**Problem statement**

Recommending new books that are similar to the book that was queried by the user by using a ranking measure for retrieved results from the corpus.

**2. Background**

Unlike regular information retrieval systems, the queries for book retrieval and recommender systems are very specific, i.e., they contain words related only to the title of the book and its author. Also, the number of matches for a well formed query (which has the most of the words in the title as well as the author(s) name(s) ) are very few. Since only very little information is required to identify the correct book based on the user query, an index using only words from the book titles and authors can be created. Thus, each book title and its corresponding author(s) can be considered as a single document, and these documents can be used to search for books.

Often, the user gives an ambiguous query which does not contain the entire name of the book or the author's names. In such cases, a robust system should be able to retrieve books that the user could have been searching for, so that he/she can select the correct book. Ranking of the retrieved results is required here so that the most appropriate results are displayed at the beginning to save the user's time while searching. A number of ranking schemes have been proposed and used with a lot of success, but it has been seen that different ranking schemes have varying performances on different collections. As the document size is extremely small, the most common ranking method - tf-idf - will not perform well as the within-document frequency will not play any role, and might even harm the performance of the system. In a subsequent section, a different ranking scheme (proposed by Croft and Harper) has been described, which is suitable for retrieval of short documents. (which is the combination of positional indexing scores)(Harper et al., 1979)

**a. Recommender Systems**

Data is getting created constantly, and at an ever increasing rate. In order for it to be fit for user consumption, the data needs to be processed thoroughly. A more mammoth task is to recommend this processed data to the user.

This is where Recommender Systems come into the picture. Simply speaking, recommender systems are the systems which recommend data based upon user's input. Recommender Systems are used by companies to improve their user response by recommending them data which matches their informational needs. It becomes more important to do this in the time of big data where most of the data is in the form non-structured data, which in our case is text.

**b. Motivation**

Most of the times, people after reading a good book , tend to long for a similar book as they want to relive the experience which they recently had. In order to do that one needs a similar book which could give the same thrill while reading as before. Thus, we found it quite useful to help in such a case. We hope that this project could aid the user in finding the book which his heart desires.

**C. Technical issues**

1) **2 Indexes** - Due to the different needs of the indexes for searching for the query as well as for the similarity measure, the terms would have to be different. Thus, we needed two indexes, it was unavoidable.

2) **Crawling Amazon** - Amazon had security measures against web crawlers i.e captcha if a abnormally high number of requests from a IP. We resolved it by getting the same data via GoodReads.com .

3) **Avoiding Links** - The biggest challenge with web crawling is avoiding links that point to unwanted sites/images etc. So to solve this problem, instead of blindly calling all the links, the Spider has a predefined list of links that it needs to crawl systematically and retrieve the required data.

4) **Presence of near-duplicates** - GoodReads has a lot of books stored in it. As we crawled page by page so we encountered duplicates with different ISBN code which makes it very expensive to remove near-duplicates.

**3. Related Work:**

A significant amount of work has been done in the recent past on document retrieval and ranking. One of the most used methods for ranking of documents represented as vectors is by calculating the cosine similarity of tf-idf scores (Manning *et al.*, 2008). Though this simple model has had a lot of success, it has certain drawbacks (Yih, 2009). One of its major limitations is that it does not take the relative positions of the words into account, i.e., it treats both the documents and the queries as "bags-of-words". It is also not domain dependent, and thus cannot be easily adjusted for a particular application.

In the current problem, very short documents (containing only the book title and the author(s)) have to be retrieved. It has been seen that the similarity measures used for documents of significant length do not work well for short documents due to data sparseness (Metzler *et al.*, 2007). Here, within-document term frequency does not play much of a role because the document size is so small that the chance of a word getting repeated is rare (unless it is a stop word). Thus idf scores are more informative in this case (Frakes et al).

Croft *et al.* (1979) proposed a probabilistic method to measure the similarity of two documents that does not require relevance information. Experimental results have shown that this performs better than when only IDF scoring is used. Another advantage of this method is that it can be modified to suit different kinds of collections. By setting the parameter 'C' one can adjust the scores assigned based on the type and length of documents that are present in the corpus being used for the study. The similarity measure had been tested with the UKCIS2 collection (Harper 1980) which contains only document titles and it was found that a low value of C gives good results (Frakes et al).

**4. System Description**

**Web Crawler**

The HTML of page is retrieved by sending a request from the crawler with a header indicating that it is a browser

**WEB CRAWLER BLOCK DIAGRAM**

Scrapper uses the XPATH of the required elements to fetch the individual elements

The elements are grouped together into a data-block and appended into a variable to be dumped into a json file

Multiple instances of the scrapper are run in parallel

The required elements are now parsed for special characters and made into strings

**Note:** XPATH IS THE PATH OF A DOM ELEMENT OF A WEBPAGE AS STORED IN XML

Web crawling has been the go-to way for collecting information when a dataset is not readily available for us. Due to this reason, our model demands crawling the web to retrieve a large quantity of dataset (that includes the following: Name of the book ,author, Category, Rating, ISBN).

We use a Scrapper to do it. The Scrapper uses the Xpath of the required elements to fetch the individual elements. The data has been stored in json format.

\*Xpath- it is the path of a DOM element of a web page as stored in XML.

To handle the bottleneck of the delay of sending a request and receiving data, multiple instances of the code are run in parallel.

**Index Construction**

Collect documents from corpus containing all parameters

**INDEX CONSTRUCTION BLOCK DIAGRAM**

Remove symbols which are neither alphabets nor numbers

Convert the parameters to lower case and tokenize them

Simultaneously map DocID with the document

For each document index them using inverted index with term-DocID mapping term frequency also in the list

Index construction consists of mainly four parts

The problem statement demands the construction of two indexes :

1) For searching the query which requires the use of only author name and book name.

2) For computing the similarity graph between two books which requires the use of all the features extracted.

It also requires the use of a mapping from DocID to Document in order to identify Document.

1) The index is a simple inverted index between term and DocID. The terms (author name and book name) are extracted from the crawler data(which is in the form of a json file) .

2) The index is a between terms and DocID. The terms are author name, book name, Ratings, Category, and plot summary .

**a)Collect documents to be indexed**

The collection of documents to be indexed is common for both the indexes. The data is collected by using the in-built json library of python which directly converts the file into a list of hash tables.

From then on, we choose the documents which have all parameters present. We put the OR condition in the 'for loop' to only select the documents which satisfy the non-empty condition.

**b)Tokenization**

The two indexes have similar tokenisation technique's. The tokenizer splits the string based on space present.

For the first index, the name entry for both book name and author name needs to be tokenized whereas for the second index, the system is tokenizing every parameter extracted except rating which is a single value and thus does not need to be tokenized.

**c)Linguistic processing**

We are using porter stemming to process the summary parameter because stemming improves precision as it expands on the document terms. Later we also have removed the duplicates from these summary terms as duplicates will not lend any semantic value.

**d)Indexing**

There are total of three hashtables present in the code , two are for indexing terms to DocID and the third one is for mapping DocID's to Documents. We are using simple inverted index with document frequency incorporated at the start of the list for each term.

After index construction we have 70,807 distinct terms and 11,544 documents.

**Similarity Measurement**

**SIMILARITY MEASUREMENT BLOCK DIAGRAM**

Process query

Calculate similarity with documents

Sort in decreasing order of similarity

Take top K results

Display to the user

User selects most relevant result as query entered may be incomplete

**Note:** For the retrieval of the appropriate book based on the user query, the query is first processed in the same manner as the documents had been before the index creation. This involves splitting the sentences into its constituent words and removing all punctuation. Once the query has been processed, the similarity between the query and each document is measured using the method proposed by Croft and Harper (1979).  According to this method Formula of similarity Varying C  
The documents are arranged in the order of decreasing similarity with the query and the top k results are returned. The user selects the result that is most relevant. This is necessary because in most cases the query that is entered is incomplete, i.e, it does not have the entire title or author name.

We have performed local term expansion based on stemming.  
Following are the steps in our methodology.  
1. *Identify initial set of tokens --* Our system first identified the individual terms  
occurring in the document collection.  
2. *Pre-processing –* Stop word removal and stemming is performed. We used  
porter-stemming algorithm [20].  
3. *Document weighting*. We assigned weights to the terms in each document by the df.idf scheme by Croft and Harper (1983).  
5. *Document ranking with unexpanded query*: We computed a document ranking by computing attribute-based similarity with tuned weights that go toward calculating the final similarity score between the document vectors and the unexpanded query vector.

**Similarity = wName\*sim(Name) + wAuthor\*sim(Author) + wRating\*sim(Rating) + wCategory\*sim(Category) + wPlot Summary\*sim(Plot Summary)**

**where sim(x) is the similarity of the book with the test document for attribute x**

Name, Author, Rating, Category and Plot Summary were the attributes of each book considered for measuring similarity between the document and the query as above, with the following heuristics:

Name: wName was taken to be .2

Author: Higher weighting if author is same i.e. wauthor = .6; else 0

Rating: wRating was taken to be.5

Category:wcategory was set at .9

Plot Summary: wPlot Summary = .7 was **found to give the best results**

Order of weights taken for computing similarity:

**wCategory>wPlot Summary­>wRating>wAuthor>wName**

**Category** gives broad concepts and genres, and have most information about the book. Books with similar categories therefore should carry more weight, for they deal with same themes.

**Plot Summary** gives a concise description of the book, and its content, and books with similar plot details should be given more weightage over books having similar ratings, or same author.

**Ratings** show how good the book was, and therefore, books with higher rating should score higher in recommendation.

**Authors** tend to write books in similar tastes, with similar themes and topics. But authors sharing first, or last names need not write similar books. Therefore, weight is given only when author is same; else, it is not considered for similarity score. eg: Jane Austen and Jane Costello share the same name, but write very different books, and suggesting books of one for a book of the other based on same first name would be bad heuristics.

**Name** of the book may carry weightage in computing similarity, when they have same words, suggesting that they probably deal with same thing, but this might not always be true. Therefore, we take weighting for it to be minimum as compared with weights of other attributes, but not zero.

**GUI Description**

To make the book recommender more user-friendly, a GUI was made for the application. It takes a query from the user which includes the name of the book he/she wants to find, as well as the author name(s), if they are know. On clicking "ENTER" The top 5 search results are displayed to the user, who is then supposed to enter the index number of the book he/she was looking for in another text box. On submitting this index number, the application recommends 10 books to the user which it has judged to be similar to the book the user had queried for, using certain heuristics. The scores of all the recommended books are also displayed to help the user decide which book is most suitable for his/her need.

**5. Evaluation Strategy**

The following evaluation takes place over 24 queries which gives the top 10 results for each query put in the text box of the GUI.

**QUERY SIMILARITY**

For the search query, the most important measure is precision@1 because the user wants the informational need to be addressed by the query being present at the first place.

**RECOMMENDER SIMILARITY**

For recommender system, we are using three evaluation measures i.e. precision@4 , precision@10 and MAP(Imran et al.,2010). We are using these measures because the most important metric for a recommender system is precision(because we want all the recommended books to be relevant). As we also have to capture the order of the books as the results are ranked, thus we are using these three measures.

**6. Experimental Results and evaluation: Present your results and evaluation of your system**

**The evaluation procedure includes searching for a query in the recommender system and then the members marked the results of the recommender as relevant or non-relevant(by thoroughly seeing the plot, author-name, ratings and the theme). Based on these values by the team members, various evaluation measures were calculated.**

**a. Different queries and their retrieved results and ranking.**

**We ran a python script to automatically store the result of the recommender searches in a file as soon as the search was completed. All such files have been attached with the report.**

**SIMILARITY IN RANKING SEARCH QUERIES**

The precision@1 measure was coming out to be 1. All the files related to search queries have been attached with the report.

**SIMILARITY IN RANKING RECOMMENDER SIMILARITY**

There were 24 queries given out of which 2 have been displayed below. All the others have been attached.

The following table displays the index, book-name , author-name and the similarity score calculated.

Number the Stars by Lois Lowry

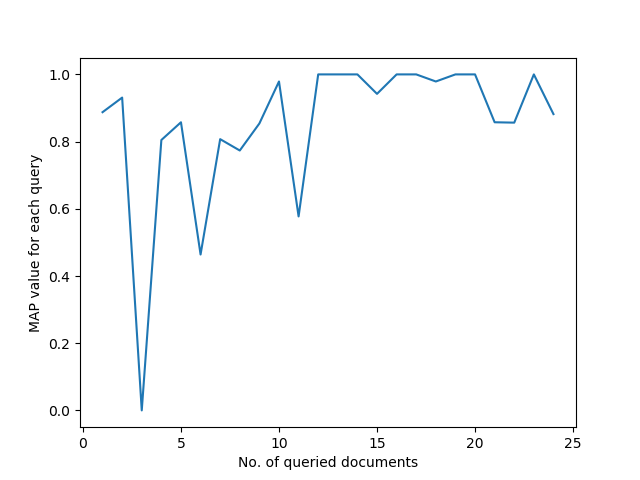
|  |  |  |  |
| --- | --- | --- | --- |
| Index | BookName | AuthorName | Similarity Measure |
| 1 | The Upstairs Room | Johanna Reiss | 16.5340409493 |
| 2 | Once | Morris Gleitzman | 16.4197247937 |
| 3 | I Am David | Anne Holm | 16.4197247937 |
| 4 | When Hitler Stole Pink Rabbit | Judith Kerr | 16.4197247937 |
| 5 | The Painted Bird | Jerzy KosiAski | 15.3182046106 |
| 6 | The Diary of a Young Girl | Anne Frank | 15.3182046106 |
| 7 | Address Unknown | Kathrine Kressmann | 15.203888455 |
| 8 | The Complete Maus | Art Spiegelman | 14.6527746168 |
| 9 | The Silent Boy | Lois Lowry | 14.6150835448 |
| 10 | Enemies A Love Story | Isaac Bashevis Singer | 14.5384584612 |

Black Beauty by Anna Sewell

|  |  |  |  |
| --- | --- | --- | --- |
| Index | BookName | AuthorName | Similarity Measure |
| 1 | The Black Stallion | Walter Farley | 11.5387551279 |
| 2 | My Friend Flicka | Mary OHara | 10.8838785286 |
| 3 | Three Beloved Classics by E B White Charlottes Web the Trumpet of the Swan Stuart Little | EB White | 10.8838785286 |
| 4 | The Real Mother Goose | Blanche Fisher Wright | 10.8838785286 |
| 5 | King of the Wind The Story of the Godolphin Arabian | Marguerite Henry | 10.8838785286 |
| 6 | Misty of Chincoteague | Marguerite Henry | 10.8838785286 |
| 7 | Shiloh | Phyllis Reynolds Naylor | 10.8838785286 |
| 8 | Summer of the Monkeys | Wilson Rawls | 10.8838785286 |
| 9 | Where the Red Fern Grows | Wilson Rawls | 10.8838785286 |
| 10 | There Was an Old Lady Who Swallowed a Fly Classic Books | Pam Adams | 10.0998212236 |

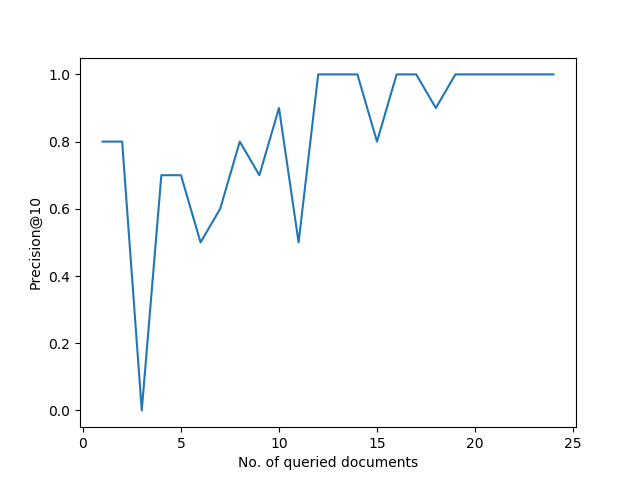
**b. Goodness of the retrieved results along with ground truth.**

**In order to find these values and compute the graphs, we have written separate python script for each of them. The python scripts are attached with the Report. By evaluating the results and using measures like MAP(Mean Average Precision), Precision@10 and Precision@4, we get the following values.**



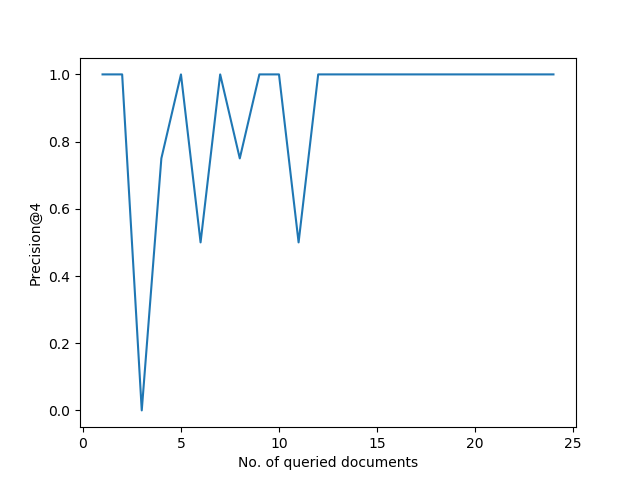
**Average MAP = 0.852185846561**

**It means that for on an average the first 85.2% of the results will be relevant by the recommender.**



**Average Precision@10 = 0.820833333333**

**It means that out of the first 10 results presented to the user, 82.08% of the results will be relevant to the user on an average.**



**Average Precision@4 = 0.895833333333**

**It means that out of the first 4 results presented to the user, 89.5% of the results will be relevant to the user on an average.**

**The ground truths have been attached with the file. However, here we have displayed tables for 2 of the 24 values.**

Number the Stars by Lois Lowry

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index | BookName | AuthorName | Similarity Measure | Relevance(1/0) |
| 1 | The Upstairs Room | Johanna Reiss | 16.5340409493 | 1 |
| 2 | Once | Morris Gleitzman | 16.4197247937 | 1 |
| 3 | I Am David | Anne Holm | 16.4197247937 | 1 |
| 4 | When Hitler Stole Pink Rabbit | Judith Kerr | 16.4197247937 | 1 |
| 5 | The Painted Bird | Jerzy KosiAski | 15.3182046106 | 0 |
| 6 | The Diary of a Young Girl | Anne Frank | 15.3182046106 | 1 |
| 7 | Address Unknown | Kathrine Kressmann | 15.203888455 | 1 |
| 8 | The Complete Maus | Art Spiegelman | 14.6527746168 | 0 |
| 9 | The Silent Boy | Lois Lowry | 14.6150835448 | 1 |
| 10 | Enemies A Love Story | Isaac Bashevis Singer | 14.5384584612 | 1 |

Black Beauty by Anna Sewell

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index | BookName | AuthorName | Similarity Measure | Relevance(0/1) |
| 1 | The Black Stallion | Walter Farley | 11.5387551279 | 1 |
| 2 | My Friend Flicka | Mary OHara | 10.8838785286 | 1 |
| 3 | Three Beloved Classics by E B White Charlottes Web the Trumpet of the Swan Stuart Little | EB White | 10.8838785286 | 1 |
| 4 | The Real Mother Goose | Blanche Fisher Wright | 10.8838785286 | 1 |
| 5 | King of the Wind The Story of the Godolphin Arabian | Marguerite Henry | 10.8838785286 | 1 |
| 6 | Misty of Chincoteague | Marguerite Henry | 10.8838785286 | 1 |
| 7 | Shiloh | Phyllis Reynolds Naylor | 10.8838785286 | 0 |
| 8 | Summer of the Monkeys | Wilson Rawls | 10.8838785286 | 1 |
| 9 | Where the Red Fern Grows | Wilson Rawls | 10.8838785286 | 1 |
| 10 | There Was an Old Lady Who Swallowed a Fly Classic Books | Pam Adams | 10.0998212236 | 0 |

**7. Conclusion and future work**

In the beginning of the report, we said that our Recommender System has the job of recommending new books that are similar to the book that was queried by the user by using a ranking measure for retrieved results from the corpus. The values generated by evaluation shows satisfactory results. However, there are some limitations to our model like limited corpus and static model. Due to this, it is possible that the most relevant documents are not shown in comparison to most book recommender. However, there is always a scope for improvement.

There can be many improvements to the book recommender system to make it more user-friendly.  As the recommender suggests books by analysing the category, plot summary, rating, author and title of the book, changes can be incorporated so that the users are able to select which parameters are more important to them, and by how much (currently this is done by the programmer itself). Thus, if the user wants to find more books by the same author, but is not very particular that it should only be of the same genre and should have a similar plot line, he/she can give 'author' more weightage and give the remaining parameters weightages close to zero.

Another possibility could be to adapt this system so that it can be used for Research Paper recommendation by using a suitable database. Literature review is one of the most important parts of research, and with a research paper recommender in place, this task could be simplified considerably. The heuristics would be similar. Instead of plot summary the abstract could be used, instead of category/genre the keywords of the paper could be used, instead of ratings citations can be used, etc.

**References**

1. Information Retrieval: Data structures and Algorithms edited by William B. Frakes and Ricardo Baeza-Yates
2. CROFT, W. B., and D. J. HARPER. 1979. "Using Probabilistic Models of Document Retrieval Without Relevance Information." Documentation, 35(4), 285-95.
3. Manning C.D., Raghavan P., & Schütze H. Introduction to Information Retrieval, Cambridge University Press, Cambridge (UK), 2008.
4. Yih W., “Learning Term-weighting Functions for Similarity Measures”, Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics ACL, 2009.
5. Metzler D., Dumais S., & Meek C., “Similarity Measures for Short Segments of Text”, ECIR'07 Proceedings of the 29th European Conference on IR Research, Springer-Verlag Berlin, Heidelberg, 2007, p. 16-27.
6. HARPER, D. J. 1980. *Relevance Feedback in Document Retrieval Systems: An Evaluation of Probabilistic Strategies.* Doctoral dissertation, Jesus College, Cambridge, England.
7. Imran Hazra ,Sharan Aditi,*" Selecting effective expansion terms for better Information Retrieval ".* International Journal of Computer Science and Applications, Technomathematics Research Foundation© Vol. 7, No. 2, pp 52 - 64, 2010