48	1	66
research	11.67	3	6929
high risk	11.44	1	111
heart	11.28	5	7591
evidence of human	10.91	1	6

Table 9: Terms most relevant to a document according to sum of LSA-derived weights

6.1 The DocumentsFromDB GATE Application

In order access more documents, it was clear that we needed a processing resource that was able to load documents from the Lydia databases. Though our initial folder based method was successful, it is more likely that data will come from a database. Conveniently, Java is well suited for this task. We note that in order to run this software on any machine that it is essential to keep the proper database connector within the source of the processing resource. We also chose to store our data as a serial datastore. The draw back of this is that it is not entirely clear how GATE handles serial datastores. It seems that for the most part, a large serial corpus that has reasonably sized documents exists quite contently in the environment. We saw that the benefit of using serial datastores allowed us to use corpora created on any platform.

6.2 Use of ANNIE

We decided to use ANNIE because of its interoperability with GATE as well as it's accessibility for programming when using the API. All ANNIE constants are saved and thus, manipulating annotations is considerably easier. Also, the addition of Noun Chunking and the Pronomial Coreference component was also compelling factor. Using another Coreference or Chunking resource may have required additional processing, which would be a detriment as seen below. We did find that using the Coreference module provided by ANNIE was very useful, but only created coreferences for the orthographic and named entity transducer. We were able to reverse engineer the cryptic document feature "MatchedAnnots" and apply it to any annotation of our choosing.

6.3 The BaseNP Jape Transducer

This transducer was created to find the baseNP of the noun phrases. It also identifies the head noun. Our goal was to use this information for the next step. It is true that another processing resource could have been developed to create these annotations. It is, however, very unnatural and much more intuitive to express in terms of a context-free grammar. We created a multi-phase transducer that runs over the "NounChunk" annotation and labels the corresponding internal "Token" annotations with their position relative to the annotation. A second and third phase used these annotations in order to identify the baseNP and head noun within the "NounChunk". The baseNP and head noun can be retrieved by simply querying the document's annotation set and retrieving the baseNP and head nouns that are contained within the bounds of the desired noun chunk.

6.4 Knowledge Rich Phase

After completing the first and second project submissions, we agreed that though debugging was easier with multiple processing resources, it was difficult to explain and maintain the proper ordering of the plugins. For the last few weeks, the various plugins were incorporated into a large plugin, KR_SP. The Knowledge Rich approach to Keyphrase detection is composed of many different stages that determine the essential features. We use the "NounChunk" as the potential set for good significant phrases. We then apply the following procedures.

6.4.1 Document position

GATE implements Annotations as HashMaps that leads to one specific problem: annotations are not returned in document position order. Initially, we were very confused by this and realized soon that in many instances document ordering was crucial. The class AnnotationInOrder was created for this purpose. We first extract all sentences that ANNIE annotates and put them into an instance of AnnotationInOrder. Each sentence is labeled with its position in

the document. The NounChunks are then retrieved and labeled with their corresponding sentence position.

6.4.2 tf.idf score

An interesting problem arises with using tf.idf and noun chunks. We note that a NounChunk can be of arbitrary size. It is necessary, however, for tf.idf to be a fixed size n-gram. To solve this problem, we look at the n-grams contained in the noun chunk. For this project, we initially wrote a separate tf.idf module for GATE. We found this to be difficult to run because all plugins are applied to document before going to the next. The issue here is that n-grams must be collocated, put into a corpus pipeline, run, and then the tf.idf computation can run. Thus, we decided to use the tf.idf information stored in the Knowledge Poor database since the same documents were used to make comparisons. And thus, each noun chunk is broken into corresponding unigram, bigram and trigrams. The n-gram with the greatest tf.idf score is used to denote the uniqueness of the annotation.

6.4.3 Coreference counts

In order to properly gauge how many times an entity is referred in a document, we must take into account pronominal coreferences. That is, if ”President Bush” gave a lecture, ”he” may have amazed the crowd. It is important that we remember that the entity corresponding to ”President Bush” was also referenced by ”he”. In order to facilitate this, we ran the ANNIE Pronomial Coreference module. Interestingly, this module only specifically updates Annotations created by the previous Orthomatcher and NE transducer. As said earlier, we desire to use the noun chunks annotations as the basis of good key phrases. Thus, we were able to use the ”MatchesAnnot” feature produced by the coreference module and then find the annotations with coreference information. Each annotation that was referred to by the feature of another was processed by retrieving the referred annotation and finding its document position. This allowed us to find the intersecting noun chunks and add a feature called ”coconuts”, the number of pronominal counts which is updated with each coreference found.

6.4.4 Merge chunks

The last step is a combination of finding coreferences of nouns other than prounouns and collecting data. The output of this step creates the ”CoreferenceChain” annotation. We used the techniques employed by Morton for proper and common noun resolution. This worked surprisingly well, but this fact may be explained by the corpus we were using.

6.5 Training phase

After the noun chunks have been merged into noun chunks representing the (hopefully) unique set of entities that exist in the text, the training phase begins. This PR has three functionalities. First, we apply an LSA score to the coreference chains, as done in the Knowledge Poor approach. Next, we define the attributes for machine learning. Here, we have made a few detours from a linear path to implementation from our theoretical concept of significant phrases. Since both approaches use the same information for tf.idf and LSA, it is necessary that we take into account that the database is not concerned with proper nouns. The motivation for this was the fact that proper noun detection is already done in Lydia. We add an attribute to the training data such that it indicates if a particular word is a proper noun. This will prevent ”muddying” other data since the LSA and tf.idf scores will be incredibly low. The training stage is also important since what we are doing for this project is inherently different the approaches of Katz and KEA. Here, we note that these authors are focusing on the scientific domain and some heuristics they employ may not be suitable for what we are doing. The attributes that are used for learning and prediction are as follows:

- Is it a proper nouns
- tf.idf
- LSA score
- start
- span burst
- Is it a significant phrase

Like the Knowledge Rich approach, training has been done by extracting keywords and words of the

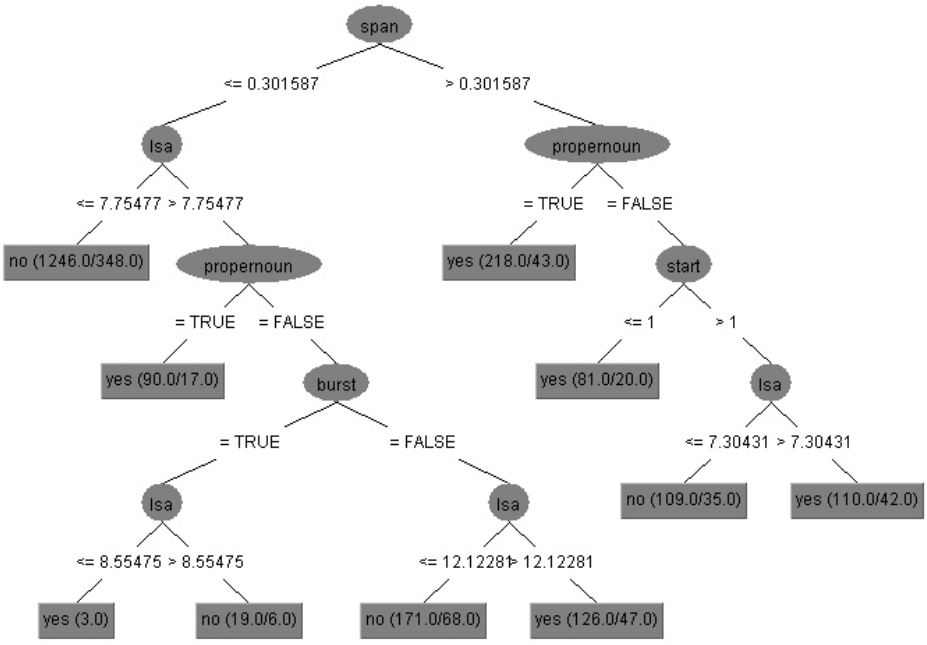

Figure 4: A simplified (pruned) tree generated for Knowledge Rich training data

title. These words are scanned through the coreference chain annotations and if it is coreferent with an annotation, we desiginate it as a significant phrase and put them in our training data. Afterwards, we go through the rest of the annotations and we allow the same number of non-significant phrases as significant phrases be added to the training data.

It was mentioned that LSA is used in this step. However, the Knowledge Poor uses a stemmer! At this juncture we also instantiate a stemmer (not actually a GATE plugin but an actual class) to facilitate these transactions. Prior to stemming, database about the LSA information of the stems is included in the After stemming, a look up is done and the LSA value for the stem is added. To detect proper nouns, we simply look for proper noun features that were produced in ANNIE stage of the pipeline.

Like the Knowledge Poor, a WEKA C4.5 decision tree classifier is trained from the training data. WEKA has nice properties because it is not only written in Java but provides fairly robust algorithms for a wide spectrum of learning algorithms. We constructed a script that tested several of these classifiers including neural networks (Modified Perceptron), Bayes, etc. We decided on the C4.5 decision tree because of the quality and the ability to gain intuition about how predictions are made. The issue with neural networks, which did produce competitive results to the C4.5 classifier, was that we could not exactly understand what was happening and opted to not be in a position to trust the black box. Because of the our algorithm choice, we were able to notice that denoting proper nouns was important. Needless to say, the data produced at this step are ARFF files containing instances with corresponding attributes. To detect significant phrases, we run the C4.5 classifier with prediction enabled. The words are ranked first by if they are significant phrases and then secondly on their confidence levels.

Lastly, the Knowledge Rich Training processing resource also produces files that contain values that be predicted by our classifier. That is, if the document has no training data associated with it, the plugin will create an ARFF file where the value for ”Is a significant phrase” is denoted with a ?. Therefore, we have consolidated another plugin one. We did not include this in the previous phase because many modifications were made to this plugin and made us able to hot swap different verisions.

7 Discussion

The Knowledge Rich approach was ambitious, but maybe too much so. The core issue is that the approach depended on discovering coreferential entities in the text, which is by far not a trivial task by any means. The training data may have contributed to performance problems of identifying keywords. To generate training data, it would have been infeasible for us to annotate significant phrases in numerous documents. As discussed above, we used keywords generated by the document source. Keywords have an important facet: they are a characterization and not a carbon copy of the source material. The importance of this relation is that a key word about the unrest in France may generate the keyword ”Europe”. This is necessary for some indexing algorithms, but causes problems for generating . Future work could investigate into generating keyword trees where the leaves are significant phrases from source materials. For our purposes, I believe that this method may have been adverse. Looking at the training data, we noted that many of the significant phrases in the data did not actually have bursts. We have noticed that in the training set, 321 instances of Significant Phrases had bursts versus 935 without bursts.

This engenders an interesting question about the qualities of significant phrases and content words. Our approach may be flawed and not be capturing the essence of these structures in the training data. On the other hand, let us consider Katz's material. We note that Katz studied bigrams and trigrams that are more or less atomic. What we mean by this is that many of the ngrams studied by Katz were scientific terms that persevre their lexical representaiton. Authors that use these terms do not interchange ”fs” for ”frequency spectrum”. The entites in the news articles that we examined have many different lexical representations and incur the necessity of doing coreference resolution for many types of noun phrases. We also must admit that our approach is far from exhaustive of catching each coreference. For example, if we had ”the Red Sox” and ”the Beantown ball club”, this would not be detected by our algorithms. Therefore, I argue that before Katz's claim can be fully substantiated for a generic domain, careful investigation of entity resolution in text must confirm that significant phrases (corresponding to one or more of these entities) do indeed occur as bursts. In fact, it may be the case that bursts may exist but not necessarily consecutive in nature. Katz used the metric of sentences to quantify bursts. Perhaps terms that are significantly closer in proximity to other instances of themselves relative to other terms (regardless if the burst is a consecutive sentence) should be consider. One last suggestions is perhaps bursts should be relagated to the paragraph level. If burst patterns are indeed hard to qualify, domain specific characterization of significant phrases will be necessary.

Another key issue was performance for Knowledge Rich. Some of this may be expediated with more carefully planned algorithms and execution environments. Working with GATE, as noted in earlier project reports, caused many frustration. It is clear that pronominal coreference is not implemented as specified in the documentation. Also, for Morton's coreference algorithms, it was necessary for us to have annotations in a particular order. Annotations are hash maps and that chronological ordering is not a guarantee for these sets. This is not to dismiss many of the positive characteristics of GATE which fosters great NLP applications. Rather, to be competitive with knowledge poor algorithms, it is necessary for Knowledge Rich approach to be fast and robust. Given more time, optimizations would be made for Morton's coreference strategy for proper nouns. The algorithmic issue is determining if a new string instance is a substring of set of mutually substring distinct strings. We choose a rather naive but reliable implementation for this. Another important hit came from the decision to find the best ngram tf.idf score for a noun phrase. A noun phrase of length 5 has 5 unigrams + 4 bigrams + 3 trigrams which is 12 database queries. We could prevent his by looking solely at head nouns, but this would not take into account that adjective modifications could possibly highlight statistically significant phrases. However, being able to have 12 tf.idf scores can give insight into the words that compose of the noun phrase.

8 Conclusion

Approaching the problem of key phrase identification from two perspectives, we found out that it is difficult in the generic domain. The statistical approach finds phrases that are mostly relevant to the document topic, and it is robust and fast. The natural language processing-based approach provides phrases of a bigger variance, which results in smaller precision, but from the news reader's point of view knowledge rich keywords are more natural and sometimes very precisely catch the topic of the document. So, the ultimate goal of the future research in this direction is in careful blending of statistical methods, theory and deep parsing. Looking at our results as a baseline, it is clear that a combination of these methods can provide an advanced solution to the keyword detection problem.

References

A. Bookstein, S. T. Klein, and T. Raita. 1995. Detecting content-bearing words by serial clusteringextended abstract. In *SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 319–327, New York, NY, USA. ACM Press.

Jaime Carbonell and Jade Goldstein. 1998. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336, New York, NY, USA. ACM Press.

Dileep Damle and Victoria Uren. 2005. Extracting significant words from corpora for ontology extraction. In *K-CAP '05: Proceedings of the 3rd international conference on Knowledge capture*, pages 187–188, New York, NY, USA. ACM Press.

S. M. Katz. 1996. Distribution of Content Words and Phrases in Text and Language Modelling. Natural Language Engineering, 2: 15-59.

Inderjeet Mani, David House, Gary Klein, Lynette Hirschman, Therese Firmin, and Beth Sundheim. 1999. The tipster summac text summarization evaluation. In *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, pages 77–85, Morristown, NJ, USA. Association for Computational Linguistics.

Christopher D. Manning and Hinrich Schutze. 2003. *Foundations of Statistical Natural Language Processing*. The MIT Press.

Bridget T. McInnes. 2004. Extending the Log Likelihood Measure to Improve Collocation Identification. Master's thesis.

Robert C. Moore. 2004. On Log-likelihood-Ratios and the Significance of Rare Events. Conference on Empirical Methods in Natural Language Processing.