Fast Mining of Interesting Phrases from Subsets of Text Corpora

Fast Mining of Interesting Phrases from Subsets of Text Corpora

Deepak P   Atreyee Dey   Debapriyo Majumdar
IBM Research - India
Bangalore, India.
{deepak.s.p,atreyee.dey,debapriyo}@in.ibm.com

ABSTRACT

We address the problem of mining interesting phrases from subsets of a text corpus where the subset is specified using a set of features such as keywords that form a query. Previous algorithms for the problem have proposed solutions that involve sifting through a phrase dictionary based index or a document-based index where the solution is linear in either the phrase dictionary size or the size of the document subset. We propose the usage of an independence assumption between query keywords given the top correlated phrases, wherein the pre-processing could be reduced to discovering phrases from among the top phrases per each feature in the query. We then outline an indexing mechanism where per-keyword phrase lists are stored either in disk or memory, so that popular aggregation algorithms such as No Random Access and Sort-merge Join may be adapted to do the scoring at real-time to identify the top interesting phrases. Though such an approach is expected to be approximate, we empirically illustrate that very high accuracies (of over 90%) are achieved against the results of exact algorithms. Due to the simplified list-aggregation, we are also able to provide response times that are orders of magnitude better than state-of-the-art algorithms. Interestingly, our disk-based approach outperforms the in-memory baselines by up to hundred times and sometimes more, confirming the superiority of the proposed method.

1. INTRODUCTION

From a generic corpus of text documents such as collections of documents from social media or news sources that encompass a variety of topics, it is often necessary to drill-down to topic-specific subsets of interest. Such drill down is usually accomplished by using keyword queries (e.g., set of text documents that contain one of a few keywords), and less commonly, using metadata queries (e.g., set of news documents tagged with a specific topic). Analysts are often interested in getting a *feel* of the topic-specific corpus using information such as characteristic keywords or phrases. A common tool that is employed to aid such analysis is to derive a small set of characteristic tags or words that best describe the collection and visualize them using tag-clouds or sets of *hot* keywords[1]. While word and tag information do provide useful information that characterize the text document subset, variable-length phrases mined from the body of the text documents in the chosen collection could provide richer information since phrase-level mining could potentially unearth names of people and organizations, marketing slogans and much more[2]. This observation sparked interest in *mining interesting phrases* from *dynamically derived document subsets*, i.e., the problem of real-time discovery of interesting phrases from document datasets that are chosen by a user based on keyword-queries or metadata facets through an IR-like query interface.

Notion of Interestingness: While interestingness may be thought of as directly linked to the abundance of the phrase in the chosen document sub-collection, a purely frequency based scoring is likely to score phrases composed of stopwords highly. However, this is easily overcome by normalizing the score by the corpus-wide frequency of the phrase, leading to an interestingness measure as follows:

$$I_{\mathcal{D}}(p, \mathcal{D}') = \frac{freq(p, \mathcal{D}')}{freq(p, \mathcal{D})} \qquad (1)$$

where $\mathcal{D}$ and $\mathcal{D}'$ are meant to refer to the entire corpus of documents and the chosen sub-collection respectively. The interestingness $I_{\mathcal{D}}(.,.)$ of the phrase p is defined as the frequency of the phrase in the chosen sub-collection (i.e., $freq(p, \mathcal{D}')$), normalized by its frequency in the entire corpus of documents denoted by $freq(p, \mathcal{D})$. Though there are alternative formulations for interestingness such as pointwise mutual information[19], recent work has focused on scoring phrases using the above normalization-based model[2, 8]. The normalization using global frequency helps in de-prioritizing phrases that are abundant all over $\mathcal{D}$, and not just in $\mathcal{D}'$. For example, while choosing a subset of documents related to database research from among a more general corpus comprising of computer science research papers, phrases such as *complexity analysis* would be de-prioritized by the normalization in favor of database specific ones such as *query optimization* despite the former kind of phrases being common in database papers as well.

Selecting the Sub-Collection: Though previous work on interesting phrase mining allow for $\mathcal{D}'$ to be any arbitrary subset of the corpus $\mathcal{D}$ (e.g.,[2, 8]) in principle, it is not very useful to select subsets of unrelated documents from $\mathcal{D}$. Thus, for all practical purposes, $\mathcal{D}'$ is formed by select-

10.5441/002/edbt.2014.18

Table 1: Methods for Selecting Sub-Collections

Feature	Operator	References
Keywords	AND/OR	[15, 2, 8]
Metadata Facet	AND	[15, 2]

ing subsets of documents that have a common feature such as a word or a metadata value (e.g., tag, topic, date etc.); followed by aggregating such collections across multiple features using *and* or *or* operators to form a single collection. Infact, all previous work on mining interesting phrases have evaluated their techniques on sub-collections that were constructed using the above model as illustrated in Table 1. For example, *venue:sigmod* and *year:1997* represent distinct sub-collections that satisfy the indicated metadata features, and these could be aggregated using an intersection (AND) to form a single collection of *sigmod* papers from *1997*.

Our Contributions: We address the problem of discovering interesting phrases from document sub-collections that are formed by an aggregation of sub-collections each defined by the inclusion of a specific keyword or metadata facet. Though previous techniques have used arbitrary sub-collections constructed using such a framework in their empirical evaluation, their techniques per se do not exploit the property. Thus, such techniques either start their search from a global set of pre-compiled interesting phrases to narrow down to the phrases that are interesting for the sub-collection, or start from the chosen set of documents $\mathcal{D}'$ and mine for phrases therein. The former approach obviously leads to a worst-case linear complexity of $\mathcal{O}(|\mathcal{P}|)$ where $\mathcal{P}$ refers to the global set of interesting phrases (which is typically many times the number of documents in the corpus), whereas the latter leads to a worst case complexity that is linear in the size of the chosen document sub-collection (i.e., $|\mathcal{D}'|$). Clearly, either of these are very expensive and are impractical if we need to achieve real-time responses. We propose indexing globally interesting phrases on the keywords (or facets), so that upon a query where the document collection $\mathcal{D}'$ is defined by $'a'\ OR\ 'b'$, the phrase lists corresponding to a and b could be accessed and interesting phrases could be mined therein. This paradigm of starting the search process from the features that defines the sub-collection leads to drastic improvements in response times since the number of features that define a specific sub-collection $\mathcal{D}'$ may be very few; for example, web search queries typically comprise of 2-3 keywords[16]. Our contributions are the following:

- We outline the assumption of conditional independence of the query keywords given phrases that are deemed to be interesting for a sub-collection (defined on the query keywords), and motivate why it is expected to hold in most cases.
- Towards harnessing the independence assumption in phrase scoring, we propose keyword-specific phrase list indexes to hold pre-computed conditional probabilities between words and phrases. We develop a technique that leverages the conditional independence assumption to mine interesting phrases from sub-collections using such keyword-specific phrase indexes.
- Through an extensive empirical evaluation on real-world datasets, we analyze the accuracy and establish the superiority of our technique over state of the art techniques for interesting phrase mining.

Table 2: Notations

Notation	Meaning
$\mathcal{D}$	Corpus of Documents
$\mathcal{D}'$	Selected subset of Documents
Q	Query used for selecting $\mathcal{D}'$ from $\mathcal{D}$
W	Set of features from $\mathcal{D}$
$\mathcal{P}$	Set of phrases from $\mathcal{D}$ to be considered for Interesting Phrase Mining
d	Any document from $\mathcal{D}$
p	Any phrase from $\mathcal{P}$
w	Any feature from $\mathcal{W}$

Notations: To help the narrative, we now introduce some notations in Table 2 that will be used in the rest of the paper. While $\mathcal{D}$ and $\mathcal{D}'$ have already been introduced to stand for the document corpus and the selected sub-collection respectively, W and $\mathcal{P}$ will be henceforth used to refer to the set of features (words and metadata facets) and phrases among documents in $\mathcal{D}$ respectively. Of these, $\mathcal{P}$, as is the case in many previous works such as [2], is not meant to include all phrases, but, only those word n-grams of up to 6 words which occur in more than a pre-specified number (usually, 5 or 10) of documents in $\mathcal{D}$. Since our method makes use of the features that are used to define the construction of $\mathcal{D}'$ from $\mathcal{D}$, we introduce the notation Q to refer to the set of features (e.g., keywords etc.) that implicitly specify $\mathcal{D}'$.

2. PREVIOUS ALGORITHMS

Previous approaches addressing the same problem have used list-based indexes that are common in techniques for Information Retrieval. Table 3 provides a summary of the three existing techniques particularly targeted at the problem of mining interesting phrases from sub-collections. All of these use list-based indexes, but differ in the construction of the indexes. In [15], the index comprises of $\mathcal{P}$ lists, with the i^{th} list comprising of information on the documents that contain the i^{th} phrase; these lists are ordered in the decreasing order of cardinalities, with the first list comprising of information for the most abundant phrase. Upon selection of $\mathcal{D}'$, the interestingness of a phrase p may be evaluated by intersecting the list corresponding to p with $\mathcal{D}'$. However, the technique works in two-phases, and the first phase simply chooses to ignore lists that have lengths lesser than the intersection cardinality of an already seen phrase. The second phase scores the phrases using a normalization-based interestingness score. It may be noted that, due to the normalization in the score computation, some phrases that have been discarded due to not having a long enough list may have a higher interestingness than those chosen for consideration from the first phase; this disconnect between the first-phase filtering and second-phase scoring leads to an approximation in the results. The techniques in [2] and [8] both employ document specific forward lists in their indexes. Thus, there is a list for every document in $\mathcal{D}$ that comprises

Table 3: Techniques for Interesting Phrase Mining

Technique	#Lists in Index	Length of Lists	#Lists Accessed	Approximate Scoring?
Simitsis et. al.[15]	$\mathcal{P}$, one list per p	#Docs containing p	upto $\mathcal{P}$	Yes
Bedathur et. al.[2]	$\mathcal{D}$, one list per d	(Phrases in d) $\cap\ \mathcal{P}$	$\mathcal{D}'$	No
Gao & Michel[8]	$\mathcal{D}$, one list per d	(Phrases in d) $\cap\ \mathcal{P}$	$\mathcal{D}'$	No
Our Technique	$\mathcal{W}$, one list per word	Upto (Phrases in $\mathcal{D}(w)$) $\cap\ \|\mathcal{P}\|$	$\|Q\|$, #words in Q	Yes

of the list of phrases from $\mathcal{P}$ that appear in the document. Upon identification of a sub-collection $\mathcal{D}'$, the lists for each document in $\mathcal{D}'$ is inspected, and merge-joined[1] so that the phrase frequency information may be obtained and scored to identify interesting phrases. The work that proposed the forward index framework [2] lists several optimizations that could be done to speed-up the mining process; for example, the fact that the presence of a phrase in a document implies the presence of its prefix can be leveraged to reduce the set of phrases that get explicitly stored in the forward index. [8] proposes clever organization of the forward lists as well as pruning strategies to improve upon the previous work. As an example, the common subsequence in the phrases ABC and CDE (considering alphabets as tokens/words) that appear in the same forward list could be indexed such that the common subsequence need not be stored twice. For both [2] and [8], the method needs to access each of the $\mathcal{D}'$ lists.

3. PROBLEM DEFINITION

We now describe the problem formally. Given a static corpus of $\mathcal{D}$ documents, our problem is to identify the top-k interesting phrases from dynamically selected document collections. Each such dynamically specified document collection, denoted by $\mathcal{D}'$, is specified by the user using a query $Q = [\{q_1, \ldots, q_k\}, O]$ where q_is represent features such as keywords or metadata facets and O denotes an operator used to aggregate feature-specific document collections. The construction of $\mathcal{D}'$ is formally specified as:

$$\mathcal{D}'\ for\ [\mathcal{D}, Q] = \begin{cases} \bigcup_i docs(\mathcal{D}, q_i) & \text{if } O = OR \\ \bigcap_i docs(\mathcal{D}, q_i) & \text{if } O = AND \end{cases} \tag{2}$$

$docs(\mathcal{D}, q_i)$ above denotes the subset of documents in $\mathcal{D}$ that have the feature q_i. Given a specification of $\mathcal{D}'$, the desired result is the set of the k *most interesting phrases* as determined by Equation 1. Formally,

$$R(\mathcal{D}, \mathcal{D}', k) = \underset{P \subseteq \mathcal{P}\ \wedge\ |P|=k}{argmax} \sum_{p \in P} I_{\mathcal{D}}(p, \mathcal{D}') \tag{3}$$

Informally, the result set $R(\mathcal{D}, \mathcal{D}', k)$ comprises of the k phrases from $\mathcal{P}$ that have the highest scores according to the interestingness measure, $I_{\mathcal{D}}(., \mathcal{D}')$. Though we would ideally like to compute the exact set of k phrases that have the highest scores, the approach that we present in the next section discovers a close approximation of $R(\mathcal{D}, \mathcal{D}', k)$ by harnessing certain assumptions that help achieve massive gains in response times.

[1]http://en.wikipedia.org/wiki/Sort-merge_join

4. OUR APPROACH

In this section, we present our approach for mining interesting phrases from document sub-collections. Firstly, we present our phrase scoring formulation and the conditional independence assumption that we make use of, in our method. Secondly, we describe the construction of the list based indexes that our approaches work on. Thirdly, we outline the algorithms that use the independence assumption and the indexes to identify the top-k interesting phrases; these are modeled on the threshold algorithm family initially proposed for middleware[7]. Lastly, we analyze the computational complexity of our approach in a lead up to the section on empirical evaluation.

4.1 Phrase Scoring under Conditional Query Word Independence

At the core of any top-k algorithm is the scoring mechanism for candidates. Here we outline the scoring mechanism that we use, in our approach for interesting phrase mining. We design a scoring function where the score of a phrase p is computed as the probability of occurence of the phrase in the chosen sub-collection $\mathcal{D}'$ normalized by its probability of occurence in the entire corpus (i.e., $\mathcal{D}$). This is almost identical in construction to the standard formulation of the interestingness measure in Equation 1. Notationally,

$$\mathcal{S}_{\mathcal{D}}(p, Q) = \frac{P_{\mathcal{D}'}(p)}{P_{\mathcal{D}}(p)} = \frac{P_{\mathcal{D}'}(p)}{P(p)} \tag{4}$$

where $\mathcal{D}'$ is implicitly defined by the combination $[\mathcal{D}, Q]$. As indicated in the rightmost form, we simply drop the suffix when the probability is used to denote the corpus-wide probability. For notational convenience, we re-write the numerator as being the conditional probability of p given the query:

$$P_{\mathcal{D}'}(p) \equiv P(p|Q) \equiv P(p|[\{q_1, \ldots, q_r\}, O])$$

Re-writing this using Bayes' theorem:

$$P(p|[\{q_1, \ldots, q_r\}, O]) = \frac{P([\{q_1, \ldots, q_r\}, O]|p) \times P(p)}{P([\{q_1, \ldots, q_r\}, O])}$$

Since we deal with one query at a time, and intend to score all phrases based on the chosen query, the denominator that denotes the probability of the query can be dropped since that simply scales down the probability of each phrase by the same amount. Thus,

$$P(p|[\{q_1, \ldots, q_r\}, O]) \approx P([\{q_1, \ldots, q_r\}, O]|p) \times P(p)$$

Before further simplification, it is useful to note that the scoring function has a much simpler formulation due to the corpus-wide phrase probability getting cancelled out from the numerator and the denominator.

$$\begin{aligned}\mathcal{S}_{\mathcal{D}}(p,Q) &= \frac{P([\{q_1,\ldots,q_r\},O]|p)\times \cancel{P(p)}}{\cancel{P(p)}} \\ &= P([\{q_1,\ldots,q_k\},O]|p)\end{aligned} \tag{5}$$

We now break down the composite probability term that comprises of many query keywords, to simpler terms each of which pertain to the scoring wrt one of the keywords. Evidently, this breakdown depends on the operator O.

$$P([\{q_1,\ldots,q_r\},O]|p) = \begin{cases} P((\cap_i q_i)|p) & \text{if } O = AND \\ P((\cup_i q_i)|p) & \text{if } O = OR \end{cases} \tag{6}$$

In the above, we overload notations and use q_i to refer to $docs(\mathcal{D}, q_i)$ (Ref. Eq. 2); since we do not explicitly model $docs(.,.)$ in our formulation, there is no notational conflict in the ensuing discussion. We now introduce an independence assumption that guides further simplification of the form in Equation 6.

4.1.1 The Query Word Independence Assumption

Take the example of a query for the Reuters-21578 dataset[2], *trade reserves* for which *economic minister* is the top interesting phrase according to the interestingness measure in Equation 1. Since the phrase has high interestingness for the query, it is obviously expected to be well-correlated with the query words. For such phrases, we assume that given the occurence information of the phrase *economic minister*, the occurence of the keywords *trade* and *reserves* are conditionally independent of each other. Let us consider an *AND* query composed of query terms q_1 and q_2 and any phrase p:

$$P(q_1,q_2|p) = \frac{P(q_1,q_2,p)}{P(p)} = \frac{P(q_1|q_2,p)\times P(q_2|p)\times \cancel{P(p)}}{\cancel{P(p)}}$$

The second simplification is done by using the Chain Rule[3]. Now, for a phrase that has a high interestingness for the query, its occurence already provides significant evidence on the occurence of each query word. Thus, we postulate that the occurence information of q_2 is of very low incremental utility in determining $P(q_1|q_2,p)$ when p is among the top interesting phrases for the query. Coming back to our example, the argument translates to saying that given the occurence information of *economic minister* in a document, the occurence or non-occurence of the word *reserves* provides very little additional information to estimate the occurence of the word *trade*. More formally, our assumption is that:

$$\forall_p, I_{\mathcal{D}}(p,\mathcal{D}') \rightarrow 1.0, P(q_1|q_2,p) \approx P(q_1|p)$$

i.e.,

$$\forall_p, I_{\mathcal{D}}(p,\mathcal{D}') \rightarrow 1.0, P(q_1,q_2|p) \approx P(q_1|p)\times P(q_2|p)$$

More generally stated, for a query $Q = [\{q_1,\ldots,q_r\}, AND]$,

$$\forall_p, I_{\mathcal{D}}(p,\mathcal{D}') \rightarrow 1.0, P((\cap_i q_i)|p) \approx \prod_i P(q_i|p) \tag{7}$$

[2] http://www.daviddlewis.com/resources/testcollections/reuters21578/

[3] http://en.wikipedia.org/wiki/Chain_rule_(probability)

Usage of the Independence Assumption: Though our assumption above is expected to correctly estimate the interestingness for top phrases correlated with the query words, the simplification of the formulation under the independence assumption is very attractive, for computational reasons. Moreover, we do not know the top phrases beforehand, to know whether the independence assumption would hold for a particular candidate phrase. The only potential hazard in usage of the independence assumption for all phrases during the phrase discovery process is whether not-so-good phrases could *overtake* the scores of the good ones. In our two-word query example, this could happen if:

$$P(q_1|p_{ni})\times P(q_2|p_{ni}) >> P(q_1,q_2|p_{ni})$$

for a *n*on-*i*nteresting phrase p_{ni} (wrt the query q_i *and* q_2). In particular, our formulation does not suffer if the independence assumption underestimates the value of $P(q_1,q_2|p_{ni})$, but, does suffer when the LHS[4] in the equation above significantly overestimates the RHS. Using the chain rule simplification above, our potentially hazardous condition translates to:

$$P(q_1|p_{ni})\times P(q_2|p_{ni}) >> P(q_1|q_2,p_{ni})\times P(q_2|p_{ni})$$

i.e.,

$$P(q_1|p_{ni}) >> P(q_1|q_2,p_{ni})$$

Consider the case where p_{ni} is not very well correlated with either q_1 or q_2. In the case where there is no correlation between any pair in the triplet $\{q_1,q_2,p_{ni}\}$, the LHS would evaluate to the RHS and thus the hazardous inequality above wont fire. In the case where q_1 is correlated with q_2 and p_{ni} is correlated with neither, the probability of occurence of q_1 actually increases with the occurence of q_2, and thus the LHS could evaluate to a score lesser than the RHS. Thus, we postulate that we are probably *safe* under most reasonable scenarios. ■

As discussed above, we use the independence assumption across all phrases in $\mathcal{P}$ in our phrase scoring method. Since the simplifications beyond Eq. 6 are operator specific, we describe the AND and OR operator cases in separate subsections below.

4.1.2 The AND Operator

Resuming from Equation 6 and applying the independence assumption from Equation 7,

$$S_{\mathcal{D}}^{AND}(p,Q) = \prod_i P(q_i|p)$$

where $S_{\mathcal{D}}^{AND}(.,.)$ is used to represent the scoring function for *AND* queries. Re-writing it in log terms converts this into a sum form as follows:

$$S_{\mathcal{D}}^{AND}(p,Q) = \sum_i log(P(q_i|p)) \tag{8}$$

4.1.3 The OR Operator

[4] left-hand side part of the equation

Consider a three word *OR* query composed of words $\{q_1, q_2, q_3\}$. Now, to compute the score for the union, we could simply take the sum of the terms corresponding to the individual q_is, and then subtract the intersections for every 2-word combinations such as $[q_1, q_2]$ and $[q_1, q_3]$ since they get counted twice in the sum of terms. Evidently, the subtraction discounts the score corresponding to the triplet $[q_1, q_2, q_3]$ as many times as it was added, and thus needs to be added back. Thus, the three word query evaluates to the following when computed according to Equation 6.

$$S_{\mathcal{D}}^{OR}(p, [\{q_1, q_2, q_3\}, OR]) = (P(q_1|p) + P(q_2|p) + P(q_3|p)) - (P(q_1, q_2|p) + P(q_1, q_3|p) + P(q_2, q_3|p)) + P(q_1, q_2, q_3|p) \tag{9}$$

Generalizing this to any query $Q = [\{q_1, \ldots, q_k\}, OR]$,

$$S_{\mathcal{D}}^{OR}(p, Q) = (-1)^0 \sum_i P(q_i|p) + (-1)^1 \sum_{i,j,i\neq j} P(q_i, q_j|p) + \ldots + (-1)^{k-1} P(q_1, \ldots, q_k|p) \tag{10}$$

Applying the independence assumption to the above, the terms involving the joint probabilities can be simplified:

$$S_{\mathcal{D}}^{OR}(p, Q) = (-1)^0 \sum_i P(q_i|p) + (-1)^1 \sum_{(i,j),i\neq j} \prod_{x\in\{i,j\}} P(q_x|p) + \ldots + (-1)^{k-1} \prod_{i=1}^{k} P(q_i|p) \tag{11}$$

The term with the co-efficient $(-1)^x$ represents the sum of ${}^kC_{x+1}$ terms, each of which are the product of $(x + 1)$ probability terms. Thus, the absolute value and hence the influence of the term in the score $S_{\mathcal{D}}^{OR}(.,.)$ is likely to reduce with increasing x. We could thus approximate $S_{\mathcal{D}}^{OR}(.,.)$ by discarding the terms beyond a threshold value of x. For example, if we cut-off at $x \geq 1$, thus retaining only the first term, the formulation becomes:

$$S_{\mathcal{D}}^{OR}(p, Q) = \sum_i P(q_i|p) \tag{12}$$

We will use the above formulation for OR queries in the rest of the paper.

4.2 Disk-resident List Indexes

Much like previous methods for interesting phrase mining, we employ list-based indexes in our approach. For most cases where large datasets are to be dealt with, the indexes become very large and need to be disk-resident. We use two kinds of disk-resident indexes, (1) for storing phrases, and (2) for storing word-specific lists of scored phrases.

4.2.1 Phrase List

The phrase list stores the lexical representation of each phrase from $\mathcal{P}$ that satisfies the minimum frequency threshold (Ref. Section 2). Each entry in this list is of length s bytes, with shorter phrases being padded with zeros upto s bytes. This imposes a restriction that we cannot handle phrases containing more than s characters; thus, s may be set to be sufficiently high to accomodate even long phrases. We use an s value of 50, and this was seen to cover all the phrases that we encountered in the text corpora that we experimented with. We use the position of the phrase in the list to denote the ID of the phrase. Thus, to find the phrase with ID $= i$, we would check for the stretch of bytes spanning from Offset $[(i-1) \times s + 1]$ through $[i \times s]$; an illustration of the phrase list and the offset calculation appears in Figure 1. Having defined this ID $\rightarrow$ phrase mapping using the Phrase List index, we can now use these indexes to represent phrase scores in the word-specific lists that we describe next.

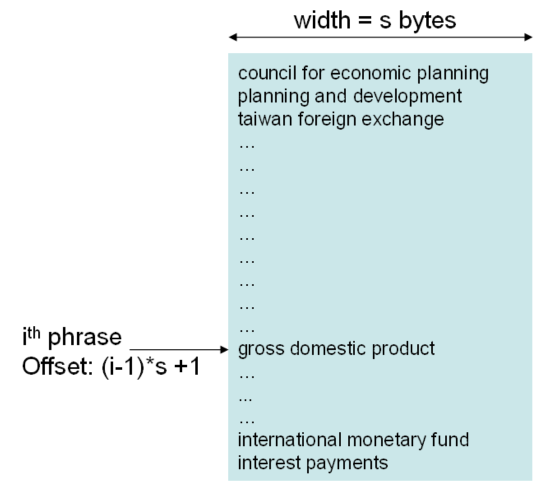

Figure 1: Phrase List Example & Offset Calculation

4.2.2 Word-specific Lists

In our phrase scoring methods as outlined in Equations 8 and 12, we make use of $P(q_i|p)$ scores. Phrases with high values of these scores (for any word q_i in the query) are considered more valuable than the others. Thus, we use a sorted list representation for storing phrase scores for each word. As mentioned earlier, we use *word* to generically refer to any word or metadata facet that could appear in the query. For every word q, we maintain a list of $[phraseid, prob]$ pairs sorted in the non-increasing order of scores. An example word-specific list is illustrated in Figure 2. The prob field is used to represent $P(q|p)$ (Ref. Section 4.1) where p is the phrase that corresponds to the phraseid in the tuple. At the risk of re-stating the obvious, the following calculation us used to compute the $P(q|p)$ score:

$$P(q|p) = \frac{|docs(\mathcal{D}, q) \bigcap docs(\mathcal{D}, p)|}{|docs(\mathcal{D}, p)|} \tag{13}$$

When multiple phrases are tied on the same score, they are ordered in the increasing order of phrase IDs (like Phrases 1134 and 1987 in the example figure). Each pair in the phrase list occupies exactly $\lceil log(|\mathcal{P}|)\rceil + 64$ bits. This is so since the PhraseID can be represented in $\lceil log(|\mathcal{P}|)\rceil$ bits since there are at most $|\mathcal{P}|$ that could appear in the lists, and a double precision floating point value takes up 64 bits[5]. We omit phrases that score 0.0 since they do not add to

[5]http://en.wikipedia.org/wiki/Double_precision_floating-point_format

the score under our phrase scoring formulation, and could additionally omit very low-scored phrases if storage space is at a premium. We will discuss such optimizations in the experimental section.

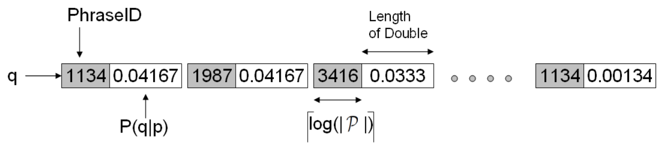

Figure 2: An Example Word-specific List

The number of qs for which such lists need to be stored is of interest to determine the size of the index. If we would like to allow for querying only based on metadata facets, the number of lists are likely to be small enough since categorical metadata facets rarely go beyond a few hundreds in number; numerical metadata (e.g., price of a product) may be bucketed appropriately. However, if we would like to allow for a very expressive query system, it would be necessary to maintain lists for any word that occurs in the corpus, i.e., we would need to accomodate $|W|$ lists. We will analyze practical index sizes in terms of absolute numbers in the experimental evaluation.

4.3 Scoring using Disk-resident Indexes

The algorithm for scoring phrases using disk-resident indexes makes use of the *No Random Access* algorithm[6] for combining information from multiple subsystems. Let the lists corresponding to the query word q_i be L_i; we access entries from these lists in round-robin fashion in the course of the algorithm. Thus, the first entries of each of the r lists (where there are r words in the query) are read, followed by the second entries and so on. Having read a few entries each from each of the lists, there would be phrases that are partially seen (seen in some, but not in all of the r lists), fully seen (seen in all of the r lists already), and unseen (not yet seen in any of the lists). Given that the lists are sorted in the non-increasing order of phrase scores as outlined in Section 4.2.2, we can define upper bounds of the scores based on the values encountered so far.

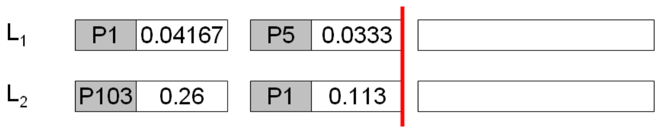

Figure 3: Bounded Candidate Scores Example

Our sum based score formulation in Eq. 8 and 12 makes it easy to arrive at score bounds for unseen and partially seen candidates. We illustrate candidate score bounds by means of an example in Figure 3. Consider an OR query with two words, where the two lists in the figure have been read till the red line; till this, $P1$ has been seen on both the lists, whereas $P5$ and $P103$ have been seen only on one list each. Given that we have seen scores till 0.0333 on L_1, any hitherto unseen entry in L_1 would have a maximum score of 0.0333. We call such bounds based on the last seen values as *global bounds*. Thus, the maximum possible score for $P103$ based on the currently available information is $(0.26+0.0333) = 0.2933$. Consequently, the range of possible scores

Alg. 1 ***Scoring using Disk-resident Indexes (NRA)***

Input. $\mathcal{D}, Q = [\{q_1,\ldots,q_r\},O], k$
Output. Top-k Interesting Phrases $\approx R(\mathcal{D},\mathcal{D}',k)$

```
1. $\forall_{i=1}^{r}, fetch\ L_i = list\ corresponding\ to\ q_i$
2. $\mathcal{C} = \{\}$
3. $checknew = true$
4. $while(not\ all\ lists\ have\ been\ fully\ read)$
5.     $for\ each\ list, L_i$
6.         $read\ next\ [phrase, prob]\ from\ L_i$
7.         $score = (O = OR)?prob : log(prob)$
8.         $if(phrase \in \mathcal{C} \vee (isNew(phrase) \wedge checknew))$
9.             $update\ bounds\ for\ phrase\ in\ \mathcal{C}$
10.     $update\ global\ bounds\ based\ on\ scores\ seen$
11.     $check\ if\ new\ candidates\ need\ to\ be\ considered$
            $and\ update\ checknew\ flag$
12.     $prune\ candidates\ in\ \mathcal{C}\ based\ on\ new\ local\ bounds$
13.     $if(current\ top-k\ is\ final) break$
14. $return\ the\ top-k\ phrases\ from\ \mathcal{C}$
```

for $P103$ would be $0.26 \leq Score(P103) \leq 0.2933$ (i.e., the *candidate specific bounds*). Analogously, the upper bound for $P5$ would be $(0.113 + 0.0333) = 0.1433$. The score of any fully unseen candidate would hence be limited by an upper bound of $(0.113 + 0.0333) = 0.1433$. Consider the case where $k = 2$; the two top candidates as of now are $P1$ with score 0.15467 and $P103 = [0.26, 0.2933]$. Since the upper bound of $P5$, the only other candidate seen so far, has an upper bound of 0.1433, it would not be able to overtake either $P1$ or $P103$ (with lower bound of 0.26) whatever be the content of the remaining lists. Similarly, any hitherto unseen candidate would also be unable to beat either since such candidates have a score upper bound of 0.1433. Given these two conditions, we are now safe to stop reading the lists and declare that the top two phrases are $\{P1, P103\}$.

Algorithm 1 illustrates the complete scoring methodology using bounds based pruning as illustrated above; we will refer to this approach as **NRA** (based on the *No Random Access* framework on which it is modeled). The algorithm maintains the candidate set $\mathcal{C}$ and proceeds by reading entries from each list (Line 6), computing the operator-specific score (Line 7) and using such scores to update the candidate-specific bounds (Line 8-9) and global bounds (Line 10). If the lower bound of the current top-k element in $\mathcal{C}$ is higher than the highest possible score of any unseen candidate, the *checknew* flag is turned off so that no previously unseen candidates would be considered thereafter. Once the iterations are over, the phrases corresponding to top-k candidates from $\mathcal{C}$ based on their upper bounds are lookedup from the Phrase List (Section 4.2.1) and can be output as the result set.

Partial Lists for early Termination: Though Algorithm 1 is shown to run till the lists are exhausted, we could choose to just scan a fraction of the lists for applications that require very low response times and can manage with coarse approximations of results. In our experimental evaluation, we will experiment with partial lists, where a parameter is used to denote the percentage of lists that would be traversed. If the parameter is set to 10%, our algorithm is meant to terminate after reading 10% of each list, L_i.

4.4 In-Memory Operation

In cases where we have enough memory to hold the indexes, or are working with small datasets, or can tolerate coarse approximations in results through usage of small partial lists, the (partial) lists from Section 4.2 could be held in memory, and the Algorithm in Section 4.3 could work on the in-memory lists. However, smaller lists (or small fractions of large lists) could be organized differently in memory to enable fast scoring, and thus, low response times. We discuss such a design of lists, and a scoring algorithm for such lists in this section.

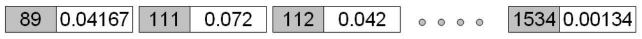

Figure 4: Example Phrase ID Ordered List

4.4.1 *Word-specific ID-ordered Phrase Lists*

Instead of ordering word-specific lists (Ref. Section 4.2.2) based on the scores, we propose ordering them based on the phrase IDs for this adaptation. Thus, instead of ordering the [*phraseid*, *prob*] pairs on the score, these are ordered in the increasing order of phrase IDs. As earlier, those with phrases p with $P(q|p) = 0$ are omitted from the index. Additionally, similar to the discussion in Section 4.3, we can employ partial lists as needed according to the response time requirements; these are constructed by truncating the score-ordered lists to extract the prefix (e.g., a specific percentage of the lists), which are then re-ordered based on the phrase IDs. An example Phrase-ID ordered list appears in Figure 4; since the list is ordered on IDs, the value of *prob* may vary haphazardly while progressing down the list. In addition to such lists, we also use in-memory phrase lists exactly as outlined in Section 4.2.1 for in-memory operations.

4.4.2 *Scoring using ID-ordered Phrase Lists*

Scoring phrases in response to a query using the Phrase ID-ordered list is fairly straightforward and can be accomplished using the Sort-merge Join algorithm[6] since the lists are ordered by the join attribute, the phrase ID; we will call this approach SMJ and is illustrated in Algorithm 2. The candidate set is initialized in Line 1. In each iteration, the list having the least unread phrase ID is read (Line 4-5) and the candidate set is updated with the newly seen score (Lines 6-7). When the lists are exhausted, the phrases in $\mathcal{C}$ are ordered based on the scores, and the top-k phrases are returned as the results (Line 8).

Working with Partial Lists: Each phrase list may be constructed by taking a fraction of the corresponding word-specific lists (Ref. Section 4.2.2) and re-ordering them based on Phrase IDs. Construction of such partial lists is a construction-time decision, and once the ID-ordered lists have been constructed using a pre-specified fraction (say, 20%) of the word-specific lists, we cannot, at run-time, decide to work with a larger or a smaller fraction. This is so since the lists are ID-ordered, and many highly scored phrases could be towards the end of the list since the lists are not ordered by scores; this may be contrasted with NRA that can choose to stop at different fractions during different runs, since the lists it works with are ordered based on scores.

[6]http://en.wikipedia.org/wiki/Sort-merge_join

Alg. 2 ***Scoring using PhraseID-ordered Lists (SMJ)***

```
Input. $\mathcal{D}, Q = [\{q_1, \ldots, q_r\}, O], k$
Output. Top-k Interesting Phrases $\approx R(\mathcal{D}, \mathcal{D}', k)$

1. $\mathcal{C} = \{\}$
2. $\forall_{i=1}^{r}, fetch\ L_i = the\ ID\text{-}ordered\ list\ for\ q_i$
3. $while(not\ all\ lists\ are\ exhausted)$
4.        $index = list\ containing\ lowest$
          $phraseid\ among\ unread\ entries$
5.        $read\ next\ entry\ [phrase, prob]\ from\ L_{index}$
6.        $score = (O = OR)?prob : log(prob)$
7.        $add\ and/or\ update\ score\ of\ phrase\ in\ \mathcal{C}$
8. $order\ \mathcal{C}\ based\ on\ scores\ and\ return\ top-k$
   $phrases\ as\ the\ result\ set$
```

4.5 Analysis

We now analyze the running times of the presented approaches against the input parameters. We denote the length of the word-specific lists (score or ID-ordered) by l. For the NRA approach in Section 4.3, the outer loop runs through the each entry in each of the r lists. In the highly unlikely case where phrases do not repeat across lists, these could encompass lr phrases. The candidate set could thus grow up to a cardinality of lr; as an optimization, we perform the pruning operations (Line 12) only once in a batch of b iterations (at the cost of delayed pruning within $\mathcal{C}$). While small batch sizes in the order of thousands could drastically improve run-times, extremely large values can be detrimental because prunable candidates are unnecessarily held too long in $\mathcal{C}$. Since the pruning operations are linear in $|\mathcal{C}|(= \mathcal{O}(lr))$, the entire complexity evaluates to $\mathcal{O}(lr \times \frac{lr}{b}) \equiv \mathcal{O}(l^2r^2/b)$. Of these, the number of words in the query, r, is typically 2-5; the list lengths can be pruned to reduce l at the expense of approximating the results, and the batch size may be tuned based on response time requirements. The complexity of the SMJ approach fom Section 4.4.2 may also be derived from a similar construction since the only difference is the absence of the pruning/book-keeping operations. Thus, the complexity evaluates to $\mathcal{O}(lr + k\ log(lr))$, where the $k\ log(lr)$ term stands for the final partial sort to derive the top-k results from the candidate set.

Though the NRA approach is seen to be worse in time complexity, the pruning phase is expected to allow for early stopping, thus making it necessary only to see a small fraction of the l entries in each list. SMJ, on the other hand, cannot stop without scanning each of the l items; this difference makes NRA suitable for long lists, wheras SMJ is expected to be very effective for short lists (i.e., low values of l).

4.5.1 *Incremental Operation*

Since we maintain conditional probabilities, it is not easy to maintain the indexes current in the presence of a lot of document insertions and deletions into the corpus. However, to avoid re-computing the indexes at every document update, a separate inverted index can be maintained on the updated (added or deleted) documents indexed on the features and phrases. When a particular phrase is taken for consideration into the candidate set within SMJ or NRA, an additional query may be performed on the separate index to get the delta of the conditional probabilities for the word-phrase pair so that the correct conditional probability can be used within SMJ or NRA. While this would work correctly for SMJ, such probability adjustments make NRA's pruning phase approximate thus resulting in further approximations of the result. In any case, periodically, the separate index may be flushed when it grows big enough, and the list-indexes can be re-computed offline.

5. EXPERIMENTAL EVALUATION

5.1 Datasets and Experimental Setup

We use two datasets in our experimental evaluation; the *Reuters* dataset that comprises of *21578* documents and a much larger *Pubmed* datasets that has *655k* documents. The *Reuters* dataset is a collection of newswire articles and is popular in the data mining community. We use *100* queries as the query set for *Reuters* and these are harvested from among frequent phrases in the corpus. Among the query set are two queries of six words each, and a further two queries made up of five words each; the rest are formed of two to four words. The *Pubmed* abstracts dataset is a collection of abstracts from among biomedical literature, and is of a total size of close to $2GB$. Due to the wide diversity of phrases in the dataset and given the absence of a standard query set, we randomly picked 10 frequent phrases occuring in the PubMed abstracts and fetched 10 extended phrases using Google AutoComplete API[7]. Since the Google API is not domain specific, it also threw up a lot of non-biomedical phrases which were seen to match with very few documents in the Pubmed dataset; thus, we chose Google API suggestions that were relevant to the biomedical domain and had at least a dozen matches in the Pubmed dataset and finalized on a query set comprising of 52 queries.

All methods described in this paper were implemented in Java (Using Oracle Sun JDK 1.7). The experiments were run on a Linux 64 bit machine having 16GB of main memeory, quad core Intel Xeon(R) processor with 2.13GHz. We consistently set the number of interesting phrases parameter, k, to 5, for our experiments.

5.2 Baseline and Comparative Evaluation

In our evaluations in this section, we compare our approach against the Improved Sequential Pattern Indexing approach, the latest algorithm among those reviewed in Section 2. We refer to this approach as *GM* based on the initials of the authors. The criteria for comparison are two-fold; we first evaluate the quality of the results from our (approximate) approaches using standard Information Retrieval evaluation measures such as Precision, NDCG, MAP and MRR against the exact results returned by *GM*. We then compare *GM* against our approaches on response times.

Precision, MRR, MAP and NDCG: Precision represents the fraction of correct results among the top-k (i.e., top-5, for our case) results whereas MRR stands for the reciprocal rank of the first correct result (1.0 if the first correct result is at the top position, 0.5 if at the second and so on). NDCG and average precision (MAP) are rank-sensitive measures unlike precision which just counts the fraction of correct results; for example, if the 2 correct results among 5 results retrieved are the top-2, the NDCG and MAP would assign a higher score than if they were in the 4^{th} and 5^{th} positions. All these measures are in the range $[0, 1]$ with the 1.0 standing for best conformance with the correct results. Over the many years, these have become standard evaluation measures for retrieval evaluation[8].

[7]http://gofishdigital.com/autocomplete/

5.3 Result Quality Evaluation

Our techniques for interesting phrase mining, SMJ and NRA, both strive to discover close approximations of the exact set of top-k interesting phrases. This is so since both of them use the independence assumption outlined in Section 4.1.1, which may not hold in *all* cases. It is hence important to quantify the extent to which usage of the independence assumption affects our results. Since SMJ and NRA differ only in the organization of the lists and the traversal strategy, these give exactly the same results for any query-dataset combination. As discussed in Section 4, both of these approaches can be fed with partial lists that are formed by choosing a specific fraction of the top-scored entries in the word-specific phrase lists.

For every query, we collect the top-5 result phrases from our list-based approach (either SMJ or NRA), and mark each of them as *correct* if they either have an actual interestingness of 1.0 (being the absolute maximum interestingness possible) or are among the top-5 most interesting phrases for that query; interestingness is estimated using Equation 1. All other results are marked incorrect. Information Retrieval measures such as Precision, MRR, NDCG and MAP quantify the correctness of the results.

We plot the quality measures averaged across queries for the AND and OR queries against 20% and 50% of the partial lists for Reuters and Pubmed in Figures 5 and 6 respectively. The value in the X-axis represents the configuration as a [*percentage of list*, *operator*] pair; thus, 20-AND stands for the evaluation of the run on 20% of the partial lists for the AND operator. It may be seen that we consistently get very good results even while only processing 20% of the lists for both the datasets. For the AND query, results on Reuters are seen to improve from 0.90 to 0.95 for most IR evaluation measures, while the precision lags a little behind; the accuracy on the OR query is much better with our evaluation measures reaching very close to unity even at 20%. Our techniques achieve a better result on the larger Pubmed dataset, with accuracy inching towards the absolute maximum of unity even for lists of 20% size. The performance on the OR queries are generally seen to be better than those for the AND queries, thus suggesting that our scoring mechanism is very close to reality when working with larger subcollections. The significantly improved performance on the larger Pubmed dataset confirms this; these observations are intuitive since statistical estimates improve with larger sample sizes. The above figures confirm that the independence assumption has served us well, with the losses in accuracy being very negligible and less than 5-10% for most cases.

5.4 Runtimes for In-Memory Operation: SMJ vs. GM

We now evaluate the performance of the SMJ approach from Section 4.4.2 against the GM baseline. In SMJ, we could use partial lists where the top-p% scores from each word's list are truncated and re-ordered according to Phase IDs. Since our approach traverses word-specific lists unlike

[8]http://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-ranked-retrieval-results-1.html

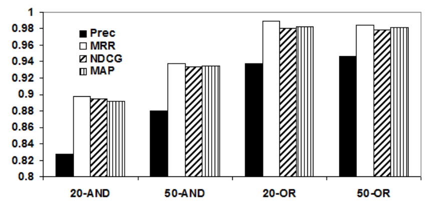

Figure 5: Result Quality Evaluation (Reuters)

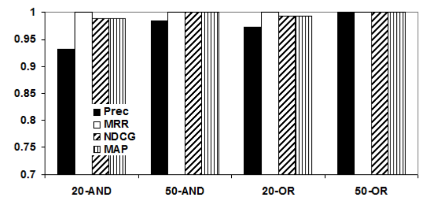

Figure 6: Result Quality Evaluation (Pubmed)

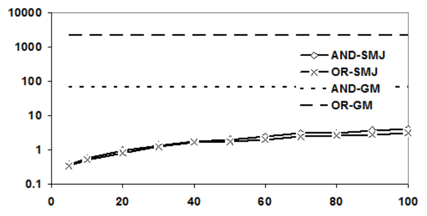

Figure 7: Running Times (in ms): SMJ vs. GM (Reuters)

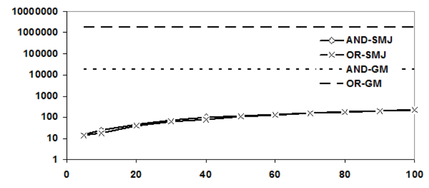

Figure 8: Running Times (in ms): SMJ vs. GM (Pubmed)

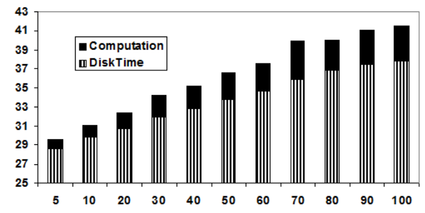

Figure 9: Break-up of Times for Reuters AND queries (ms)

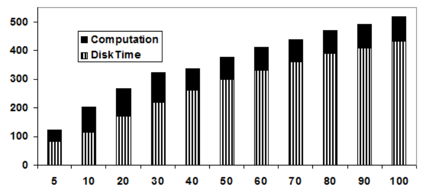

Figure 10: Break-up of Times for Pubmed AND queries (ms)

GM that needs to access as many document indexes that satisfy the query, our approach is expected to perform much faster than the latter. We plot the various running times for different percentages of partial lists for SMJ along with the response times for GM; the Reuters and Pubmed charts with the running times in log-scale are in Figures 7 and 8 respectively.

For Reuters, SMJ is able to discover results within fractions of a millisecond as long as only one-fifth or less lists are traversed (i.e., upto 20%), and takes only as much as $3.5ms$ even when the entire lists are traversed. The AND and OR queries take similar times since they vary only in the score aggregation mechanism. This is in sharp contrast with the GM approach whose response times are of the order of many tens of milliseconds for the AND queries and a few seconds for the OR query. The difference between the running times of AND and OR queries is intuitive since the OR query leads to more documents to be considered since all documents that contain at least one word in the query are under consideration, whereas the AND query only needs to check documents that contain *all* the words in the query. The relative trends for the much larger Pubmed dataset are similar to the ones observed for Reuters with SMJ outperforming GM by *2* and *4* orders of magnitude for AND and OR queries respectively. While the response times for SMJ remain under a quarter of a second for all configurations, GM takes as much as tens of seconds to respond to AND queries and close to half-an-hour to respond to OR queries making GM inappropriate for any query system with a real-time usage scenario.

5.5 Disk-based Operation: NRA

We now evaluate the response time of the NRA algorithm (Ref. Section 4.3) that does scoring using disk-resident lists. Since there is no disk-based version of GM, we first analyze NRA in isolation on various metrics such as response time and disk IO costs. In order to separate out and analyze the computational and disk costs, we use a simulation of disk-based runs. Our disk-based simulation uses the same

Table 4: Sample Results

Pubmed AND Query: *protein expression bacteria*
binding protein hfq
rna binding protein hfq
proteins expressed in bacteria
protein a ccpa
expression in bacteria

Reuters OR Query: *trade reserves*
economic minister
reserves
taiwan's foreign exchange reserves
economic planning
economic planning and development

setup described in [4] where the disk IO costs are calculated based on the log of disk accesses; we use a page size of 32 kilobytes and use a 16 page LRU[9] cache which does a 1-page lookahead on a page access; each sequential access and random access is accounted for by adding 1ms and 10ms respectively, to the disk IO time. These disk IO costs are in line with reported numbers for Windows[10] and Linux[11] operating systems. The disk IO cost computed using the logs is then simply added to the response time of an in-memory implementation of the NRA algorithm on a machine with large amounts of memory (enough to hold the disk indexes) to get the response time of a disk-based implementation [14]. Such a simulation enables us to profile the disk access costs and the computation time separately and is hence more useful than the total response time from a real disk-based implementation, from an analysis perspective.

Figures 9 and 10 show the average NRA costs for the AND queries for the Reuters and Pubmed datasets respectively. The figures, in addition to indicating the total re-

[9] http://en.wikipedia.org/wiki/Least_recently_used#Least_Recently_Used
[10] http://research.microsoft.com/pubs/69781/tr-2000-55.pdf
[11] http://www.linuxinsight.com/how_fast_is_your_disk.html

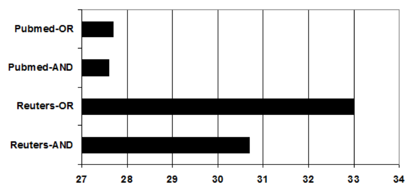

Figure 11: Percentage of Lists Traversed by NRA

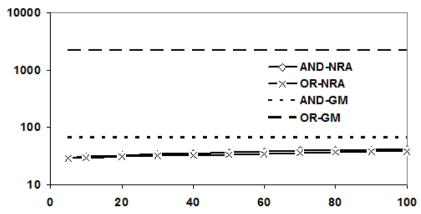

Figure 12: Running Times (in ms): NRA vs. GM (Reuters)

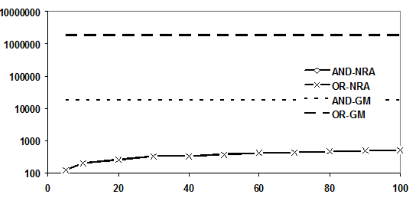

Figure 13: Running Times (in ms): NRA vs. GM (Pubmed)

sponse times, also show the breakup of costs as computational and disk access costs. As could be easily observed from the figures, disk access is responsible for upto $84-89\%$ of the response time across the datasets. The trends for tne OR queries were similar and the charts have been omitted for brevity. The absolute response times are in order of 20-40ms for Reuters and 0.1-0.5 seconds on Pubmed, inclusive of the disk costs. One aspect of interest is the tapering off of the disk and computation costs at increasing percentages. For example, while the Pubmed disk access costs increase from $114ms$ to $171ms$ (i.e., a delta of $57ms$) while moving from the 10% configuration to the 20% configuration, the corresponding increase from $80\% \rightarrow 90\%$ is just $22ms$; the corresponding values of computational cost deltas are $8ms$ and $2ms$. This is illustrative of the pruning effectiveness since the NRA approach, by virtue of pruning, needs to go until the $80-90\%$ depths of the lists only for very few queries. The experimental results indicate that the NRA algorithm that works on disk-resident lists, is able to deliver very good response times and is able to revert back with results within half-a-millisecond for Pubmed queries.

Stopping Condition Effectiveness: Instead of the indirect way of judging pruning effectiveness as discussed above, we now resort to a more direct way of quantifying the depth to which NRA progresses through the lists. Figure 11 plots the average fractions of lists traversed for queries on the various datasets before the pruning condition enables it to stop. The pruning condition is seen to cause NRA to stop after reading a little over a quarter of the Pubmed lists, on an average. On Reuters lists, NRA is seen to go a little further, to just over 30% of the lists. The pruning condition effectiveness is not seen to vary much between AND and OR, with the dataset specific fractions being similar across the two different types of queries.

Deciding between NRA and SMJ for in-memory Operations: Since the pruning condition in NRA is seen to be effective and can enable early stopping unlike the SMJ which needs to go through entire lists before announcing the results, it is useful to analyze which is the preferred method for in-memory operations. Since the per-iteration computation is simpler in SMJ, it is expected to be effective when dealing with small fractional lists. Infact, SMJ beats NRA in in-memory operation response time until a partial list percentage of 35% for Pubmed, beyond which the pruning effectiveness of NRA makes it faster than SMJ. The corresponding value for Reuters is 90% since the word-specific lists are much smaller, Reuters being a small dataset. Thus, as long as coarse approximations are good enough and memory is not at a premium, small partial lists may be constructed and re-ordered according to IDs for SMJ to work on. For longer lists and large datasets, it would be much better off to employ NRA.

Comparison against in-memory GM: We now compare the response time of the disk-based NRA against the in-memory GM approach. Though this comparison is unfairly biased in favor of GM (since it does not need to do disk accesses while NRA would have to fetch each entry from disk incurring extra time), it serves to illustrate that the quantum of improvements that we achieve by using the list-based approach are massive. Figures 12 and 13 illustrate the comparison on the Reuters and Pubmed datasets respectively. For the Reuters dataset, NRA achieves upto 50% gains in response time over *GM* on AND queries, whereas it is seen to be 50 times faster on OR queries. The differences in running times are much more pronounced in the Pubmed dataset with NRA response times being 1/35th and 1/3500th of the GM response times on the AND and OR queries respectively.

5.6 Example Phrases

Having evaluated the techniques empirically, we now illustrate a few example phrases that were discovered by our techniques. The capability of the interestingness measure to unearth such correlated phrases makes it distinct from other query-based phrase retrieval tasks from the Information Retrieval community such as *query expansion*[18] and *query suggestions*[3]. Table 4 lists the top-5 results retrieved for two queries from our query collection; the first is an AND query on the Pubmed dataset, *protein expression bacteria*. Despite this being an AND query, each of the top-5 phrases have just one word overlapping with the keywords in the query. In the second case, that of an OR query on the Reuters dataset, three out of the top-5 phrases do not even have one word in common with the words in the query, but, are evidently very related to the query words themselves. Phrases with words from the query itself would have limited utility due to the redundant information. In cases where we would like to supress such redundant information altogether, we could just use a post-retrieval filter to filter out results with high overlap with the query. As seen from the results, our technique, by usage of the interestingness measure formulation, is able to discover phrases correlated with the query with and without lexical overlap with the query itself.

5.7 Miscallaneous Analyses

Index Size: The size of the index is of interest to judge the storage requirements for any disk-based query system. Since we use feature-specific list indexes, our index sizes are expected to be higher than the document specific indexes used by the baseline GM method. We analyzed the word-specific index list sizes for the words that we used in our

Table 5: Index Sizes

Dataset	List %	Index Size	NDCG	
			AND	OR
Reuters (30 megabytes)	10%	56 megabytes	0.83	0.98
	20%	111 megabytes	0.89	0.98
	50%	277 megabytes	0.93	0.98
Pubmed (2 gigabytes)	10%	90 gigabytes	0.96	0.99
	20%	179 gigabytes	0.99	0.99
	50%	446 gigabytes	1.0	1.0

Table 6: Interestingness Accuracy: Mean Difference

Reuters		Pubmed	
AND	OR	AND	OR
0.048	0.001	0.021	0.001

query. The average list sizes for Reuters was seen to be $37kb$ assuming 12 bytes per entry (4 for phrase ID and 8 for storing the probability value), and those for Pubmed were seen to be 5.4 megabytes. The vocabulary (number of distinct words) of these datasets were found to be approximately $15k$ and $170k$ respectively. As we have seen from the empirical observations, one-fifth of the index lists are typically enough to achieve high accuracies. Assuming the case where we would like to enable querying over all words, we analyze the index sizes for these datasets on various list sizes with respect to the accuracy achieved based on empirical observations in Table 5. As may be seen, 250 mb and 90 GB of storage for the Reuters and Pubmed datasets respectively enable achieving very high accuracies (> 0.9 on NDCG) for the phrase retrieval problem.

Accuracy of estimated Interestingness: The relative ordering of the top-k phrases arrived at using the independence assumption-based formulation is seen to be rather consistent with the reality, as seen from the quality analyses in Section 5.3. We now analyze the absolute divergence of our interestingness estimates from the reality. The mean difference between the estimated and real interestingness of the result phrases for each dataset, query-type configuration is listed in Table 6. Consistent with the relative trends in result quality, the mean difference is seen to be very low for OR queries and in the 0.02-0.05 range for AND queries. Thus, it is not only the relative ordering that is getting preserved among top phrases, but, the actual interestingness values are also estimated with very low error rates.

Extension to Metadata Facets: Though we have experimented solely with keyword queries due to the unavailability of metadata facets in the datasets we used (similar was the case in the baseline paper [8]), our technique may be easily extended to metadata facets by creating list indexes for keyword facets. The independence assumption is intuitively expected to hold as long as the metadata facets represent coherent sets of documents (e.g., topical metadata facets such as those that indicate a geographical region of origin of the article, or category of the article etc.) much like keywords. However, metadata facets that do not have this property (e.g., all articles from across geos and across topics published in 2001) could pose some challenges and the extent of validity of the independence assumption on those need to be empirically verified.

5.8 Summary of Experiments

We summarize the results for in-memory operation in Table 7. Since the baseline method is only for in-memory operation, we include only the in-memory runtimes in the Table. As may be easily inferred from the table, our methods are able to perform orders of magnitude faster than the baseline algorithm while being able to deliver results at pretty high accuracy.

Table 7: Experiments Summary: Quality and Performance in In-Memory Operation

Reuters Dataset					
Method	List %	NDCG		Runtime (ms)	
		AND	OR	AND	OR
GM (Baseline)	NA	1.0	1.0	67	2210
NRA	20%	0.89	0.98	1.7	1.4
	50%	0.93	0.98	2.7	2.1
SMJ	20%	0.89	0.98	1.0	0.8
	50%	0.93	0.98	2.0	1.7
Pubmed Dataset					
Method	List %	NDCG		Runtime (ms)	
		AND	OR	AND	OR
GM (Baseline)	NA	1.0	1.0	17817	1770119
NRA	20%	0.96	0.99	96	86
	50%	0.99	1.0	99	88
SMJ	20%	0.96	0.99	44	39
	50%	0.99	1.0	114	106

In our empirical analyses, firstly, we illustrated that the approximate top-k interesting phrases discovered by our methods that rely on the independence assumption mirrors the actual top-k phrases according to the interestingness criteria at accuracies of $> 90-95\%$ on all settings. Secondly, our running time analysis comparing the in-memory SMJ approach against the baseline method (GM) shows that SMJ is always much faster with response time improvements reaching upto two orders of magnitude in many cases. Our disk-based approach, NRA, was also seen to be always better than the in-memory baseline approach in terms of response times, despite the former incurring disk access costs as much as $8-9$ times of its computational expenses. The stopping condition in NRA was seen to be effective to the extent that results could be arrived at even with just traversing 30% of the lists. Lastly, we illustrated that the index sizes for our approach are small enough and can be implemented at low storage costs. As a summary, the massive improvements in response times achieved by our methods makes it possible to incorporate interesting phrase mining into real-time query systems; the previous approaches were probably usable only as pre-processing or back-end offline batch query processing systems due to response times reaching several minutes even on datasets with sub-million documents.

6. RELATED WORK

Discovery of meaningful phrases from a document corpus has been a popular field of research in text mining. Phrase retrieval is the core task in various problems such as document summarization, query expansion, query autocompletion, faceted search, generating phrase cloud etc. Document summarization[11, 5] is the process of reducing a text document to retain the most important points, for easy perusal or for scaling up indexing of large document corpora. The basic idea is to identify the minimal set of meaningful phrases required for describing a document. Once such interesting phrases are identified, each document can be represented solely using the set of interesting phrases it contains.

Tag Clouds[9] enable visualization of frequent phrases from a text corpus and is typically used to depict keyword metadata (tags) on websites, or to visualize free form text. Apart from such non-query specific summarization tasks, phrases are dealt with in various query-specific retrieval tasks central to Information Retrieval too. Query expansion[18, 12] is the task of suggesting meaningful re-formulations of an input IR query, to aid the user to zero-in onto documents of interest. Query expansion techniques often rely on interesting phrase discovery as a pre-processing step, so that phrases that scope down the user query meaningfully can be fetched in real-time when a user interacts with the system. Query suggestion mechanisms that try to aid the user while typing in a query is a similar task, but, is different in that it finds phrases that represent meaningful completions of incompletely specified information needs [3, 13]. Though broadly similar to such literature in being a phrase discovery task, our problem is different from the above in that we are looking to discover interesting phrases that are abundant in a sub-collection which is specified at query time using features such as keywords and metadata facets. Such interactive drill-down on large collections using features is commonly used in faceted search[17, 10] systems such as those in online shopping portals; however, we focus on drill-down on text corpora using keyword features.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we considered the problem of mining interesting phrases from document sub-collections specified using features such as keywords. We proposed the usage of a conditional independence assumption that assumes that the keyword features are independent of each other, given the phrases that are well-correlated with them. We postulated that such a conditional independence assumption may be used to score all phrases despite the assumption being expected to hold only for the top correlated ones. The independence assumption greatly simplifies the phrase scoring process, with scoring being reduced to sum and product based aggregations of conditional probabilities that could be pre-computed and stored. We outlined the construction of disk-based and in-memory word-specific list indexes and algorithms that can use them to discover top-k phrases given AND and OR queries on specified keywords. We empirically evaluated the accuracy of the top-k interesting phrases discovered by our methods against those discovered by an exact state-of-the-art method, GM. Based on a comparison on various Information Retrieval evaluation measures, our methods were seen to discover very close approximations of the actual top-k interesting phrases, consistently scoring in the $> 90\%$ range on the various evaluation measures employed. Our simplified phrase scoring process was seen to serve us well on the performance front too, with our methods being able to deliver orders of magnitude better response times than GM. Infact, our disk-based method was seen to outperform the in-memory baseline approach by up to two-three orders of magnitude on response times. Our empirical evaluation thus was seen to confirm that our list-based approaches would be the preferred approach for the problem of mining interesting phrases from document sub-collections. By lowering the response times to the millisecond ranges from several seconds and minutes through our approaches, real-time phrase mining is now a feasible task for search-like interactive systems.

Though we proposed the independence assumption as a means of solving the interesting phrase discovery problem, it could have many wide-ranging applications in techniques that deal with phrases as a first class entity (e.g., query expansion). Whether it can be used to simplify other kinds of interestingness formulations (e.g., [15]) could be a potential direction for future exploration.

8. REFERENCES

[1] N. Bansal and N. Koudas. Blogscope: a system for online analysis of high volume text streams. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 1410–1413. VLDB Endowment, 2007.

[2] S. Bedathur, K. Berberich, J. Dittrich, N. Mamoulis, and G. Weikum. Interesting-phrase mining for ad-hoc text analytics. *Proc. VLDB Endow.*, 3(1-2):1348–1357, Sept. 2010.

[3] S. Bhatia, D. Majumdar, and P. Mitra. Query suggestions in the absence of query logs. In *SIGIR*, pages 795–804, 2011.

[4] P. M. Deshpande, D. Padmanabhan, and K. Kummamuru. Efficient online top-k retrieval with arbitrary similarity measures. In *EDBT*, pages 356–367, 2008.

[5] G. Erkan and D. R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *J. Artif. Intell. Res.(JAIR)*, 22(1):457–479, 2004.

[6] R. Fagin. Combining fuzzy information: an overview. *SIGMOD Record*, 31(2):109–118, 2002.

[7] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[8] C. Gao and S. Michel. Top-k interesting phrase mining in ad-hoc collections using sequence pattern indexing. In *EDBT*, pages 264–275, 2012.

[9] M. J. Halvey and M. T. Keane. An assessment of tag presentation techniques. In *Proceedings of the 16th international conference on World Wide Web*, pages 1313–1314. ACM, 2007.

[10] C. Hostetter. Faceted searching with apache solr. *ApacheCon US 2006*, 2006.

[11] R. Mihalcea and P. Tarau. Textrank: Bringing order into texts. In *Proceedings of EMNLP*, volume 4. Barcelona, Spain, 2004.

[12] R. Navigli and P. Velardi. An analysis of ontology-based query expansion strategies. In *Proceedings of the 14th European Conference on Machine Learning, Workshop on Adaptive Text Extraction and Mining, Cavtat-Dubrovnik, Croatia*, pages 42–49, 2003.

[13] D. P, S. Chakraborti, and D. Khemani. Query suggestions for textual problem solution repositories. In *ECIR*, pages 569–581, 2013.

[14] D. Padmanabhan and P. Deshpande. Efficient rknn retrieval with arbitrary non-metric similarity measures. *PVLDB*, 3(1):1243–1254, 2010.

[15] A. Simitsis, A. Baid, Y. Sismanis, and B. Reinwald. Multidimensional content exploration. *PVLDB*, 1(1):660–671, 2008.

[16] A. Spink, D. Wolfram, M. B. J. Jansen, and T. Saracevic. Searching the Web: the public and their queries. *J. Am. Soc. Inf. Sci. Technol.*, 52(3):226–234, Feb. 2001.

[17] D. Tunkelang. Faceted search. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1(1):1–80, 2009.

[18] O. Vechtomova and Y. Wang. A study of the effect of term proximity on query expansion. *Journal of Information Science*, 32(4):324–333, 2006.

[19] Y. Yang, N. Bansal, W. Dakka, P. Ipeirotis, N. Koudas, and D. Papadias. Query by document. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, WSDM '09, pages 34–43, New York, NY, USA, 2009. ACM.