An Application of Statistical indexing for Searching and Ranking of documents – A Case Study on Telugu Script

N.V. Ganapathi Raju
Associate Professor, Dept of CSE
GRIET, Hyderabad

Bhavya Sukavasi
PG Student, Dept of MCA
GRIET, Hyderabad

Sai Rama Krishna Chava
PG Student, Dept of MCA
GRIET, Hyderabad

Vidya Rani Vadisala
PG Student, Dept of MCA
GRIET, Hyderabad

ABSTRACT
The lack of proper indexing and ranking techniques for Telugu documents motivated us for implementation of this work. The results provide a way to analyze efficiency of algorithms used for indexing and ranking. This paper summarizes the Automatic Term Weighting and Inverted File Structure approaches for Telugu documents and provides baseline of single term indexing to develop more elaborate techniques like content based analysis.

General Terms
Term Weighting, Cosine similarity, Stop words, Corpus

Keywords
Statistical indexing, Ranking, Inverted File Structure, Stemming, Telugu Documents, Term weighting.

1. INTRODUCTION
Information retrieval (IR) corresponds to representation, storage, organization, and access to information items. Information can be composed of text images, audio, video and other multi-media objects. Usually information retrieval is viewed upon as a circular procedure, where the user makes a request for information to a system and recursively evaluates the response until the information need is fulfilled. The user's request is compared with a description of stored items in the system. Here description of stored data means index file. Usually the system makes some automatic relevance assessment of the documents retrieve and presents to the user in descending order of estimated relevance.

The general objective of an Information Retrieval System is to minimize the overhead of a user locating needed information. Overhead can be expressed as the time a user spends in all of the steps leading to reading an item containing the needed information (e.g., query generation, query execution, scanning results of query to select items to read, reading non-relevant items).

Many researchers have been worked in the area of I.R. Among them Gerald Salton, Ted Nelson are the main contributors in this area. Gerald Salton was father of modern search technology. His teams developed SMART information retrieval systems. Salton introduced concepts like vector space model, Inverse Document Frequency (IDF), Term Frequency (TF), term discrimination values, and relevancy feedback mechanisms. Ted Nelson created Project Xanadu in 1960 and coined the term hypertext in 1963. His goal with Project Xanadu was to create a computer network with a simple user interface that solved many social problems like attribution. [4]

2. AUTOMATIC INDEXING
Indexing is the act of extraction of terms from a document to indicate what the document is about or to summarize its content. Indexing is very important in the context of information retrieval as it decrease the time of searching and is basis for ranking the retrieved documents. Automatic indexing is the process of analyzing an item to extract the information to be permanently kept in an index. Figure-1 shows the overall indexing process including Identify Processing Tokens, Apply Stop Lists, Characterize tokens, Apply Stemming and Create Searchable Data Structure is all part of the indexing process. From many approaches of automatic indexing this paper discuss about inverted file structure, statistical indexing by using term weights which are used for searching and ranking of Telugu Documents. [4]

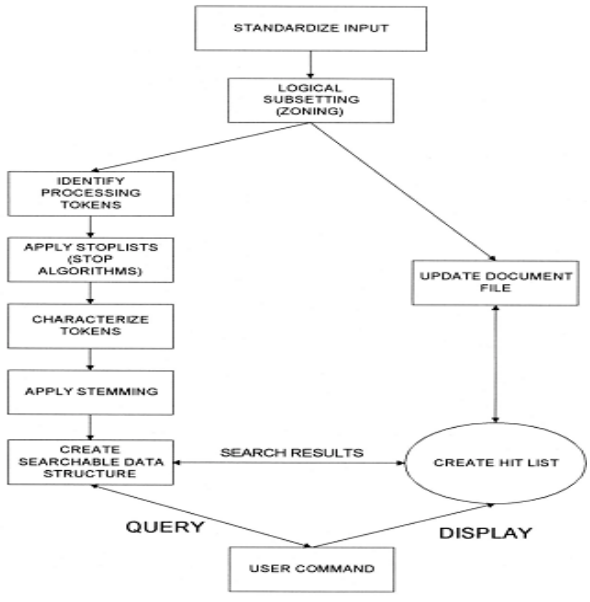

Fig 1. Indexing Process

2.1 Inverted File Structure

An Inverted File Structure contains, for each term in the lexicon, an inverted list that stores a list of pointers to all occurrences of that term in the main text, where each pointer is, in effect, the number of a document in which that term appears. The concept of the inverted file type of index is as follows. Assume a set of documents. Each document is assigned a list of keywords or attributes, with optional relevance weights associated with each keyword (attribute). An inverted file is then the sorted list (or index) of keywords (attributes), with each keyword having links to the documents containing that keyword index found in most commercial library systems. The use of an inverted file improves search efficiency by several orders of magnitude, a necessity for very large text files. The penalty paid for this efficiency is the need to store a data structure that ranges from 10 percent to 100 percent or more of the size of the text itself, and a need to update that index as the data set changes [4]
Example for building Inverted File Structure

Table 1. Document Sets

Documents	Text
1	బీపీఓ కేంద్రంగా భారత్
2	టెలిఫోన్ బూత్ లో బ్యాంక్
3	న్యూడిల్లీ బ్యాంక్ అకౌంట్ పబ్లిక్ టెలిఫోన్ బూత్
4	బ్యాంకులు టెలికామ్ కంపెనీల

Then we build the corresponding index.

Table 2. Inverted File Structure

Documents	Term	Times; Documents
1	బీపీఓ	<1; 1>
2	భారత్	<1; 1>
3	కేంద్రంగా	<1; 1>
4	టెలిఫోన్	<2; 2,3>
5	బూత్	<2; 2,3>
6	బ్యాంక్	<3; 2,3,4>
7	న్యూడిల్లీ	<1; 3>
8	అకౌంట్	<1; 3>
10	పబ్లిక్	<1; 3>

2.2 Term Weighting

In the late 195Os, Luhn first suggested that automatic text retrieval systems could be designed based on a comparison of content identifiers attached both to the stored texts and to the users' information queries. In either case, the documents or user queries would be represented by term vectors of the form

A formal representation of the term vectors is obtained by including in each vector all possible content terms allowed in the system and adding term weight assignments to provide distinctions among the terms. Thus, if Wk represents the weight of term tk in document D (or query Q), and t terms in all are available for content representation, the term vectors for document D and query Q can be written as

D = (t0, w0; tl, w1; . . . ; tn,, Wn)

and

Q = (q0, w0; ql, w1;. . .; qr,wr).

Here wt represents is TF-IDF of the term t. TF term frequency refers to the number of times the word present in that particular document. IDF is ratio of total number of documents to the number of documents in which the word is present. TF-IDF is the product of TF and IDF [1]

2.3 Ranking

A by-product of use of similarity measures for selecting Hit items is a value that can be used in ranking the output. Ranking the output implies ordering the output from most likely items that satisfy the query to least likely items. This reduces the user overhead by allowing the user to display the most likely relevant terms first. The original Boolean systems returned items ordered by date of entry into the system versus by likelihood of relevance to the user's search statement. With the inclusion of statistical similarity techniques into commercial systems and the large number of hits that originate from searching diverse corpora, such as the Internet, ranking has become a common feature of modern systems. To rank a document retrieved by a query similarity between them has to be calculated. The below formula is used to measure similarity between query and item.

$$similarity(Q,D) = \frac{\sum_{k=1}^{t} w_{qk} \cdot w_{dk}}{\sqrt{\sum_{k=1}^{t} (w_{qk})^2 \cdot \sum_{k=1}^{t} (w_{dk})^2}}$$

3. IMPLEMENTATION

We used Python and Java programming languages for implementation of this work. Python is an interpreter, interactive, object-oriented, extensible programming language. There are two types of strings in Python: byte strings and Unicode strings. Python handles Unicode strings same as that of byte strings. Unicode strings are encoded in UTF-8 format. Python has codecs module which convert UTF-8 encoded byte strings to Unicode strings. We used Java (JDBC) to make database operations using MS Access.

The system can be divided into following modules:

- Collection of data from the web
- Preprocessing and extraction of the words from the corpus
- Applying stop word removal
- Applying N-gram based Stemming process
- Constructing of Inversed File System based on sorted array
- Developing the Index Tables
- Creating searching process
- Creating ranking process

3.1 Collection of data from the web

We collected the Unicode Telugu data from Telugu daily news articles like www.unimedhas.org. The collected Telugu data is classified manually into various categories like literature, politics, science, sports, business, rivers, and editorial.

3.2 Preprocessing and Extraction of Words from the Corpus

In preprocessing we removed or ignored the characters which are other than the Telugu language. All the numbers, special characters, and any unwanted letters except "spaces" will be removed. From the preprocessed file we will extract the words using a space identifier which separates each word.

3.3 Applying Stop Word Removal

It has been recognized since the earliest days of information retrieval that many of the most frequently occurring words in any language are worthless as index terms. A search using one of these terms is likely to retrieve almost every item in a database regardless of its relevance, so their discrimination value is low. Furthermore, these words make up a large fraction of the text of most documents: the ten most frequently occurring words in any language typically account for 20 to 30 percent of the tokens in a document. Eliminating such words from consideration early in automatic indexing speeds processing, saves huge amounts of space in indexes, and does not damage retrieval effectiveness. A list of words filtered out during automatic indexing because they make poor index terms is called a stop list or a negative dictionary. [5]

Stop words are words which have very little informational content. These are words such as: గురించి, ఎంత, ఎందుకు, ఒక, అవి, ఎళ్లు, ఎారు etc. Here we remove the words such as articles, Prepositions, conjunctions etc. from the documents. For this we collected 450 Telugu stop words and kept in a stop list file.

3.4 Applying N-gram based Stemming Process:

Stemming is common form of language processing in most, "A failure to process morphological variants results in retrieving only 2% - 10% of the documents retrieved with such processing".

In this step we develop three dictionaries having key as stem bigram/trigram/quadgram) and value as words for those corresponding N-grams.

- Here we will give priority order as quadgram to bigram
- Starting with quadict if a key is quadict having two or more words as value then we will consider the key as stem and push into a stemmed dictionary.
- The above step is repeated for tridict and bidict.
- Now stemmed dictionary containing some key value pairs by giving higher priority to quadgram we will remove if any other values in the dictionary contains same words which under come in the values of quadgram.
- If a word is exactly of length 2 or 3 or 4 and doesn't have any morphological variants then that word itself is considered as root.

3.5 Constructing Inverted File Structure Based on Sorted Array

Based on the sorted array technique the inverted file structure with unique words as well as stemmed words was built.

- All the unique words from all documents are collected in sorted order.
- For each word all the documents are compared to find the same word.
- All documents containing that word are putted on the sorted list and written to invert file.

3.6 Developing the Index Tables:

Ranking algorithms are based on Statistics like weights. For the maintenance of these statistics we should maintain a table for each document and calculation of TF-IDF values are necessary.

- For the insertion of values into a table corresponding to file we process count file associated with it.
- These values are then inserted into the file by systematically processing count file.
- Once all the training documents have its associated initial tables we can calculate Inverse Document Frequency (IDF) and Term Frequency Inverse Document Frequency (TF-IDF).

3.7 Creating Searching Process

Searching is the important in the system, we retrieve information based on the search process. This technique gives results based on the query what we given. Finally the related documents should be displayed on the Search Process window.

- Tokenize the given query.
- Then store those words in the list.
- Applying searching process based on the given query and finding related documents to those words in the inverted file system.
- Finding all related documents and write to result file.

3.8 Creating Ranking Process

Ranking is the final module of the system. This module sorts the retrieved documents based on their relevance to the query.

- Ranking is done in two phases called coarse grain ranking and fine grain ranking.
- In the coarse grain ranking the documents are sorted depending on the frequency in result file
- In the fine grain ranking each set of documents same number of frequencies are again sorted depending on the similarity measure. We use cosine similarity here

$$similarity(Q,D) = \frac{\sum_{k=1}^{t} w_{qk} \cdot w_{dk}}{\sqrt{\sum_{k=1}^{t} (w_{qk})^2 \cdot \sum_{k=1}^{t} (w_{dk})^2}}$$

4. ALGORITHMS USED

4.1. Algorithm for Inverse File Structure.
4.2. Algorithm for Term Weighting.
4.3. Algorithm for Search Process.
4.4. Algorithm for Ranking Process

4.1 Algorithm for Inverse File Structure:

Read all the files from a category

Preprocess all the files

Eliminate redundant words and sort unique words write them to sort file

Applying Stemming Process and finding Stemmed words

For each word in sort file

For each document in corpus
If word exists in the document
Write document name to Inverted file system
Create inverted file system to all categories

4.2 Algorithm for Term Weighting:

For each category
Preprocess all file and generate words.txt file
Eliminate stop words from words.txt file
Stem all words by using N-gram based Stemming
Create a count file showing frequency of each stem word for each file
Insert count file into database
Calculate inverse document frequency
Calculate TF/IDF and insert into data base

4.3 Algorithm for Search Process:

Read the given query
Tokenize the given query into words save it on a list
For word in the list
If word not a special character
Open invert index file
If word exist in the file
Retrieve the sorted document list for the word
Close the indexed file

4.4 Algorithm for Ranking Process:

Read the result file
For each line in the file
Tokenize the line and insert into the array of strings
Collect the string containing document name as its value
Find frequency of each document in that collection into another array
Sort the Document String array depending on their associated frequency array
For each value in Document string array
Calculate similarity using Cosine Similarity and store them.
For each document having same frequency
Sort the document based upon similarity value
Write the sorted array to the HTML file for browsing.

5. SCREENS:

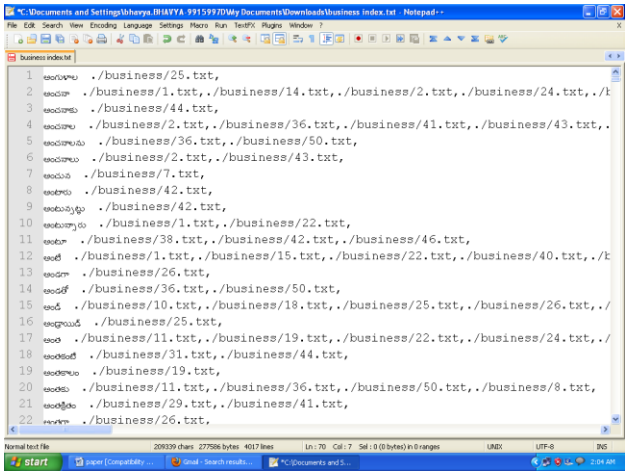

Fig 2. A sample invert index file

Fig 2 shows the Inverted File index created in which each word is having a list of file names in which it is present. This structure is very much useful in retrieving the relevant documents. The file is sorted in the alphabetical order.

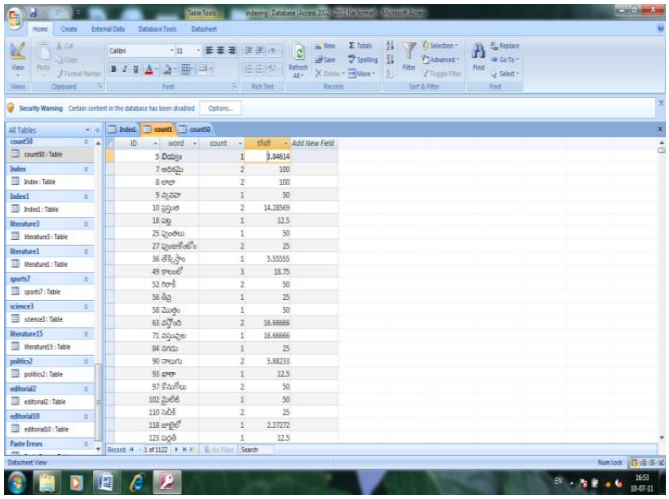

Fig 3. A sample index file

Fig 3 shows the index file generated in MS – Access in which for each word TF-IDF values are calculated and are inserted into the file.

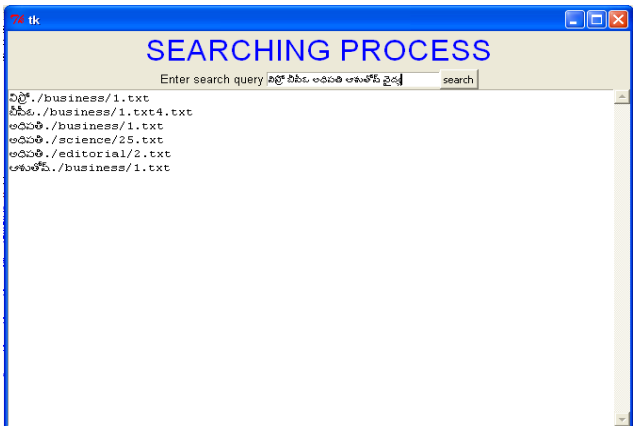

Fig 4. Displaying Results

Fig 4 shows the query in text box and results associated with the query which are result of search process. The lists of files shown have redundant file names which are to be eliminated in ranking process.

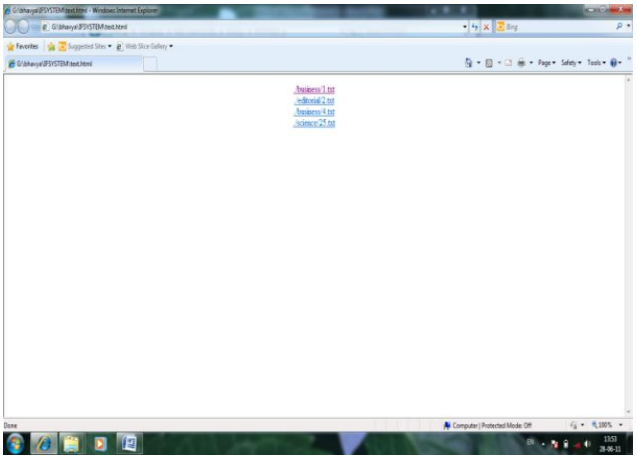

Fig 5. Browser window displaying documents

Fig 5 shows the HTML file generated as a result of ranking process opened through a web browser which contains ranked documents to the given query. The documents listed by using anchor tags through which we can see the document retrieved.

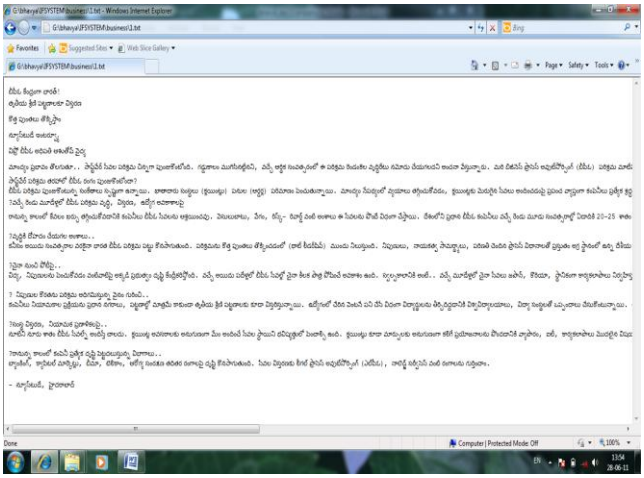

Fig 6. Displaying Telugu document in a browser

Fig 11 shows the Telugu document opened via anchor tag. The browser displays corresponding document to that tag.

6. Results:

For the random queries results are collected and are tabulated in the ascending order of number of documents retrieved. Here we considered Number of Possible Relevant as 50 because we have 50 documents in each category and they can be only possible relevant documents for any query related that category. And for all queries The Number of Retrieved Documents and The Number of Retrieved Relevant documents are calculated manually. The table obtained is as follows:

Table 3. Precision and Recall

S.I	Number o document retrieved	Number of documents Relevant	Number o Retrieved Relevant	precision	Recall
1	3	50	3	1	0.06
2	5	50	4	0.8	0.08
3	10	50	6	0.6	0.12
4	20	50	10	0.5	0.2
5	65	50	25	0.38	0.5
6	100	50	35	0.35	0.7

Graph is drawn for obtained values of precision and recall which is as follows:

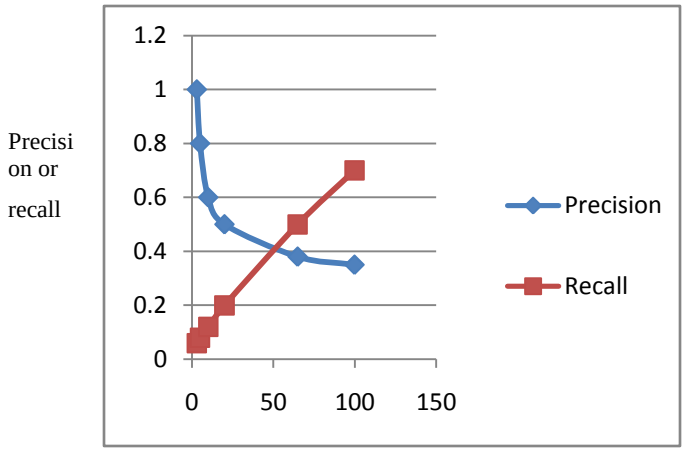

Fig 7. Graph of Precision and Recall

The X – axis shows Number of Documents Retrieved and the Y– axis shows precision or recall values. We can clearly observe from the graph that the at less number of documents retrieved the precision is high and at more number of documents retrieved recall high. So we can conclude that as the number of documents retrieved increases precision decreases and recall increase.The ideal behavior of precision and recall.

The overhead can be reduced by the implementation of ranking. Ranking is done in two steps and the overhead is calculated without ranking and at each step of ranking implementation and the results are tabulated as below.

Table 4. Table of overhead values

Number of documents retrieved	Over head in seconds in average		
	Without ranking	With coarse grain ranking	With fine grain ranking
5	30	15	10
10	40	18	10
15	45	22	10
20	55	25	10
25	63	28	10

In Coarse grain ranking depends upon completeness of the document i.e. the document containing all query terms will be ranked first. The fine grain ranking comes into picture when two or more documents ranked to a single position in first step i.e. coarse grain ranking. The fine grain ranking depends upon weights of terms (TFIDF). In this phase actual similarity function is calculated between document and query.

The above results correspond to each of these steps i.e. without ranking, after coarse grain ranking, after fine grain ranking.

The results are converted into a graph for easy analysis. The graph is shown below

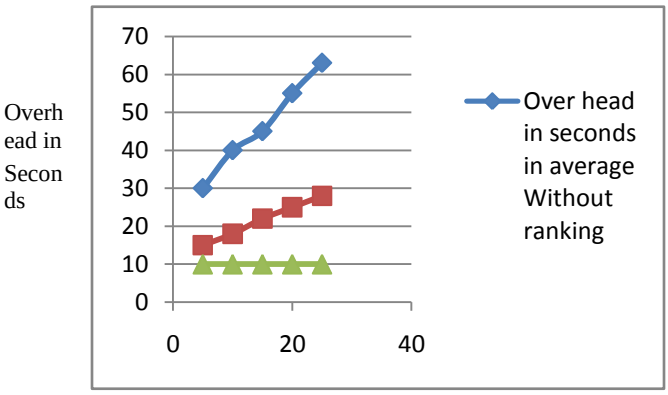

Fig 8 Graph for overheads at different steps of ranking

The graph with number of documents retrieved as X – axis and overhead as Y – axis.

7. CONCLUSION

This system covers major part of search engine implementation like stop word removal, stemming, Automatic Indexing, searching. To make this system a complete search engine we could add other parts of it like clustering, thesaurus expansion. We could implement this system for any Indian Language whose canonical structure resembles Telugu. This system takes a lot of time for updating a new document by changing few implementation strategies to make this faster.

8. ACKNOWLEDGMENTS

We would like to express our gratitude to Dr.B.Vishnu Vardhan for his valuable suggestions.

9. REFERENCES

[1] Gerald Salton and Christopher Buckley "Term-Weighting approaches in automatic text retrieval".

[2] Ma, W., Zhang, H. and Hon, H. (2004). "Towards Next Generation Web Information Retrieval".

[3] Wall, A. (2004). "History of search engines & web history"

[4] Gerald J.Kowalski, Mark T. Maybury. "Information Storage and Retrieval Systems Theory and Implementation"

[5] Williams B. Frakes and Ricardo Baeza- Yates, " Information Retrieval: Data Structures & Algorithms"

[6] Glenda Browne, Jon Jermey. "The indexing companion."

[7] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. "Modern Information Retrieval"