Book Recommendation System

Group 2

December 5, 2016

1 Introduction

The Blockbuster book recommender system is based on the need for generating relevant book recommendations to users based on their need and requirements of the user. The system needs to recognise the need of the user at a particular instant in time as his preferences and requirements may keep changing. The books which are recommended must assist the work (or essays) written by the users in the future and expose them to these ideas before they feel the need to brainstorm and use a search engine thus saving time for them.

The report first describes the System Design of the Blockbuster system before going into the specifications of the prototype and stating the limitations of the design and possible improvements.

2 System Design

The figure displays the black box diagram for the system design of Blockbuster. Given an input essay from the user the system first stores it into the database along with any previous work done by the user. The essay is then passed onto the Natural Language Processing(NLP) block to extract the keywords before using the Google API to fetch books. The recommender then uses all the users and essays stored in the database to provide recommendations to the user.
The following sections describe each function in details, including the motivation for design choices.

2.1 Extracting Keywords

There are mainly three kinds of keyword extraction algorithms

1. Extraction using word frequency
2. Extraction using lexical analysis
3. Extraction using Bayes classifier

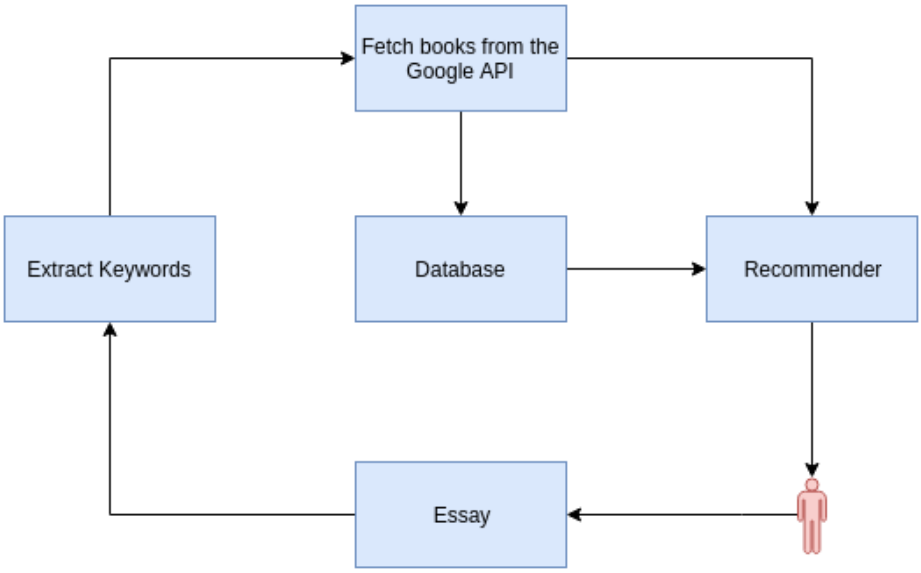

Figure 1:

Our goal is to use a method of keyword extraction that operates on individual documents and is not corpus dependant or context dependent. The method should extract the same keywords each time regardless the current status of the corpus. For this purpose RAKE seems like the best option mainly because it is language, domain and grammar independent, it can handle dynamic collections very well and it can adapt to style and content of the text.

It works in the following manner[1]

1. Get a list of stopwords, phrase delimiters and word delimiters.
2. Partition the document into candidate keywords where the stop words and delimiters act as a partition of the document (forms a sequence of content words)
3. Evaluate each candidate based on 3 metrics(word frequency, word degree and their ration) and receive the final keywords.
4. Add the content words containing interior stop words from previous list
5. Select one third of all the content words having the highest score as keyword.

[1]Stuart Rose, Dave Engel, Nick Cramer and Wendy Cowley. Automatic keyword extraction from individual document

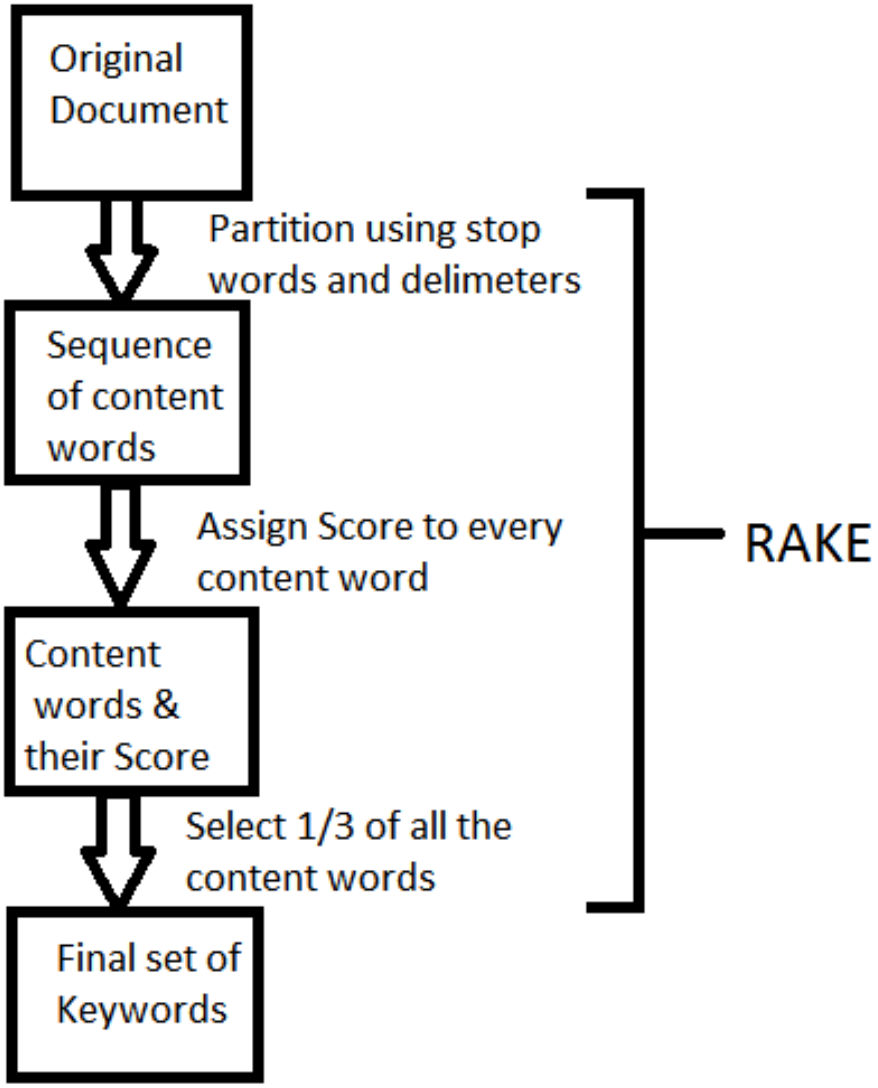

Figure 2:

Why use RAKE in book recommender?

1. Our user will add essays to their accounts regularly. Dynamic nature of RAKE is quite useful here.

2. Writing style of every writer is different and RAKE works independent to

that.

2.1.1 Ideal Model

It takes care of every NLP tasks. It provides the keyword based on

1. What the writer is clearly saying
2. What the writer is trying to imply
3. What writer is feeling
4. How much importance is writer giving to his/her current edit
5. What is writers main idea or theme
6. What the writer might need in future based on his/her current work

2.2 Fetching books from Google API

Google provides a limited Google Books API. It limits the total number of request to Google servers in order to push people to use the embedded viewer. For demonstration purposes, we make use of this limited API but for production level system, we need to employ caching.

2.2.1 Caching

Due to the limitations on the number of queries made to Google Books API, we need to reduce the queries by using various caching mechanisms. The most suitable way, here, is to maintain a list (hashed) of all the keyword sequences along with its reply of list of books. This will also decrease the response time as the system scales and the website becomes famous.

The production level system will use Memcached a FOSS high performance, distributed memory object caching system. Its an in-memory key-value store for small chunks of arbitrary data, and is supported by the Django framework which has been used in the development of the prototype.

2.3 Collaborative Recommender

We first discuss our motivation for choosing the collaborative recommender and then talk about the way it is implemented in the prototype. The subsequent section on the Limitations discusses possible shortcoming and the means to overcome them.

2.3.1 Motivation

The type of recommendation algorithm to choose depends on the user perspective and the type of experience we as product developers want to provide to our users. As a business the USP of Blockbuster is to create a hub of writers, and the only way to attract writers is to provide them with a platform that enables them to get over their writers block. A way where they can extend their given essays to the next idea instead of getting more information related to the idea they have been working on till now. As a system designer, given the set of all users and their essays, we need to recommend a book to the active user based on the following characteristics:

1. Topic of the active essay
2. The essay or work that is most relevant to the user and he wants to pursue immediately
3. The books must reflect the topics that are related to the topic of the essay and are not about the essay

In the course of implementing the product the possible recommender algorithms that can be considered are:

1. Content Based Filtering
2. Collaborative Filtering
3. Hybrid Filtering

In this regard a collaborative filtering based recommender is the natural choice for the product as it comes closest to recommending topics which are possibly the logical next step that the user may want to pursue. Given a particular essay the system will first find how relevant the topic is to the user in his current work. It will then look at other users who are working on the same things and recommend material that they use. Within collaborative filtering we have two variants that we can consider:-

1. User-User Collaborative Filtering
2. Item-Item Collaborative Filtering

The prototype of the product uses the user-user collaborative filtering. Given a particular user we first find all the similar users to him/her and then recommend items that these users rate highly. Thus if the number of users is U, searching for similar users is an $\mathcal{O}(U)$ operation. In addition to this the groupings may change each time a user changes or adds a new rating. This clustering needs to be performed each time the active user has to be provided with recommendations. Succeeding sections of the report discuss alternatives such as item-item collaborative filtering and hybrid filtering.

2.3.2 Implementation

A fundamental assumption behind the user-user collaborative filtering method is that other users opinions can be selected and aggregated in such a way as to provide a reasonable prediction of the active users preference.

Given a series of essays by a user and the corresponding books that are textually similar to these essays, we can estimate the rating that each user gives to every book that is retrieved by the system. Each time a book is retrieved because of its textual similarity with the input essay we increment the rating of the book by a constant amount to simulate its relevance to the user's current work. The user and the book ratings data can be then treated as a matrix with the rows being the users and the columns representing each book with the entries corresponding to individual estimated ratings of each book by a given user.

The users are then clustered into groups using the K-Means clustering algorithm and recommendations to the active user are provided based on his membership to a group. The number of clusters were approximately one-tenth of the number of users, as intuitively the number of clusters will increase linearly with the number of users. In the clustering process there are numerous measures of similarity that we can use:

1. Pearson correlation
2. Constrained Pearson correlation
3. Spearman Rank correlation
4. Cosine similarity

The prototype of Blockbuster uses the cosine similarity approach. However experiments show that Pearson correlation tends to perform the best for movie based datasets which may be extended to books[2].

In constructing the ideal product a hybrid approach using a pearson correlation based clustering algorithm, and an item-item collaborative filtering would work best as it would address the issue of cold start and scalability (as discussed in the section on Limitations).

3 Product

This section describes the functionality of the prototype and its working. The I/O specifications of the application from a user's perspective are:

1. Input - Idea that user is working on in the form of an essay.

[2]Michael D. Ekstrand, John T. Reidel, Joseph A. Konstan. Collaborative FIltering Recommender Systems

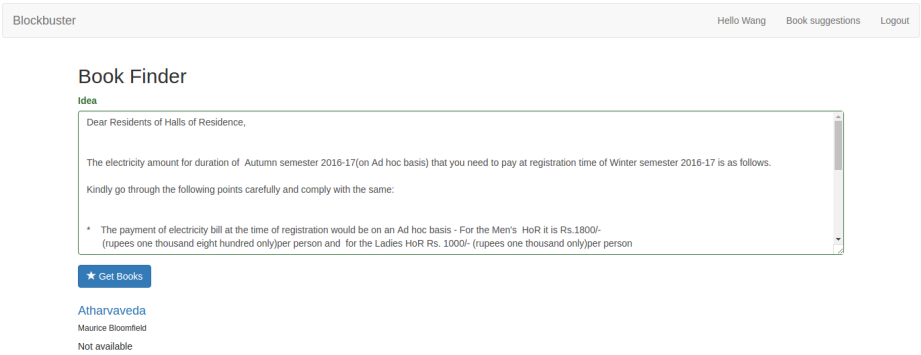

Figure 3:

2. Output - Book recommendations to the user of the following two types:
   (a) Books similar to the idea of the active user as displayed in the image above.
   (b) Books that will help the user extend his idea.

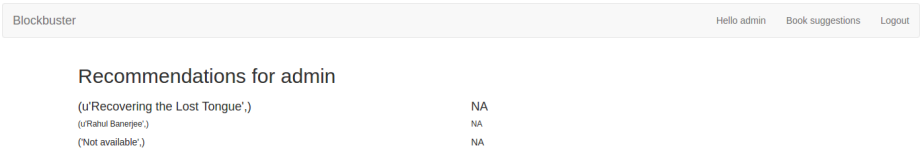

Figure 4:

In order to implement the system features described in section 2, the following tools were used:

1. Gensim API for Extracting Keywords
2. Google books API for fetching the books and MySQL for the database
3. K-Means clustering algorithm using Scikit for implementing user-user collaborative filtering.

The prototype has been programmed in Python using the Django framework and is currently hosted on Heroku.

4 Limitations

1. Computationally Expensive Recommendation Filtering - Useruser collaborative filtering, while effective, suffers from scalability problems as the user base grows. Searching for the neighbors of a user is an O(U) operation as each time a user rates a new book the clusters have to be reformed. This is particularly expensive when the number of users are extremely large. A solution to this is to use item-item collaborative filtering. In this if two items tend to have the same users like and dislike them, then they are similar and users are expected to have similar preferences for similar items. A list is maintained of the top similar books for any given books, and when a user rates a given book then the ratings for only those top similar books are estimated and recommended. In systems with a sufficiently high user to item ratio, however, one user adding or changing ratings is unlikely to significantly change the similarity between two items, particularly when the items have many ratings. Therefore, it is reasonable to pre-compute similarities between items in an itemitem similarity matrix and cache the k most similar items to each item.
2. Cold Start - A collaborative filtering recommendation is effective only when we have existing user data. To recover from this we can use a hybrid filtering approach.

5 Future Scope

This model could be improved by adding following features:

1. Predict on certain level what the writer is mostly going to need (in future) on the basis of how he/she is proceeding and recommend books according to that.
2. Make the recommendation of book real time thus the list of book recommended to the writer would change as he/she writes instead of current model where the recommender will pull the essay AFTER the writer is finished and then recommend books to him/her.