Key-Based Top-K Search in Multidimensional Databases

K.Anuratha[1], S.Senthamaraikannan[2] and R.Rajaguru[3]
[1]*Department of Computer Science and Engineering,* [2&3]*Department of Information Technology,*
Sethu Institute of Technology, Virudhunagar District, India.
[1]anuvinaya2003@gmail.com, [2]stanfordssk@gmail.com, [3]rajaguru.rama@yahoo.in

***Abstract* - Previous studies on supporting free- form keyword queries over RDBMSs provide users with linked-structures (e.g., a set of joined tuples) that are relevant to a given keyword query. Most of them focus on ranking individual tuples from one table or joins of multiple tables containing a set of keywords. The problem of keyword search in a data cube with text-rich dimension(s) (so-called *text cube*) is studied. The text cube is built on a multidimensional text database, where each row is associated with some text data (a document) and other structural dimensions (attributes). A cell in the text cube aggregates a set of documents with matching attribute values in a subset of dimensions. Given a keyword query, the goal is to find the *top-k* most relevant cells. This project studies the problem of keyword-based top k search in text cube, i.e., given a keyword query, find the top-k most relevant cells in a text cube. When users want to retrieve information from a text cube using keyword queries, relevant cells, rather than relevant documents, are preferred as the answers, because:(i) relevant cells are easy for users to browse; and (ii)relevant cells provide users insights about the relationship between the values of relational attributes and the text data. The proposed algorithm uses relevance scoring formula for finding the top-k relevant cells by exploring only a small portion of the whole text cube (when k is small) and enables early termination.**

I. Introduction

Analysis of documents in text databases and on the World Wide Web has been attracting researchers from various areas, such as data mining, machine learning, information retrieval, database systems, and natural language processing.

In general, studies in different areas have different emphases. Traditional information retrieval techniques (e.g., the inverted index and vector-space model) prove to be efficient and effective in searching relevant documents to answer unstructured keyword-based queries. Machine learning approaches are also widely used in text mining, providing with effective solutions to various problems. For example, the Naive Bayes model and the Support Vector Machines (SVMs) are used in document classification; K- means and the Expectation-Maximization (EM) algorithms are used in document clustering.

On the other hand, data warehousing and data mining also play important roles in analyzing documents, especially those stored in a special kind of databases called multi-dimensional text databases (ones with both relational dimensions and text fields). While information retrieval mainly addresses searching for documents and for information within documents according to users' information needs, the goal of text mining differs in the following sense: it focuses on finding and extracting useful patterns and hidden knowledge rom the information in documents and/or text databases, so as to improve the decision making process based on the text information.

In many real-life database applications, documents and the text data within them are stored in multidimensional text databases [2]. These kinds of databases are distinct from traditional data sources including relational databases, transaction databases, and text corpora. Formally, a multidimensional text database is defined as a relational database with text fields. A sample text database is shown in Table 1.1. The first three dimensions (Event, Time, and Publisher) are standard dimensions, just like those in relational databases. The last column contains text dimensions which are documents with text terms. Text databases provide structured attributes of documents, and the information needs of users vary where such needs can be modeled hierarchically. This makes OLAP and data cubes applicable. For instance (using Table 3.1), if a user wants to read news on the ice hockey games reported by the Vancouver Sun on February 20, 2010, then two documents d1 and d3 matching the query {Event = Ice hockey,

Table I: A multidimensional text database - Olympic News

Event	Time	Publisher	Text Data: Documents
Ice Hockey	2010/2/20	Vancouver Sun	D1={t1t2t3t4}
Ice Hockey	2010/2/23	Global and Mail	d3 = {t1, t2, t3, t6}
Ice Hockey	2010/2/20	Vancouver Sun	d4 = {t2, t4, t6, t7}
Figure Skating	2010/2/20	Global and Mail	d5 = {t1, t3, t5, t7}
Figure Skating	2010/2/23	Vancouver Sun	d6 = {t2, t5, t7, t9}
Curling	210/2/23	New York Times	d7 = {t3, t6, t8, t9}

Time = 2010/2/20, Publisher = Vancouver Sun} will be returned to her. If another user wants to skim all Olympic news reported by the Vancouver Sun on that day, we shall roll up to query {Event = _, Time = 2010/2/20,Publisher = Vancouver Sun} and return documents d1, d3 and d5 to her. The opposite operation of roll-up is called drill-down. In fact, roll-up and drill-down are two OLAP operations of great importance. Therefore, to meet different levels of information needs, it is natural for us to apply the data cube to

model and extend this text database.

II. Related Work

A. Keyword Search in RDBMs

Although based on different applications and motivations, keyword search in text cube is related to keyword search in RDBMSs, which has attracted a lot of attention recently by G. Weikum, *et al.* (2007). Most previous studies on keyword search in RDBMSs model the RDB as a graph (tuples/ tables as nodes, and foreign-key links as edges) and focus on finding minimal connected tuple trees that contain all the keywords. They can be categorized into two types. Y. Luo, *et al.* (2007) proposed the first type that uses SQL to find the connected trees. B. Ding, *et al.* (2007) presented the second type which materializes the RDB graph and proposes algorithms to enumerate (top-*k*) sub trees in the graph. Different from these two types of works, two recent studies L. Qin, *et al.* (2009) found single-center sub graphs from the RDB graph and multi-center induced sub graphs.

B.OLAP on Multidimensional Text Data

Cindy Xide Lin, et al (2008) introduced the text cube model and it mainly focuses on how to partially materialize inverted indexes and term frequency vectors in cells of text cube, and how to support OLAP queries (not keyword query) efficiently. D. Zhang, *et al.* (2009) proposed the topic cube model and is different from the text cube. The topic cube materializes the language model of the aggregated document in each cell. Efficient algorithms are proposed to compute this topic cube. The techniques presented by Cindy Xide Lin, *et al.* (2009) cannot be used directly to support keyword search, because the information materialized in text cube (term frequencies and inverted indexes) and in topic cube (language model) is query- independent.

C. Bottom-Up Computation

Constructs the data cube bottom-up, from the most aggregated apex cuboid to group-bys on a single dimension, then on a pair of dimensions, and so on. It also uses many optimization techniques introduced in the previous section. Figure 2.1 illustrates the processing tree and the partition method used in BUC on a 4-dimensional base table. Subfigure (b) shows the recursive nature of BUC: after sorting and partitioning data on dimension A, we deal with the partition (a1, _, _,_) first and recursively partition it on dimension B to proceed to its parent cell (a1, b1, _, _) and then the ancestor (a1, b1, c1, _) and so on. After dealing with partition a1, BUC continues on to process partitions a2, a3 and a4 in the same manner until all cells are materialized.

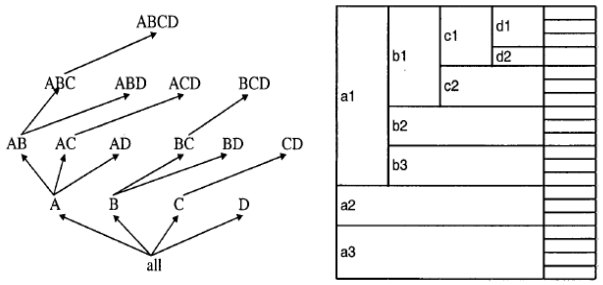

Fig.1 (a) BUC Processing Tree (b) BUC Partition

D. Frequency Pattern Mining Algorithm

Frequent patterns are patterns (sets of items, sequence, etc.) that occur frequently in a database . The supports of frequent patterns must exceed a pre-defined minimal support threshold. Frequent pattern mining has been studied extensively in the past two decades. It lays the foundation for many data mining tasks such as association rules and emerging pattern mining. Although its definition is concise, the mining algorithms are not trivial. FP-Growth is more important as efficient emerging pattern mining algorithms that use the FP-tree proposed in FP-Growth as data structures. The first scan of a database finds all frequent items, ranks them in frequency-descending order, and puts them into a head table. Then it compresses the database into a prefix tree called FP-tree. A complete set of frequent patterns can be mined by recursively constructing projected databases and the FP-trees based on them.

III. Conceptual Design

A. Automatic Keyword Extraction

The task of automatic keyword extraction is to identify a set of words, representative for a document. To achieve this we use a simple statistical approach. Thereby, as we intend to exploit the properties of a document and of a repository, we need to find the comparable measures. One of the simple weighting is TF*IDF. The TF part intends to give a higher score to a document that has more occurrences of a term, while the IDF part is to penalize words that are popular in the whole collection. The further factors such as position of the word in a document or the length of a document is not comparable, as the database entries are much more shorter. Due to the type of extraction, we divide the automatic keyword extraction into 3 groups:

- Text – Based
- Database – Based
- Text – and Database – Based

B. Keyword Request Processing

In order to construct the structured keyword request for

an entity (i.e. individual, instance, "thing"), we first need to identify the attributes in which each keyword appears. This is performed in one step using an inverted index available in the entity repository Then the score is computed for every sub query q, which is a combination of an attribute a and a keyword *k* so that *q* = "k occurs in a". In our work we evaluate several attribute ranking approaches. In the next step, possible structured queries, each being a conjunction of sub queries, are constructed. Finally, these queries are ranked using query ranking criteria discussed in the following in Section and executed against the entity repository.

C. Attribute Ranking Factors

As our keyword request is a bundle of terms without the specification of attribute names, our first task is an identification of the attributes where each keyword appears in the repository. Then a specific score is computed for each attribute/keyword pair. The three intuitive and desirable constraints that any reasonable retrieval formula should satisfy are: term frequency *tf,* inverse term frequency *idf* and document length normalization *dl*. Applied to our attribute-specific approach, the *tf* heuristic intends to assign a higher score to an attribute of a single entity that has more occurrences of a query term. By intuition, in a collection, the more entities a term appears in a certain attribute, the worse discriminator it is, and it should be assigned a smaller *idf* weight. The attribute length normalization is to avoid favoring long attributes, as long attributes generally have more chances to match a query term simply because they contain more words. As to the entity representation considered in this work, its attribute values tend to contain in average about 2 words, so that the *tf* and *dl* score will have no effect, as the term usually appears only once pro attribute value and all attributes are approximately of the same length. Because of this as a basis for our score computing we use only the attribute-specific *idf* weight of a keyword, which is computed as follows :

Attribute specific IDF score (IDF):

$$IDF\ (keyword, attribute) = \log\left(1 + \frac{DF\,(attribute)}{DF\,(keyword\ ,\ attribute)}\right)$$

where DF(*attribute*) is number of entities containing the given attribute and DF(*keyword, attribute*) is number of entities where the given keyword appears in the given attribute. Attribute specific DF score (DF):

Opposite to *idf* , the method that is based on the probability of Key word match in an attribute can be used. The core idea of the df score is that probability of the match increases with increasing spreading of the keyword over the attributes. If the keyword appears in the given attribute more frequently than in other attributes than this attribute/keyword combination becomes the higher score than the others. The spread score is calculated according to the formula:

$$DF\ (keyword, attribute) = \frac{DF\,(keyword\ ,\ attribute)}{\sum_{attributes} DF\,(keyword\ ,\ attribute)}$$

where DF (*keyword, attribute*) is number of documents, where the keyword appears in the given attribute. The sum of the *df* scores of different attributes is 1.

D. Query Score

After obtaining the attribute-specific score for each attribute/ keyword combination, our next step lies in constructing the structured query for further request processing. The key idea is that a structured query is composed from subqueries using the and-semantics, corresponding to the "and" operator of the boolean model. Let *q1 ,..., qn* be a set of subqueries that represent the attribute/keyword combinations, a structured query Q is then defined as the conjunction of the subqueries $q1 \circ \ldots \circ qm$, $m \leq n$.The relevance of the whole query is represented as a sum of the scores of all subqueries.

$$\text{Score (query)} = \Sigma\ \text{Score(subquery q)}$$

where Score(*subquery q*) can be defined using a combination of the above attribute ranking factors. Typical combinations are: IDF, DF, IDF*avg(DAF), IDF*CAF, ARank, IDF*ARank.

E. Query Ranking

The aim of the query ranking procedure is to identify the structured query which delivers possibly precise results to the keyword entity requests. But the number of possible structured queries increases exponentially with the growing number of keywords and attributes in the repository. As for instance Figure 3.1 shows, we become 7 structured queries from only 3 sub queries. For that reason the construction and processing of all intended entity requests will be a very expensive and time consuming operation. The first native solution is to construct all possible queries, rank them before execution and process only the high-scored conjunctions. But typically, the number of queries is too high, such that it is infeasible to build and score all possible combinations. Following optimization algorithm to iteratively calculate the highly scored requests.

Given a a sorted sub query list *{ qn1 ... qnk }* for all occurrences of a keyword n in different attributes, we build a set *S = {{ q11 ... q1k }, ... , { qn1 ... qnk }}* for all keywords from *1* to *n*. Our task is to limit the number of queries to be constructed, as we are only interested in a few *top-k* highly scored queries.

For this purpose we introduce two bounds for the score of the query *Qtop-k*. The upper bound corresponds to the score of the query *Qk*, that consists of the sub queries *q* at *k* position. The lower bound is the sum of the scores of elements at the (*k+1*)-th position in each list. For a query *Qk* istrue:

$$score(Qk-1) > score(Qk) > score(Qk+1)$$

The intermediate scores are obtained due to the fact that some of highly scored elements at *k* position can build a number of highly scored combinations with the other lower scored elements in the lists. Due to this fact, a list of queries is constructed with the participation of the sub queries at the position *k*. The query *Q* is called *top-k* query when its score satisfies the condition:

$$score(Qk) \geq score(Qtop\text{-}k) > score(Qk+1)$$

Following figure is an example for constructing the top-1 queries. The list of constructed queries consists of 40 queries, but only 11 of them satisfy the score bounds and are considered as top-1 queries. With the native solution there would be 70 possible queries.

The requests with the highest scores are then executed till we obtain the intended minimum number of results. Algorithmically this method gives an advantage, especially if the length of the lists (number of attributes) is big.

The documents returned in a ranked list produced will be clustered according to the specifications set forth by the algorithm. The algorithm decides which cluster the documents belong in, and makes the distinction between highly relevant, partially relevant, and not relevant documents. The algorithm utilizes similarity measures and ranking heuristics that evaluate the relevance of a page. The high level workflow of the algorithm is in Fig.Each document in the collection is evaluated based on key similarity metrics and ranking heuristics. For each field, a relevancy grade is recorded for the document based on its satisfaction of the criteria listed for the given field and grade. A ranking function produces an overall score that combines the grades with the weight for each field for a given document and is explicitly detailed in a later section. The resultant score determines the cluster the document belongs in.

A ranking function produces an overall score that combines the grades with the weight for each field for a given document and is explicitly detailed in a later section. The resultant score determines the cluster the document belongs in. The fields, weights, ranking criteria, and relevancy grades used to deduce a score for each item is illustrated in Table 1. In Table 1, key fields within each document are identified and assigned a weight and relevancy grade. The motivation for choosing these fields is further identified in the next section. It represents the maximum number of words in the end-user query minus words that have no meaning. It is assumed that more than one key term is entered so that $\theta>1$. $C_1\theta$ and $C_2\theta$ represent constants with the values 2/3 and 1/3 respectively. Thus, C_1 and C_2 represent fractions of the number of query terms. In all cases, the resulting value is rounded to the nearest whole number. Thus, the notation $[C_1\theta,\theta]$, for example, represents the range in the number of query terms that must be included within the specified field. Y_3, Y_2, and Y_1 represent the number of occurrences of a query term in a document. The values used for this trial were $Y_3\geq_3$, $2\leq Y_2\leq 3$, $0\leq Y_1<2$. Note that W1, W2, and W3 are weights used to assign an importance value to each field, and were set to 0.214, 0.142, and 0.072 respectively.

The algorithm is given by

Step 1: The character is assigned with their location in the text.

Step 2: Merged same character information with a list of all locations information attached to each character.

Step 3: Characters location information are saved on the secondary storage as an index file. In general, the full text used a large-scale of electronic document so; the generated index becomes a large-scale too. Therefore, the character location information data group is arranged in ascending order to do the comparing processing for retrieving it at high speed.

As an example, suppose the end-user query consists of six distinct terms. Thus, θ is 6, $C_1\theta$ is 4, and $C_2\theta$ is 2 since C_1 and C_2 are set to 2/3 and 1/3 respectively. For the term frequency category, the document returned for the given query must contain between 4 and 6 query terms inclusive with each term occurring with a frequency of at least Y_3 in order to receive a relevant grade. To receive a partially relevant grade, the document must contain between 4 and 6 query terms with each term occurring with a frequency of Y_2. A grade of not relevant is given as the default case for a document that does not satisfy the relevant or

TABLE II FIELDS, WEIGHTS, AND RELEVANCY CRITERIA

ID	Field Name	Weight	Relevancy grade(λ)		
			[3]	[2]	[1]
1	Term Frequency	[W1]	$[C_1\theta, \theta]$ query terms each appear with frequency Y_3	$[C_1\theta, \theta]$ query terms each appear with frequency Y_2	$[C_1\theta, \theta]$query Terms each Appear with frequency Y1
2	Title	[W1]	$[C_1\theta, \theta]$ query terms appear	$[C_2\theta, C_1\theta]$ query terms appear	$[0, C_2\theta]$ query terms Appear
3	URL	[W1]	$[1, \theta]$ query terms appear	[1, 1] query terms appear	(0,0]query terms appear
4	Anchor text	[W2]	$[C_1\theta]$ θ, query terms appear	$[C_2\theta, C_1]$ query terms appear	$[0, C_2\theta]$query terms appear
5	Location	[W3]	Max Query terms in Top 1/3 region	Max Query terms in Mid 1/3 region	Max Query Terms in Bottom 1/3 region

partially relevant categories. Similarly, the document receives a relevant, partially relevant, or not relevant grade for each of the remaining fields. The motivation for using these fields, weights, constants, and criteria are delineated in the sections below.

F. Fields and Constant

The HTML makeup of page contains key fields that can indicate the importance of the document and improve retrieval. Intuitively, the title, six headings, and emphasized text such as bold, underline, and italic provide useful information about the page. Another heuristic that can play a significant role in retrieval effectiveness is location. The idea behind location is that a term near the beginning of the page may carry greater significance than terms lower on the page for term frequency, if a term occurs many times in the document; it represents the importance of the term within the page and may symbolize the importance of the term in the document.

G. Relevancy Grades

The relevance grades used in this study are derived from the notion of multi-graded relevance that is amply evident previous work. For each heuristic, a top grade is given assuming the document satisfies the necessary requirements to the fullest. This stringent criteria for each field is visible down the leftmost column of Table 1 under grade three. Similarly, a satisfactory grade is given when a document only partially satisfies the criteria. The requirements across each field are evident in the middle column of Table 1 under grade two. Finally, a low grade, which is displayed in Table 1 under grade one, characterizes non-relevant documents that either fail to meet the ranking criteria. Many documents may receive conflicting grades by satisfying relevant criteria in some cases, and partially or non-relevant criteria in other cases. Thus, these scores are aggregated into a weighted ranking function that combines individual scores as a weighted average to predict the most fitting category for the document, based on nature of relevance criteria within the document.

H. Cluster Interface

This interface illustrates how the document URL'S should be displayed to the user once the user submits a query. The clusters delineate the region of relevance each document belongs in. Documents within a specific cluster are not grouped internally according to relevance. Note that this interface represents the output of the system and was not shown to users for evaluation purposes.

I. Weights

Each measure and heuristic is given a weight to assign an appropriate importance value to the field. This value represents how much weight the field carries in assessing the relevance of a document and is included in out clustering scheme. In our algorithm, the assignment of weights to each heutristic is based on tiered-approach, where fields that are equally important are grouped together based on ad-hoc common sense and given a proportional weight in comparison to the other tiers.

J. Ranking Criteria

To determine the nature of the criteria in Table 1 that best fits relevant, partially relevant, and non relevant documents, previous work on document text characteristics was applied. Since our algorithm attempts to cluster according to regions of relevance, characteristics of relevant, partially relevant, and non-relevant regions serve as decisive factors within our ranking function. Highly relevant pages tend to discuss the topic at length, deal with several aspects of the topic, have many terms that pertain to the requested topic, and have many expressions to refer to the concepts discussed. Indeed, highly relevant documents often answer the users question, include the users search terms or concepts, are specific to the users query, and are authoritative sources. In contrast, partially relevant items tend to mention the topic only briefly. They contain only a few words matching the topic, and may discuss the topic from alternative viewpoints extending upon the original request. They often deal on partially with the subject, are not specific to the users query, and contain multiple concepts. Finally, non-relevant documents are often totally off target. This description of relevant documents, partially relevant, and non-relevant items can be translated into specified criteria that these classes of documents possess, as described in Table.

K. Ranking Function

Scores for each field in a document are aggregated to achieve a total overall score based on the weight of each field, and satisfaction of the criteria for each field. A weighted average consists of an estimation of the importance of every ranking factor through a weight proportional to the projected value. Thus, our ranking function combines the weights and relevancy grades received by a given document for each factor. The overall score is calculated as:

$$\mathrm{sc}(q,d) = \sum_{i=1}^{n} Wi \times \lambda i$$

where n represents the total number of ranking factors, W is the weight of each factor, λ is the relevancy grade received by a document for each factor, q is the query, and d represents the document. The score represents the category that best suits the document, and can fall in any one of three possible regions depending upon the characteristics of the document itself. The score category result for the document will fall in one of the following clusters by converting sc(q,d) to a region of relevance, namely Cluster(q,d).

$$\text{Cluster}(q,d) = \begin{cases} R & \text{if } \text{sc}(q,d) \geqslant \sigma f1 \\ PR & \text{if } \sigma f2 \leqslant \text{sc}(q,d) < \sigma f1 \\ NR & \text{if } \text{sc}(q,d) < \sigma f2 \end{cases}$$

The constant σ represents the maximum score possible from sc(q,d), *f*1 represents a constant factor of the maximum, and *f*2 represents a second constant factor of the maximum. The settings for the values used in our study allow for equal ranges that the score can fall within for each of the three regions of relevance. The constant values are 3 for σ since the maximum possible score according to sc(q,d) is 3, 7/9 for *f*1, and 5/9 for *f*2. The lowest possible score according to sc(q,d) is 1. As a result of this equation, every document d for a query q will be placed in either the relevant, partially relevant, or non-relevant cluster.

L. Indexing

Search engine indexing entails how data is collected, parsed, and stored to facilitate fast and accurate retrieval. Index design incorporates interdisciplinary concepts from Linguistics, Cognitive psychology, Mathematics, Informatics, Physics, and Computer science. An alternate name for the process is Web indexing, within the context of search engines designed to find web pages on the Internet. Popular engines focus on the full-text indexing of online, natural language documents, yet there are other searchable media types such as video, audio, and graphics. Meta search engines reuse the indices of other services and do not store a local index, whereas cache-based search engines permanently store the index along with the corpus. Unlike full text indices, partial text services restrict the depth indexed to reduce index size. Larger services typically perform indexing at a predetermined interval due to the required time and processing costs, whereas agent-based search engines index in real time.

IV. Proposed Work

A. Initial Cluster Generation

At this step the input is analyzed, initial clusters are produced and outliers are removed. The first thing for soft Clustering to do is to decide what constitute as "similar" documents. Essentially, we need to find a threshold value λ such that two documents are considered similar if and only if f (x, y) ≥ λ . Since soft clustering is designed to adapt to different similarity measures f, it is not reasonable for the user to supply a value for λ . As a result, SISC determines the appropriate value of λ to be based on the input documents. The value of λ can neither be too high, such that no documents will be clustered at the end; nor too low, such that all documents will be clustered into one cluster. Thus, the algorithm chooses λ such that half of the documents are assigned at least to one cluster centroid. This is done by the following method:

- Pick a set of k documents, assigning each one as the initial cluster centroid of a cluster.
- Pick λ as the largest value such that for half of the documents q in the data set, there exists a p such that f (p , q) ≥ λ ,p ϵ C , q ϵ D where C is the set of cluster centroids and D is the document set. This can be done by calculating all the similarity values f (p , q), ∀p ϵ b. Pre-Processing C, ∀q ϵ D and sorting them.

B. Pre-Processing

Electronic documents cannot be applied to a computer algorithm in their natural state; some form of processing is required to put them into a structure that can be used by the algorithm. The most common statistical method of document representation is the Vector Space Model. The basic VSM involves storing documents as vectors in which each element corresponds to the frequency of a term in the document. Essentially this model provides a 'bag of words' in that the positioning of each term is ignored. Usually the words are weighted according to their power of discrimination between topics, i.e. words that have limited discrimination power need to be de-emphasized and vice-versa.

The first step in reducing the dimensionality of the entire document collection is by feature selection. Words that are believed to contain little or no meaning can be removed from the list of candidates. These terms make what is commonly referred to as a stop list and typically contain prepositions, pronouns, articles and other non- descriptive words, for example, 'the', 'at' or 'into'.

In natural language processing, conflation is the process of merging or grouping together non-identical words that refer to the same principal concept. A word stemming algorithm can be used to conflate terms, for example, by removing any attached suffixes (for example '-ly', '-ness' or '-ment') and in some cases prefixes (such as 'anti-, 'bi-', or 'semi-') from terms. This can be a useful tool in dimensionality reduction since the stem of a term represents a broader concept than the original term. For example 'employing', 'employs' and 'employed' have the same stem 'employ'.

Pre-set thresholds could be used to remove terms appearing in more than an upper bound and less and a lower bound number of documents. The ability of words to discriminate content reached a peak at a rank order position half way between the two cut offs positions and fell off to near zero, from the peak in both directions, when nearing the cut off points. Rare words as well as very common words are assumed to contain very little information. Hence, it is safe to remove them and thus significantly reduce the dimensionality of the input space without compromising the text classification performance of the system.

C. Top-K Cells Retrieval

One key question in document retrieval is how to rank documents based on their degrees of relevance to a query. Much effort has been placed on the development of ranking functions. Traditionally, document retrieval methods only use a small number of features. Thus, it is possible to empirically tune the parameters of ranking functions. In that sense, the methods are unsupervised and language models for information retrieval are such methods. Currently, additional features have proved useful for document retrieval, including structural features and query-independent features. This increase in features makes empirical tuning of parameters difficult. The paradigm of employing supervised learning in construction of document retrieval models has drawn recent attention. For instance, document retrieval is formalized as classification. Documents are judged within two categories: relevant and irrelevant. Formalizes document retrieval as binary classification and solves it using SVM and Maximum Entropy propose employing discriminative training in creating a ranking model. For another example, document retrieval is regarded as learning to rank.

D. Feature Extraction

For each document, the extraction process was as follows:

- The set of all words that appeared at least once in the document was extracted – this was labelled as the "All Words" set.
- An additional set of words was also extracted from each document – those occurring at least once between the opening and closing "Heading 1" text decoration tags (<H1> … </H1>). This set of extracted words was labelled as the "H1 Tag" set.
- If stop-word removal was switched on, we removed any word from the appropriate set of extracted words that was listed in our stop-word list, we used V. .
- If "case-sensitive" processing was switched off, words with the same spelling (but different case) were combined, if switched on, these words were considered as not being the same.
- The frequency of each resulting word in that document was then recorded and stored.

Once each set of words has been extracted from each document, the next step is to combine one of the extracted sets into a master word list from which each document vector can be built. The resulting list is sorted in descending overall frequency, and the "all words" list is then pruned by selecting only the top 1% of most frequently occurring words. The decision to prune the master word vector leads to a decreased size of each document's vector, which leads to a decreased time both in terms of building the vectors, but also in clustering them. A very important point is that the "H1 tag" master word list was not pruned, seeing as it comprises significantly fewer words. Clearly, we have a choice of which extracted word set, either the "all words" set or "H1 tag" set, to use when constructing the master word list. The choice we employ at this point is based on the set-up of each individual experiment. Finally, each feature vector v_i was created for each document i, such that the j^{th} element in v_i was w_{ji}/s_i, where w_{ji} is the number of occurrences in the specified set of extracted words belonging to document *i* of the j^{th} most frequent word in the chosen master word list, and s_i is the total number of words from the specified set of extracted words from document i. We have taken the step to "normalise" each document vector by dividing each term by the total number of words in the document. This ensures that large documents do not have undue influence over smaller documents that may contain the same terms, but just less frequently. In summary, the process of building the document feature vectors requires a choice regarding which set of extracted words to use at three distinct stages. These stages are the creation of the master word vector, the set of document words from which to reference each element in the master word vector and finally the "normalisation" factor to use.

V. Recall And Precision

Recall and precision are two techniques, which are simple to use and are based on predicate logic. Imagine a user makes a request for information I on a set of all relevant documents R. The set of all relevant documents is information retrieval system has processed the query and has come up with a set of answer documents A. This set of answer documents is the circle on the right. The intersection of these two circles is the part we are interested in. This intersection, lets call it Ra, is the set of documents in the answer set that are relevant to the user. Therefore we can define recall as: -

The fraction of the relevant documents which has been retrieved.

$$\text{i.e. Recall} = \frac{|Ra|}{|R|}$$

And we can define precision as: -
The fraction of the retrieved documents which are relevant.

$$\text{i.e. Precision} = \frac{|Ra|}{|A|}$$

Plotted using Venn diagram below:

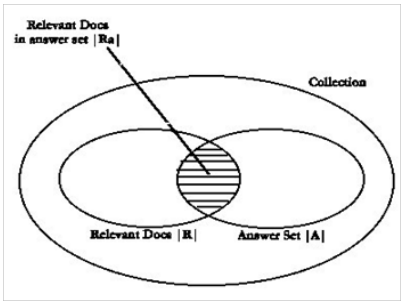

Fig. 2 Venn diagram for precision and recall

We can see from this that if there was no intersection between the two circles, then there would be no relevant documents in the answer set and therefore Precision would be zero. However, if the answer set circle wholly covered the relevant documents set, then Precision would be one as all of the answer documents are relevant. Standard recall levels of text retrieval with and without document preprocessing techniques, with the whole set of document collection for indexing and all the queries for performance evaluation.

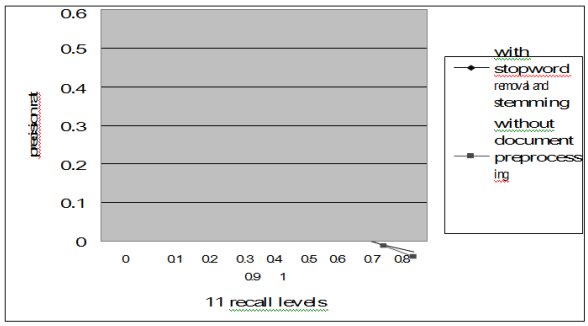

Fig. 3 Precision vs Recall

VI. Conclusion and Future Work

This research showed that clustering documents on the Web by their regions of relevance is not only feasible, but also quite successful. Our clustering scheme offers an accessible, systematic, and versatile approach towards retrieving and organizing search results to enhance the way in which users of all domains meet their information seeking goals. Since partially relevant documents are useful for novice users at the beginning stages of their search, these documents are now clearly identified and grouped together. Likewise, expert users that have a clear idea of what they are seeking, can efficiently access the documents within the relevant cluster with our scheme.

The research presented in this study can be extended in numerous directions.

- The algorithm can be embedded directly within a major Web search engine clustering scheme so that it can be fully operable on of the Web.
- Within each cluster, results can be ordered so that end-users could more selectively target potentially useful documents within each cluster.
- The number of clusters could be expanded to create an even more fine-grained clustering system.

References

[1] Cindy Xide Lin et al. Text Cube: Computing IR Measures for Multidimensional Text Database Analysis. In ICDM '08: *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining,* pages 905–910, Washington, DC, USA, 2008 IEEE Computer Society.

[2] Yintao Yu et al. iNextCube: *Information Network-Enhanced Text Cube. Proc. VLDBEndow.,* 2(2):1622–1625, 2009.

[3] Duo Zhang et al. Topic Modeling for OLAP on Multidimensional Text Databases:Topic Cube and Its Applications. Stat. Anal. Data Min., 2(56):378–395, 2009.

[4] S. Chaudhuri, R. Ramakrishnan, and G. Weikum, "Integrating db and ir technologies: What is the sound of one hand clapping?" in *Proc. Conf. on Innovative Data Syst. Research* (CIDR), 2005, pp.1–12.

[5] S. Amer-Yahia, P. Case, T. R¨olleke, J. Shanmugasundaram, and G. Weikum, "Report on the db/ir panel at sigmod 2005," SIGMOD Record, vol. 34, no. 4, pp. 71–74, 2005.

[6] G. Weikum, "Db&ir: both sides now," in Proc.ACM SIGMOD, 2007, pp. 25–30.

[7] S. Agrawal, S.Chaudhuri, and G. Das, "Dbxplorer: A system for keyword-based search over relational databases," in *Proc. IEEE Int'l Conf. Data Eng.* (ICDE), 2002, pp. 5–16

[8] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury, "Effective keyword search in relational databases," in Proc. ACM SIGMOD, 2006, pp. 563–574.

[9] Y. Luo, X. Lin, W. Wang, and X. Zhou, "Spark: top-k keyword query in relational databases," in Proc. ACM SIGMOD, 2007, pp. 115–126.

[10]]G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using banks," in *Proc. IEEE Int'l Conf. Data Eng.* (ICDE), 2002, pp. 431–440.