A SMART MEDIA-BASED RECOMMENDATION SYSTEM

A Thesis Presented to the Department of

Computer Science,

African University of Science and Technology, Abuja

In Partial Fulfillment of the Requirements

For The Degree of

MASTER OF SCIENCE

By

Ogbonna Amanze Darlington

Abuja, Nigeria.

December, 2014.
A SMART MEDIA-BASED RECOMMENDATION SYSTEM

By
Ogbonna Amanze Darlington

A THESIS APPROVED BY THE DEPARTMENT OF
COMPUTER SCIENCE

RECOMMENDED: ..............................................................
Supervisor, Professor Mohamed Hamada

..............................................................
Head, Department of Computer Science

APPROVED: ..............................................................
Chief Academic Officer

..............................................................
Date
Abstract

Smart media devices such as: smartphones and tablets are getting more powerful, smarter, cheaper and hence more popular.

Recommendation systems become very common in e-business and e-Commerce, for example: Amazon, Google, eBay, Facebook, etc. all are using recommendation systems to promote their business. Recommendation systems are rarely used in learning; however it can be very useful.

The proposed project works as follow:

- Send a learning query to sites, sources and repositories across the Web and gather relevant information through the use of recommendation system that filter all the useless or irrelevant materials off the main list of recommended items.
- Filter result from other user's preference using collaborative-filtering, having the current query in mind.
- Use TFIDF for content-based filtering and ranking of the shortlisted pages or articles.
- Present the shortlist based on their rank to the user of the system.
Dedication

This thesis is dedicated to the glory of the Lord God Almighty; Heaven and Earth are full of His grace; and to every member of my family, most especially my parents, and my big-sisters; for their love and support. I also honor every professor that has molded me to be a better Computer Scientist, and made me to be more confident in my field.

To you all I say, God bless you.
Acknowledgement

I acknowledge the Supreme Being, the Almighty God, who gave me the strength to carry-on.

I want to appreciate the fact that my supervisor Prof. Hamada can find time from his busy schedule to guide me in the right path. He helped me to focus on the right direction, when I was busy doing the wrong thing.

I appreciate the Head of Department Computer Science, for his continuous monitoring of our progress. He made sure we were doing enough, in order to prep us before our thesis presentation.

I also appreciate my fellow class-mates that were there to answer my silly but important questions (Everyone in MSc. Computer Science 2013/2014).
# Table of Contents

Abstract .......................................................................................................................... iii

Dedication......................................................................................................................... iv

Acknowledgement ........................................................................................................... v

Table of Figures ................................................................................................................ 4

1. Introduction .................................................................................................................. 6
   1.1. Context ..................................................................................................................... 6
   1.3. Research Objective ............................................................................................... 7
   1.4. Research Methodology .......................................................................................... 8
       1.4.1. Information gathering ...................................................................................... 8
       1.4.2. Collaborative Filtering .................................................................................... 8
       1.4.3. Content-based filtering .................................................................................. 9
       1.4.4. Hybridization of Both filtering methods ......................................................... 9
       1.4.5. Connecting of the Android application to the Server through HTTP protocol .... 9
   1.5. Organization of document ...................................................................................... 10

2. State of Art .................................................................................................................... 11
   2.1. Collaborative filtering .......................................................................................... 11
       2.1.1. Overview of the Collaborative Filtering Process .............................................. 11
       2.1.2. Types of Collaborative Filtering Algorithms .................................................. 13
       2.1.3. Application ...................................................................................................... 14
   2.2. Content based filtering ......................................................................................... 15
       2.2.1. TFIDF ............................................................................................................. 15
       2.2.2. Application of TF-IDF .................................................................................. 18
   2.3. Java ....................................................................................................................... 18
   2.4. Android ............................................................................................................... 18
2.5. Servlet / JSP ........................................................................................................... 19
2.6. JSON ....................................................................................................................... 20
2.7. Related work ........................................................................................................... 22
  2.7.2. Combining Content-Based and Collaborative Filters in an Online Newspaper [15] 30
  2.7.3. Item-Based Collaborative Filtering Recommendation Algorithms [1] ............... 31
2.8. Discussion (outcome, pros, cons) & why your work ............................................. 32
  2.8.1. Pros and Cons of Different techniques ............................................................ 32
2.9. Proposal .................................................................................................................. 35
3. Application / Case Study / Benchmarking ................................................................. 39
  3.1. Case Study ............................................................................................................ 39
  3.2. ER Diagram .......................................................................................................... 40
  3.3. Building Dataset .................................................................................................. 40
  3.4. Query the System through Android and HTTP .................................................... 41
    3.4.1. Querying the server via emulator ................................................................. 41
    3.4.2. Querying the server via a physical device .................................................... 41
  3.5. Gathering Information .......................................................................................... 42
    3.5.1. Getting URL links .......................................................................................... 42
    3.5.2. Removing irrelevant links .............................................................................. 42
    3.5.3. Avoiding Duplicates ...................................................................................... 44
    3.5.4. Making content temporarily available .......................................................... 44
  3.6. Collaborative filtering of URL-List ..................................................................... 45
  3.7. Content-based filtering of final List ..................................................................... 47
    3.7.1. Getting documents weight ............................................................................. 48
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Structure of the recommendation process</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Pearson Correlation Coefficient</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>Formula for prediction score</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>Isolation of co-rated items and similarity computation [1]</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>TDIDF example of 3 terms and 15 documents [2]</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>The servlet application architecture employing the HTTP server [3]</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>JSON Object [7]</td>
<td>21</td>
</tr>
<tr>
<td>9</td>
<td>JSON Array [7]</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>Average rank of correct answer, basic algorithms. Non-significant differences between results are indicated with brackets [19]</td>
<td>22</td>
</tr>
<tr>
<td>11</td>
<td>Training phase of a Weighted Hybrid [19]</td>
<td>23</td>
</tr>
<tr>
<td>12</td>
<td>Candidate generation in a weighted hybrid [19]</td>
<td>23</td>
</tr>
<tr>
<td>13</td>
<td>Scoring in weighted hybrid [19]</td>
<td>23</td>
</tr>
<tr>
<td>14</td>
<td>Switching decision in switching hybrid [19]</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td>Candidate generation in switching hybrid [19]</td>
<td>24</td>
</tr>
<tr>
<td>16</td>
<td>Scoring in a Switching hybrid [19]</td>
<td>24</td>
</tr>
<tr>
<td>17</td>
<td>Candidate generation in a mixed hybrid [19]</td>
<td>25</td>
</tr>
<tr>
<td>18</td>
<td>Scoring in a mixed hybrid [19]</td>
<td>25</td>
</tr>
<tr>
<td>19</td>
<td>Training phase of a feature combination hybrid [19]</td>
<td>26</td>
</tr>
<tr>
<td>20</td>
<td>Candidate generation in a feature combination hybrid [19]</td>
<td>26</td>
</tr>
<tr>
<td>21</td>
<td>Scoring in a feature combination hybrid [19]</td>
<td>26</td>
</tr>
<tr>
<td>22</td>
<td>Training phase in a feature augmentation hybrid [19]</td>
<td>27</td>
</tr>
<tr>
<td>23</td>
<td>Candidate generation in a feature augmentation hybrid [19]</td>
<td>27</td>
</tr>
<tr>
<td>24</td>
<td>Scoring in a feature augmentation hybrid [19]</td>
<td>27</td>
</tr>
<tr>
<td>25</td>
<td>Training phase in a cascade hybrid [19]</td>
<td>28</td>
</tr>
<tr>
<td>26</td>
<td>Candidate generation in a cascade hybrid [19]</td>
<td>28</td>
</tr>
<tr>
<td>27</td>
<td>Scoring in a cascade hybrid [19]</td>
<td>28</td>
</tr>
<tr>
<td>28</td>
<td>Training phase in a meta-level hybrid [19]</td>
<td>29</td>
</tr>
<tr>
<td>29</td>
<td>Candidate generation in meta-level hybrid [19]</td>
<td>29</td>
</tr>
</tbody>
</table>
Figure 30 Scoring in a meta-level hybrid [19].................................................................29
Figure 31 Application of various recommender system Hybrids [19]..............................30
Figure 32 Gathering data from search engines ................................................................36
Figure 33 Storing webpage data locally ........................................................................36
Figure 34 Checking local pages to see if it has been rated .............................................37
Figure 35 Item-based collaborative filtering using an already rated item ......................37
Figure 36 Adding recommended URLs to local data ....................................................38
Figure 37 ER diagram of the whole system ..................................................................40
Figure 38 EditText and button of Android application ..................................................41
Figure 39 Related search links in a Bing search ............................................................43
Figure 40 Combining the result from various sources ..................................................44
Figure 41 Table structure of database table 'temp' .......................................................45
Figure 42 Dataset from the MovieLens database .........................................................46
Figure 43 Android application populated with URL recommendations from server ......51
Figure 44 Landscape view of populated ListView and WebView ..................................52
Figure 45 ER diagram to show the Item-Tag-Term relationship ...................................53
Figure 46 Rating in Android Application and in Server back-end ..............................55
Figure 47 Register Activity in Popup form ....................................................................56
Figure 48 When a user picks a username that exists on the database .........................56
Figure 49 When the password in both Edit-text fields don’t match ..............................57
Figure 50 User tester successfully added to database ..................................................57
Figure 51 Login Activity in form of a Popup.................................................................58
Figure 52 Successful login in the Login Activity ..........................................................58
Figure 53 Android application waiting for server’s response .......................................59
Figure 54 Adding URL to database and viewing it on the Web-View, by clicking a List-Item...59
Figure 55 Rating an item (URL) using a rating bar ......................................................60
Figure 56 Class Diagram of Server .............................................................................65
Figure 57 Class Diagram for Server Cont. ....................................................................66
Figure 58 Class Diagram of Android Application .......................................................67
1. Introduction

With the growth of the web, there is an explosion in the size of content available to various users around the world. The materials available are a mixture of useful and useless documents to the interest of the user. This show gives the need for a better way of getting useful document from the web and mostly through the taste of the user of the web.

This project aims at using the taste and the behavior of a user to get documents that are relevant to what the user needs. Depending on the choices of the user, they may have people with the same taste as them, and the system is to harness this opportunity and recommend materials, based on interaction of the neighbor of the users with other interesting materials. Using a collaborative filtering approach, the system can find the items (document) that other users with similar interest with the current user have read and rated to a good degree. In the early stages of the system’s life, the system can’t depend on the Collaborative approach alone, or else the system’s recommendation will be inadequate, thus prompting the need for another filtering approach to support the weaknesses of the collaborative approach. The most use filtering approach that has been used with the collaborative filtering approach is the “Content-based filtering” approach. Adding a content-based filter to the result of the collaborative will yield a more concrete recommendation that will be close to what the user requires.

1.1. Context

Recently Recommendation Systems are becoming common and important in e-technologies such as e-business and e-learning. Many major companies such as Amazon, Google, etc. have its own recommendation system. Recommendation systems can help customers to pick up the most relevant products that fit with their needs. In the area of e-learning and learning in general, there is no much research have been done to design and build a reliable recommendation system that can help learners to pick up the most relevant materials that can speed up and enhance their learning process.

In this project we plan to design and implement a learning recommendation system, which supports smart media such as smartphone and tablets, as its clients.

To achieve this, the system needs to be hosted probably on a remote server with the capability to run codes efficiently, and optimally. A server side will be written in Java Servlets in order to be
able to use various libraries available. All communication will be across a network and the source data will be from the internet.

1.2. Problem Statement
In the digital world today, the amount of content finding their way to the internet, has become alarming in the rate of increase, and this has led to an increase in the scrutiny of this content by search engines and other pointer-sites. It is now glaring that there is a need to provide a personal touch to the recommendation of materials to users.

If a form of recommendation can be added to the search result of user’s, then the probability of them getting more interesting materials will become high. Take for example, a school with lots of materials in their repository, which offers the students or the public access to their wealth of content, but the only provide a generic search. If there is a way of finding the articles or papers of interest, and at the same time finding materials with similar relevant content, based on the content similarities and the people who previously viewed and rated the materials, thus saving the next person the trouble of viewing irrelevant materials.

1.3. Research Objective
In this research, the aim is to aid common search with the help of filtering techniques, in order to improve the quality of materials recommended to people or users of the system.

Due to the fact that finding material for research and study may be hard to get on the web, a user might search with a search engine for materials and information, and the user may or may not get the result he/she seek from the first search engine, and he/she may try the second, third, fourth etc. this process of searching numerous places for information may be time consuming and tedious. Then a system that searches more than one place will surely be better.

We aim to provide a multi-platform search that is based on hybrid filtering, in order to provide the user with a result that is close to what the user wants or needs.

The normal search (e.g. Google) also does its filtering, but we aim to join the result from different sites and repositories, giving users a list that is more scrutinized than the result presented by a singular website or service.
With the help of a user’s preference and the help of the characteristic of users with the same preference as the current user, our system will be able to provide a better result-set for the user of the system.

Let us consider a university of library reach in materials, and there is a way of searching for materials, the system will be able to provide good search based on the title of the material or even the some part of the content might be taking into consideration. This project not only emphasizes on the approach that will be adopted in the implementation, but it depends on the creativity of future developers to enhance the application, and use it for more advanced reasons. We aim to show how relevant the content-based and collaborative filtering algorithms are in the recommendation of materials based on the current query of a user.

1.4. Research Methodology

The approaches to be adopted in this project are:

1.4.1. Information gathering

This is where we source the internet for information, primarily using search engines to gather information, and using a tool to search the DOM for links (The Document Object Model (DOM) is an application programming interface (API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated [13]). When the list from different sources have been obtained, then they will be combined to form a single list without duplicates. This list will be sent to the collaborative filter before it’s sent to the content-based filter.

The list mentioned above is a database table with fields of ID, TITLE, URL, BODY, and SCORE. When the list gathered, then the system using JSoup, the system will get for all the URLs, the title of the page, the content of the pages (i.e. the texts inside the body of the page), and saves the URL also in the URL field. This table is used exclusively by the current user in order to avoid inconsistency in the recommendation.

1.4.2. Collaborative Filtering

Sometimes called social filtering, it is a filtering technique that uses the relationship between peer users with similar rating history, to generate recommendations using the neighborhood [19].
With this filtering process, the system makes recommendations based on whether the URLs in the list fetched from the internet have been rated before, for every URL in the list that has been rated a recommendation (finding URLs/items similar to the current one) for extra URLs are made by the system, and they are added to the list of URLs.

1.4.3. Content-based filtering

The system generates recommendations from two sources: product features, and user-ratings. Content-based recommenders treat recommendation as a user-specific classification problem and learn a classifier for the user's likes and dislikes based on product features [19].

With this process, I intend to rank documents in relation to their weight for a group of terms in a query. For each term, a document has a weight, and depending on the number of terms in a query, a documents has a combined weight. This combine-weight will be used to rank the documents in the temporary-corpus.

1.4.4. Hybridization of Both filtering methods

The combination of two different filtering approaches together to create a new system that is better than any individual filter. Using a hybrid will help to overcome the flaws of a single filter-system.

In this case, after the collaborative filter has finished recommending and adding its recommendation to the list, the list is sent to the content-based filter to undergo filtering again, thus creating a rank that brings the important documents to the top, and leaving the less-relevant ones at the bottom.

1.4.5. Connecting of the Android application to the Server through HTTP protocol

The project client is primarily an android program that queries the system to get some results proposed by the system. So the mobile app needs to communicate with the server in some way, and Android can communicate with remote web servers via a network with the Hypertext Transfer Protocol (HTTP).

To leverage the power of Java, I decided to use the Java-Servlet to run the request sent by the Android application. Different packages have been created to accommodate the various techniques to be used by the system, even the use of Java external libraries like Apache Mahout [4], JSoup, JDBC-MySQL connector etc.
1.5. Organization of document

The organization of this document is as follows: Chapter 2, Shows a detailed review of the techniques and theory used in this thesis. Chapter 3, Shows the processes of the implemented software for our case-study. Chapter 4, is the conclusion that shows the work done in the cause of the project, and the limitation of the application and proposed methods in the use-case.
2. State of Art

2.1. Collaborative filtering

This is filtering system that has been used by many recommenders systems. In general, collaborative filtering is the process of filtering for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc. Applications of collaborative filtering typically involve very large data sets. Collaborative filtering methods have been applied to many different kinds of data including: sensing and monitoring data, such as in mineral exploration, environmental sensing over large areas or multiple sensors; financial data, such as financial service institutions that integrate many financial sources; or in electronic commerce and web applications where the focus is on user data, etc. The remainder of this discussion focuses on collaborative filtering for user data, although some of the methods and approaches may apply to the other major applications as well.

Recommender systems apply data analysis techniques to the problem of helping users find the items they would like to purchase at e.g. and E-Commerce sites by producing a predicted likeliness score or a list of top-N recommended items for a given user. Item recommendation can be made using different methods. Recommendations can be based on demographics of the uses, overall top selling items, or past buying habit of users as a predictor of future items.

Collaborative Filtering is the most successful recommendation technique to date. The basic idea of collaborative filtering algorithms is to provide item-recommendations or predictions based on the opinions of other like-minded users. The opinions of users can be obtained explicitly (Directly) from the user or by using implicit means (Indirectly) [1].

Collaborative filtering is implemented by the use of neighborhood-based algorithm. In neighborhood-based algorithms a subset of users are chosen based on their similarity to active users, and a weighted combination of their ratings is used to produce predictions for the active user [17].

2.1.1. Overview of the Collaborative Filtering Process

The goal of a collaborative filtering algorithm is to suggest new items or to predict the utility of a certain item for a particular user based on the user’s previous liking and the opinions of other like-mined users.
In a typical collaborative filter scenario, there is a list of \( m \) users \( U = \{u_1, u_2, \ldots, u_m\} \) and a list of \( n \) items \( I = \{i_1, i_2, \ldots, i_n\} \).

Each user \( u_i \) has a list of items \( I_{ui} \), which the user has expressed his/her opinions about. Opinions can be explicitly given by the user as a rating score, generally within a certain numerical scale, or can be implicitly derived from purchase records, by analyzing timing logs, by mining web hyperlinks and so on [Badrul S. et al, 2001].

**Note that \( I_{ui} \subseteq I \) and it is possible for \( I_{ui} \) to be null-set.**

There exists a distinguished user \( u_a \in U \) called the active user for whom the task of a collaborative filtering algorithm is to find an item likeliness that can be of two forms.

Prediction is a numerical value, \( P_{a,j} \), expressing the predicted likeliness of item \( i_j \notin I_{ua} \) for the active user \( u_a \). This predicted value is within the same scale (e.g., from 1 to 5) as the opinion values provided by \( u_a \).

Recommendation is a list of \( N \) items, \( I_r \subset I \), that the active user will like the most. Note that the recommended list must be on items not already purchased by the active user, i.e.

\[
I_r \cap I_{ua} = \emptyset.
\]

This interface of collaborative filtering algorithms is also known as Top-N recommendation.

Collaborative filtering can be summarized in the following steps:

I. Weigh the entire user with the consideration to the similarity with the active user.
   The similarities users are measured using the Pearson correlation between their rating vectors.

II. Select \( N \) users with the highest similarity with the active user.
    The users for what is called a ‘Neighborhood’.

III. Calculate a prediction from a weighted combination using the Pearson correlation coefficient, defined below:
\[ P_{a,u} = \frac{\sum_{i=1}^{m} (r_{a,i} - \bar{r}_a) \times (r_{u,i} - \bar{r}_u)}{\sqrt{\sum_{i=1}^{m} (r_{a,i} - \bar{r}_a)^2 \times \sum_{i=1}^{m} (r_{u,i} - \bar{r}_u)^2}} \]

*Figure 2 Pearson Correlation Coefficient*

Where \( r_{a,i} \) is the rating given to item \( i \) by user \( a \);

\( \bar{r}_a \) is the mean rating given by user \( a \); and \( m \) is the total number of items.

In step 3, predictions are computed as the weighted average of deviations from the neighbor’s mean:

\[ p_{a,i} = \bar{r}_a + \frac{\sum_{u=1}^{n} (r_{u,i} - \bar{r}_u) \times P_{a,u}}{\sum_{u=1}^{n} P_{a,u}} \]

*Figure 3 Formula for prediction score*

Where \( p_{a,i} \) is the prediction for the active user \( a \) for item \( i \);

\( P_{a,u} \) is the similarity between users \( a \) and \( u \); and \( n \) is the number of users in the neighborhood.

It is common for the active user to have highly correlated neighbors that are based on very few co-rated (overlapping) item [17].

2.1.2. Types of Collaborative Filtering Algorithms

Memory-based (User based)

Memory-based algorithm utilize the entire user-item database to generate a prediction. These systems employ statistical techniques to find a set of users, known as neighbors that have a history of agreeing with the target user (i.e., they either rate different items similar or they tend to buy similar set of items). Once a neighborhood of users is formed, these systems use different algorithms to combine the preference of neighbors to produce a prediction or top-N
recommendation for the active user. The techniques, also known as nearest-neighbor or user-based collaborative filtering, are more popular and widely used in practice.

Model-based (Item based)

Model-based collaborative filtering algorithms provide item recommendation by first developing a model of user ratings. Algorithms in the category take a probabilistic approach and envision the collaborative filtering process as computing the expected value of a user prediction, given his/her ratings on other items.

![Figure 4 Isolation of co-rated items and similarity computation [1].](image)

2.1.3. Application

Collaborative filtering has been used in the GroupLens project, to filter usenet news [12]. Collaborative filtering in the tapestry mailing and repository system. The Tapestry experimental mail system developed at the Xerox Palo Alto Research Center is predicated on the belief that information filtering can be more effective when humans are involved in the filtering process. Tapestry was designed to support both content-based filtering and collaborative filtering, which entails people collaborating to help each other perform filtering by recording their reactions to documents they read. The reactions are called annotations; they can be accessed by other people’s filters. Tapestry is intended to handle any incoming stream of electronic documents and serves both as a mail filter and repository; its components are the indexer, document store,
annotation store, filterer, little box, remailer, appraiser and reader/browser. Tapestry’s client/server architecture, its various components, and the Tapestry query language are described [6].

2.2. Content based filtering
A content-based filtering system selects items based on the correlation between the content of the items and the user’s preferences as opposed to a collaborative filtering system that chooses items based on the correlation between people with similar preferences [18]. Content-based filtering technique is based on content learnt from the target items. In content-based filtering technique, the web pages are recommended for a user exclusively on a profile built up by analyzing the content of items that the user has rated in the ancient times and/or user’s personal information and preferences [20].

2.2.1. TFIDF
Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), also known as the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

\[ \text{TF}(t) = \frac{\text{Number of times term } t \text{ appears in a document}}{\text{Total number of terms in the document}} = \text{measure of how often a term appears in the document.} \]

- Implication: the more frequent a term occurs in a document, the greater its score
- Rationale: documents which contains more of a term are generally more relevant

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as
"is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

$$\text{IDF}(t) = \log_e\left(\frac{\text{Total number of documents}}{\text{Number of documents with term } t \text{ in it}}\right) = \text{measure of how often the term appears across the index.}$$

- Implication: the greater the occurrence of a term in different documents, the lower its score
- Rationale: common terms are less important than uncommon ones

There are several ways in which terms can be represented in order to be used as a basis for the learning component. A representation method that is often used is the vector space model. In the vector space model a document D is represented as an ‘m’ dimensional vector, where each dimension corresponds to a distinct term and m is the total number of terms used in the collection of documents. The document vector is written as, where $W_i$ is the weight of term $t_i$ that indicates its importance. If document D does not contain term $t_i$ then weight $W_i$ is zero. Term weights can be determined by using the tf-idf scheme. In this approach the terms are assigned a weight that is based on how often a term appears in a particular document and how frequently it occurs in the entire document collection:

$$w_i = tf_i \cdot \log\left(\frac{n}{df_i}\right)$$

where $tf_i$ is the number of occurrences of term $t_i$ in document D, n the total number of documents in the collection and $df_i$ the number of documents in which term $t_i$ appears at least once. The assumptions behind tf-idf are based on two characteristics of text documents. First, the more times a term appears in a document, the more relevant it is to the topic of the document. Second, the more times a term occurs in all documents in the collection, the more poorly it discriminates between documents.
Scoring

We introduce the *overlap score measure*: the score of a document $d$ is the sum, over all query terms, of the number of times each of the query terms ($t \in q$) occurs in $d$. We can refine this idea so that we add up not the number of occurrences of each query term $t$ in $d$, but instead the tf-idf weight of each term in $d$.

$$\text{Score}(q,d) = \sum_{t \in q} \text{tf-idf}_{t,d}.$$
2.2.2. Application of TF-IDF

In 2005 the people at Information Technology Center, Nagoya University wrote a paper titled “Improvement in TF-IDF Scheme for Web Pages Based on the Contents of Their Hyperlinked Neighboring Pages” [11]. In this paper, they proposed a method of ranking webpages based on the TF-IDF score of the pages that links into the site and the also the pages that point out of the site itself.

In June 2008, from the Hong Kong Polytechnic University, The Chinese University of Hong Kong, and Queens College, City University of New York, they wrote a paper showing that TF-IDF term weights can be the outcome of modeling relevance decision-making. It simulates the local relevance decision-making for every location of a document, and combines all of these “local” relevance decisions as the “document-wide” relevance decision for the document. [10]

2.3. Java

Java is a computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. Java is, as of 2014, one of the most popular programming languages in use, particularly for client-server web applications, with a reported 9 million developers. Java was originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them [21].

2.4. Android

Android is a mobile operating system (OS) based on the Linux kernel and currently developed by Google. With a user interface based on direct manipulation, Android is designed primarily for touchscreen mobile devices such as smartphones and tablet computers, with specialized user interfaces for televisions (Android TV), cars (Android Auto), and wrist watches (Android Wear). The OS uses touch inputs that loosely correspond to real-world actions, like swiping, tapping,
pinching, and reverse pinching to manipulate on-screen objects, and a virtual keyboard. Despite being primarily designed for touchscreen input, it also has been used in game consoles, digital cameras, and other electronics.

Figure 6 Architecture of the Android Operating System [9].

2.5. Servlet / JSP

The servlet technology was introduced by Sun Microsystems in 1996. And JSP is an extension of the servlet technology. JavaServer Pages (JSP) are server-side Java EE components that generate responses, typically HTML pages, to HTTP requests from clients. JSPs embed Java code in an HTML page by using the special delimiters <% and %>. A JSP is compiled to a Java servlet, a Java
application in its own right, the first time it is accessed. After that, the generated servlet creates the response [3].

A servlet is a Java class that can be loaded dynamically into and run by a special web server. This servlet-aware web server is called a servlet container, which was also called a servlet engine in its early days.

Servlets interact with clients through a request-response model based on the HTTP.

---

---

2.6. JSON

JSON is a textual data format. It is a subset of JavaScript, which makes it very effective in browser delivered applications. It is language independent, so it is easily supported in virtually all programming languages. The structures of JSON map directly onto the data structures of modern programming languages [5].

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language [7].

JSON is built on two structures:
A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.

An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

An *object* is an unordered set of name/value pairs. An object begins with `{` (left brace) and ends with `}` (right brace). Each name is followed by `:` (colon) and the name/value pairs are separated by `,` (comma).

![JSON Object](image)

*Figure 8 JSON Object [7]*

An *array* is an ordered collection of values. An array begins with `[` (left bracket) and ends with `]` (right bracket). Values are separated by `,` (comma).

![JSON Array](image)

*Figure 9 JSON Array [7]*
2.7. Related work

2.7.1. Hybrid Web Recommender Systems [19]

This paper tries to analyze the hybridization of the common recommendation techniques that have their own strengths and weaknesses, and compare the performance to other hybrids. Four different recommendation techniques were used and various hybrids were implemented.

![Basic algorithms](image.png)

*Figure 10 Average rank of correct answer, basic algorithms. Non-significant differences between results are indicated with brackets [19]*

Different recommender algorithms are combined together, in a particular form. This involves the joining of various recommender algorithms in ways that will yield a better recommendation. The 7 types of Hybrid forms:
• Weighted: The score of different recommendation components are combined numerically.

**Figure 11 Training phase of a Weighted Hybrid [19]**

**Figure 12 Candidate generation in a weighted hybrid [19]**

**Figure 13 Scoring in weighted hybrid [19]**
• Switching: The system chooses among recommendation components and applies the selected one.

![Diagram](image)

*Figure 14 Switching decision in switching hybrid [19]*

![Diagram](image)

*Figure 15 Candidate generation in switching hybrid [19]*

![Diagram](image)

*Figure 16 Scoring in a Switching hybrid [19]*
• Mixed: Recommendations from different recommenders are presented together.

![Diagram of candidate generation in a mixed hybrid]

*Figure 17 Candidate generation in a mixed hybrid [19]*

![Diagram of scoring in a mixed hybrid]

*Figure 18 Scoring in a mixed hybrid [19]*
• **Feature Combination**: Features derived from different knowledge sources are combined together and given to a single recommendation algorithm.

![Figure 19 Training phase of a feature combination hybrid [19]](image)

![Figure 20 Candidate generation in a feature combination hybrid [19]](image)

![Figure 21 Scoring in a feature combination hybrid [19]](image)
• Feature Augmentation: One recommendation technique is used to compute a feature or set of features, which is then part of the input to the next technique.

Figure 22 Training phase in a feature augmentation hybrid [19]

Figure 23 Candidate generation in a feature augmentation hybrid [19]

Figure 24 Scoring in a feature augmentation hybrid [19]
• **Cascade:** Recommenders are given strict priority, with the lower priority ones breaking ties in the scoring of the higher ones.

---

**Figure 25** Training phase in a cascade hybrid [19]

---

**Figure 26** Candidate generation in a cascade hybrid [19]

---

**Figure 27** Scoring in a cascade hybrid [19]
• **Meta-level**: One recommendation technique is applied and produces some sort of model, which is then the input used by the next technique.

The table below shows the implementation of various hybrids in the above hybrid forms, it also shows the ones that are redundant, existing in implementation, and impossible to implement.
Figure 31 Application of various recommender system Hybrids [19]

In the paper, the Content-based and Collaborative (CF/CN) combination can be used in all the form except Meta-level, and it is adopted by various businesses in the commercial world.

Baseline Algorithms: Four basic filtering algorithms were used in the study, as the main algorithms. The algorithms were analyzed with various session sizes.

Collaborative Pearson – CFP, Collaborative Heuristic – CFH, Content-Based – CN, Knowledge-Based – KB.

2.7.2. Combining Content-Based and Collaborative Filters in an Online Newspaper [15]

Due to the explosive growth of mailing lists, this paper aims to tackle the inefficiency in filtering, by using contentment-based filtering together with collaborative filtering to provide recommendation with depth. The research is applied to an online newspaper.

Flaws of collaborative filtering

i. Early rater problem
Prediction for an item with little or no ratings will be poor and basically unreliable.

ii. Sparsity problem
In a normal case, the items in a system far surpass what an individual user can rate or review. Thus leaving many items that has not been rated by users

iii. Gray sheep
In this case, there are some people who will never receive the right recommendation from the system due to the fact that they will never follow a particular trend. They may agree with a group ‘A’ today, and disagree with them tomorrow, thus making it hard to find a neighborhood for the user.

**Approach to Hybridization**

The two filtering approaches were combined by basing the prediction on a weighted average of the collaborative recommendation and the content-based recommendation. By using the strength of the content-based filter, the effects of sparsity and the early rater problems are reduced. The weights of the two filter approaches are measured at a per user basis, allowing the system to determine a best mix of the two filter approaches for each user [15].

2.7.3. Item-Based Collaborative Filtering Recommendation Algorithms [1]
This paper tries to tackle the growing content (information) on the internet by recommending relevant products. It also considers the impact of recommender system on the current state of business, and views the collaborative filtering techniques are the most successful in the field of recommendation.

Traditionally in recommender systems, the amount of work increases with the increase in participants of the system, so there is need to find a more efficient way to deal with this constraint. For this issue, item-based collaborative filtering is considered.

Item-based techniques first analyze the user-item matrix to identify relationships between different items, and then use these relationships to indirectly compute recommendations for users [1].

In the paper various item-based recommendation generation algorithms are analyzed, i.e. many techniques to compute item-item similarities e.g. item-item correlation vs cosine similarities between item vectors.
Modern recommenders can recommend in seconds by searching tens of thousands of neighbors, but in reality the market needs a system that can recommend for millions of potential neighbors. Existing systems have issues with individual users with large amount of information. For instance, if a site is using browsing patterns as indications of content preference, it may have thousands of data points for its most frequent visitors. The long user rows slow down the number of neighbors that can be searched per second, further reducing scalability.

2.8. Discussion (outcome, pros, cons) & why your work
This project aims to provide a filtered result based on the query presented by the user of the system, by exploiting the existing techniques of collaborative and content-based filtering. The project will provide answers to users question to a closer degree than what they will get in a normal web search engine. Searches made in the system do not really on the result of one source, but the source from many repositories and search engines. The issue is trying to combine those many results and produce an organized output that can be filtered, and then presented to the user.

The different techniques to be used are:
- Collaborative filtering
- Content-based filtering
- Server-side programming

2.8.1. Pros and Cons of Different techniques
2.8.1.1. Details on Collaborative filtering
CF systems work by collecting user feedback in the form of ratings for items in a given domain and exploit similarities and differences among profiles of several users in determining how to recommend an item. On the other hand, content-based methods provide recommendations by comparing representations of content contained in an item to representations of content that interests the user [17].

- Firstly, CF can perform in domains where there is not much content associated with items, or where the content is difficult for a computer to analyze — ideas, opinions etc.
- Secondly a CF system has the ability to provide serendipitous recommendations, i.e. it can recommend items that are relevant to the user, but do not contain content from the user’s
profile. Because of these reasons, CF systems have been used fairly successfully to build recommender systems in various domains [6].

However they suffer from two fundamental problems:

- **Sparsity**: Stated simply, most users do not rate most items and hence the user-item rating matrix is typically very sparse. Therefore the probability of finding a set of users with significantly similar ratings is usually low. This is often the case when systems have a very high item-to-user ratio. This problem is also very significant when the system is in the initial stage of use.
- **First-rater Problem**: An item cannot be recommended unless a user has rated it before. This problem applies to new items and also obscure items and is particularly detrimental to users with eclectic (derived) tastes.

2.8.1.2. **Details on Content-based filtering [16]**

The adoption of the content-based recommendation paradigm has several advantages when compared to the collaborative one:

- **User Independence**: Content-based recommenders exploit solely ratings provided by the active user to build her own profile. Instead, collaborative filtering methods need ratings from other users in order to find the “nearest neighbors” of the active user, i.e., users that have similar tastes since they rated the same items similarly. Then, only the items that are most liked by the neighbors of the active user will be recommended;
- **Transparency**: Explanations on how the recommender system works can be provided by explicitly listing content features or descriptions that caused an item to occur in the list of recommendations. Those features are indicators to consult in order to decide whether to trust a recommendation. Conversely, collaborative systems are black boxes since the only explanation for an item recommendation is that unknown users with similar tastes liked that item;
- **New Item**: Content-based recommenders are capable of recommending items not yet rated by any user. As a consequence, they do not suffer from the first-rater problem, which affects collaborative recommenders which rely solely on users’ preferences to make
recommendations. Therefore, until the new item is rated by a substantial number of users, the system would not be able to recommend it.

Even with all this advantages, content-based filtering has its own shortcomings, and they are listed below:

- **Limited Content Analysis**: Content-based techniques have a natural limit in the number and type of features that are associated, whether automatically or manually, with the objects they recommend. Domain knowledge is often needed, e.g., for movie recommendations the system needs to know the actors and directors, and sometimes, domain ontologies are also needed. No content-based recommendation system can provide suitable suggestions if the analyzed content does not contain enough information to discriminate items the user likes from items the user does not like. Some representations capture only certain aspects of the content, but there are many others that would influence a user’s experience.

  For instance, often there is not enough information in the word frequency to model the user interests in jokes or poems, while techniques for affective computing would be most appropriate. Again, for Web pages, feature extraction techniques from text completely ignore aesthetic qualities and additional multimedia information.

  To sum up, both automatic and manually assignment of features to items could not be sufficient to define distinguishing aspects of items that turn out to be necessary for the elicitation of user interests.

- **Over-Specialization**: Content-based recommenders have no inherent method for finding something unexpected. The system suggests items whose scores are high when matched against the user profile, hence the user is going to be recommended items similar to those already rated. This drawback is also called *serendipity* problem to highlight the tendency of the content-based systems to produce recommendations with a limited degree of novelty. To give an example, when a user has only rated movies directed by Stanley Kubrick, she will be recommended just that kind of movies. A “perfect” content-based technique would rarely find anything *novel*, limiting the range of applications for which it would be useful.

- **New User**: Enough ratings have to be collected before a content-based recommender system can really understand user preferences and provide accurate recommendations. Therefore,
when few ratings are available, as for a new user, the system will not be able to provide reliable recommendations.

2.8.1.3. Details on Server-side programming (Servlet/JSP)

- Performance
  The performance of servlets is good because there is no process creation for each client request. Instead, each request is handled by the servlet container process. After a servlet is finished processing the request, it stays resident in the memory, waiting for another request.

- Portability
  Similar to other Java technologies, servlet applications are portable. You can move them to other operating systems without serious hassles.

- Rapid development cycle
  As a Java technology, servlets have access to the rich Java library, which helps speed up the development process.

- Robustness
  Servlets are managed by the Java Virtual Machine. As such, you don’t need to worry about memory leak or garbage collection, which helps you to write robust applications.

- Widespread acceptance
  Java is a widely accepted technology. This means that numerous vendors work on java-based technologies. One of the advantages of this widespread acceptance is than you can easily find and purchase components that suit your needs, which saves precious development time [3].

2.9. Proposal

This project aims to use the already established filtering techniques, to tackle the problem at hand, which is recommending webpages from different sources that’s close to what the user needs.

The proposed idea for the system is as stated below:
i. User queries the system through a mobile (Android) device, through the Edit-text element in the android application, and the value from the Edit-text element is sent though a GET method or a POST method to the server.

ii. The system receives the query through the use of HTTP, to search the web for articles or pages with related topics.

![Diagram of search engines](image1.png)

*Figure 32 Gathering data from search engines*

iii. The system searches the web for related materials, i.e the system scrapes information from some search engines, and combine those result into a list or a Hashmap of Title and URL. There are some cases where the title of a page is not unique, so the list may be considered as a two dimensional arraylist.

![Diagram of storing webpage data locally](image2.png)

*Figure 33 Storing webpage data locally*
iv. Based on the list of gathered materials, the system checks if it has been recommended before. First checking the table of rated pages by comparing the titles, then through the URL if the title matches.

![Figure 34 Checking local pages to see if it has been rated](image)

v. The system will recommend a related page if it finds a webpage that has been added to the database. By finding the neighborhood based on the user or item, the system searches for items that meet a certain rating, then recommends it.

![Figure 35 Item-based collaborative filtering using an already rated item](image)

vi. The System recommended pages are then added to the main list if they were not there before. After a successful collaborative recommendation, the system appends its
recommended pages to the main list, if not; the system will not give a collaborative recommendation for the particular query.

![Diagram showing the process of adding recommended URLs to local data]

Figure 36 Adding recommended URLs to local data

vii. After the list has been processed by the collaborative filter, the system moves it to the content-based filter.

viii. The list presented to the content-based filter is a list of URLs. So the system will have to visit or download a temporary copy of the content of the webpages before the content-based filter can produce its recommendation based on the queried words.

ix. After the content-based filter is finished filtering, the system is given the final list to be presented to the mobile device user (as JSON).

x. The device user sees the result in the form of a list in an Android listview element that was populated by a JSON input from the server.

xi. For Simplicity purpose, the system will only remember (store) the pages that the user visits and rates.

xii. If a user opens a page in the web-view of the Android application, and rates the page, then the system is sent a request to add the page and its rating by the user to the stored database of pages.
3. Application / Case Study / Benchmarking

3.1. Case Study

Let us consider using the two recommender systems together to form an hybrid that will be used to recommend webpages or web-document to users based on the fact that another user has rated it and also for the fact that the weight of the document is high enough to be acknowledged by the system and recommended to the users.

In this case we will be using an Android application to request data from the server which will be developed using multiple servlets.

Each recommendation made to a user is constrained by the search query which the user searched for. The sources we will be getting our data from will be limited to 4 place for the case of this study, and they are Google.com, Bing.com, Yahoo.com, and Ask.com.

Every of the URLs that will be gotten from the search engines have to be stored temporarily for other systems to make use of.

For each of the URLs grabbed, if they have been rated before, the system should make recommendation for other items that a similar, and if the item has the same tags of the same terms to the search query then it will be added to the locally also.

Every local copy of the URLs have to be scored by the content-based recommender which leverages on the TFIDF weighting. After the documents have been their score that’s when they can be represented in JSON format to the Android device.

Every link that the user clicks will be added to the table called ‘item’, and if the user chooses to rate the item, then the system will add the rating to the table ‘rating’.
3.2. ER Diagram

![ER Diagram of the whole system](image)

*Figure 37 ER diagram of the whole system*

3.3. Building Dataset

At the infancy on the project, the system had few or no data in the dataset. The project uses a MYSQL database to store data, both permanently and temporarily. There was need to build the other parts of the system in order to make a proper dataset.

The dataset is of the form “USERID”, “ITEMID”, “VALUE”.

- USERID is the id of a user of the system in the table “USER”.
- ITEMID is the id of an item (URL) in the database table “Item”
- VALUE is the rating score given to the ITEMID by the user with USERID.

All of these details are gathered from a table “Rating” in the database of the system. So inorder to have a dataset which is going to be need by the collaborative recommender to give recommendation, there is need to provide data into the system, and also find a way to rate the items in the data.
3.4. Query the System through Android and HTTP

We are querying the server of my application via the HTTP and in order to do that from an android application. An Android application was created to receive input from users.

![Figure 38 EditText and button of Android application](image)

Through the EditText field provided by android, users can add text to the application, the text can be acquired, and used to request data from the server.

Android applications sends a request to the web server through an asynchronous method that runs in the background, and after the background task is over the application does an onPostExecute method that runs after the background task in done.

The server is hosted on a Tomcat server and thus runs on a servlet/JSP system. All the server codes are servlets, and they get their information via a GET method.

3.4.1. Querying the server via emulator

Running the code in an emulator requires the user of a special IP to access the Localhost. Normally the localhost is equal to 127.0.0.1 on the host machine, but in the Android emulator it is 10.0.2.2. The Tomcat server in my machine works on the port 8080, so to access the server from the emulator, address must be preceded by “http://10.0.2.2:8080”.

3.4.2. Querying the server via a physical device.

To send a request to the server, then the device is connected to the same network as the host-machine. The IP of the host is acquired and that will be the precedent to access the server via http. The IP in use in this project is a static IP “192.3.168.8.100” for the host machine. He code
is running locally on my machine, the host machine and the android device are connected to the same network.

3.5. Gathering Information

In this system, we are gathering information from different locations (mainly search engines). The sources are 4 in number, namely:

- Google
- Yahoo
- Bing
- Ask

The project is not restricted to this engines, but for the aim of gathering information, we are using search engines.

The main tools used here are the java and a third party java library called JSoup.

3.5.1. Getting URL links

Algorithm:

- Create a query from user request to the server based on the search engine e.g. http://google.com/search?q=term1+term2+...termN
- Get the webpage as a document
- Get all hyperlink elements from the document.

3.5.2. Removing irrelevant links

Algorithm:

- Get every link in the document
- Discard all link that portrays search-engines extra e.g. Adverts, links to more searches etc.
Figure 39 Related search links in a Bing search.

Trying to decipher which link is relevant and which one is not can be very tasking, so to save time and not to do all the work of the recommender from the gathering of information, the absolute links that starts with or has text like “google.com”, “http://webcache”, “bing.com”, “yahoo.com”, “flickr”. Though with a paid service, the search engines can give custom search results in formats that can be easily used by developers.

- Remove the extension of a single website into multiple categories e. g. Wikipedia
  Most site can have multiple extension of the same page. Like Wikipedia, for example has a page on “recommender system” that same page has segment that can be referenced by the #-sing followed by the identifier.

  - http://en.wikipedia.org/wiki/Recommender_system#Overview
  - http://en.wikipedia.org/wiki/Recommender_system#Approaches
  - http://en.wikipedia.org/wiki/Recommender_system#Beyond_Accuracy
All of the links above point to the same page, so it is inefficient to capture all of them, instead it is better to capture the first one.

- Return list of URLs for search engine

For each search engine, there is a method assigned to it. The method makes a HTTP request from its corresponding search engine and download the result as a Document object. From this document the relevant links are extracted before the method returns all the links extracted in form of an ArrayList which will be used for further processing.

3.5.3. Avoiding Duplicates

Algorithm:

- Create a new list

  A list is created to accommodate the combined results from all the methods returning web links.

- For each search engine’s list, add URL to new list, if no duplicate is found.

\[
\begin{array}{cccc}
\text{Source 1} & \text{Source 2} & \text{Source 3} & \text{Source 4} \\
\end{array}
\]

\[
\text{Combined List}
\]

*Figure 40 Combining the result from various sources*

- Return Combined List

3.5.4. Making content temporarily available

In order for the contents of a webpage to be ranked by the content-based filter, there is need to capture the details of the page before-hand. To achieve this, a table was created to capture these details.

Since a search is unique to a user, there was need to make sure nobody was using the search result for another user. So every page saved is must be under a USERID.

The table below depicts the exact structure of the MYSQL table “Temp”;
<table>
<thead>
<tr>
<th>tempid</th>
<th>title</th>
<th>url</th>
<th>body</th>
<th>Score</th>
<th>user_user_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>Null</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>Null</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>Null</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>Null</td>
<td>2</td>
</tr>
</tbody>
</table>

*Figure 41 Table structure of database table 'temp'*

- **tempid**: is the temporary ID of the local content
- **title**: is the title of the page made local. These filled can be redundant, but it is needed for displaying on the android device as result.
- **url**: is the link of the webpage
- **body**: is the text content of the HTML page which are inside the <body> tag
- **score**: is the final value that is to be assigned to the page by the content-based filter.
- **user_user_id**: is a foreign Key of a user in the table “user”. It is also the ID of the user who made the search.

### 3.6. Collaborative filtering of URL-List

In this stage, the system will need to find items (URLs) with similarities with the ones on the list. In this case, the system checks all the URLs in the list which is now locally available in the database table “temp”.

For each URL, the system checks if it has an itemid from the database table “item”. Only items that have been view by at least one user has an itemid.

If they have itemid, separate them from the ones without id, i.e. make a list of itemids and store them there.

Check to see if they have been rated before. This is done using by checking the “rating” table in the database. A URL that has been rated is needed to make recommendation for other rated URLs. If the itemid is found in a rating, then add item id to another list of rated-itemid.
The collaborative filter is done using an Apache Mahout Library [4], and this library needs a dataset of “userid, itemid, rating-score” e.g.

```
196,242,3
186,302,3
22,377,1
244,51,2
166,346,1
298,474,4
115,265,2
253,465,5
305,451,3
6,86,3
62,257,2
286,1014,5
200,222,5
210,40,3
224,29,3
303,785,3
122,387,5
194,274,2
291,1042,4
224,1184,2
```

*Figure 42 Dataset from the MovieLens database*

The collaborative filtering system creates a datamodel from the file containing the dataset (a Comma Seperated File CSV). From the datamodel created, it finds the similarities for each of the items in the data. So based on the shortlist, for each of the items (URLs) in our list, the system recommends N-items. This items are then stored into a separate list and then returns.
Due to the fact that the recommender doesn’t know the contents of its recommendation, it can recommend things that might be interesting to read but can also be irrelevant to the search query of the user. So there is need to tag items with terms that were used to search them. Based on these tags, the system will rank the items recommended by the collaborative filter.

Every item in the table “item” has at least 1 tag-term, based on the current search query by the user, the system will score each item. For each tag in query, if an item has the same tag, its score increases by one.

<table>
<thead>
<tr>
<th>URL</th>
<th>TAGS</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>url1</td>
<td>t2, t7</td>
<td>2</td>
</tr>
<tr>
<td>url2</td>
<td>t3</td>
<td>1</td>
</tr>
<tr>
<td>url3</td>
<td>t4</td>
<td>0</td>
</tr>
<tr>
<td>url4</td>
<td>t4, t7, t9, t13</td>
<td>1</td>
</tr>
</tbody>
</table>

Where:

\( T = \{t_1, t_2, \ldots t_N\} \) is the set of all possible terms

\( \text{Query} \subseteq T \)

With this score the system will be able to provide a better, and useful recommendation that is relevant to the current search query. Each URL in the list with high enough score, will then be added into the local table “temp” in the database, if there is no duplicate found.

3.7. Content-based filtering of final List

This is the stage of the project where the local URLs in the table “temp” can be rated and assigned a score based on their weight for the search query.
3.7.1. Getting documents weight
For all N terms in a query, a document has N weights. So the combined weight is needed, and this is done by adding the weight of the document for each term.

Note: We are considering every row in the table “temp” as a document.

To get the Term Frequency Inverse Document frequency (TFIDF) of a document, there is need to get the TF of each document and the IDF of the collection of the document.

3.7.2. Getting Term Frequency (TF)
In this area, from the database table “temp”, the system gets every document that is locally available for the USERID it is working on. It captures each document and put them in a Hashmap variable, using the tempid as its key and the body as the value.

```
sql = "SELECT * FROM tom.temp where user_user_id="+user;
```

For each term, the system gets the following:

- The number of the current term in the document
- The number of unique terms in the document

The system gets the TF of each term for each document in the corpus.

3.7.3. Getting the Inverse Document Frequency (IDF)
The Document Frequency is per term bases. It is not specific to a document. i.e. the IDF on doc1, doc2, doc3 in a particular collection for a term t1 is the same.

To get the IDF we have to get:

- X | The Number of document in the collection
- Y | The number of document with the term
- Do the log of (X / Y) to get the IDF for the term

3.7.4. Computing TFIDF
With the acquire TFs that was gotten from each of the documents for various term, and the IDF for the term on the collection of documents, the TFIDF score of the document for a term can be computed, e.g.
Query = t1+t2+t3

Documents = d1, d2, d3

<table>
<thead>
<tr>
<th></th>
<th>TF for t1</th>
<th>TF for t2</th>
<th>TF for t3</th>
<th>IDF for t1</th>
<th>IDF for t2</th>
<th>IDF for t3</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1</td>
<td>0.01</td>
<td>0.23</td>
<td>0.45</td>
<td>0.33</td>
<td>0.66</td>
<td>0.33</td>
</tr>
<tr>
<td>d2</td>
<td>0.20</td>
<td>0.12</td>
<td>0.01</td>
<td>0.33</td>
<td>0.66</td>
<td>0.33</td>
</tr>
<tr>
<td>d3</td>
<td>0.10</td>
<td>0.17</td>
<td>0.00</td>
<td>0.33</td>
<td>0.66</td>
<td>0.33</td>
</tr>
</tbody>
</table>

TFIDF for t1 = (TF for t1,d1) * (IDF for t1)

Every TFIDF score for a document is added together to get the combined score of the document for the search query.

After the weight is ready, then there is need to register the score to the temporary local database table of documents called “temp”.

The table is updated with the score of each document with an SQL update statement that is of the form below:

```
UPDATE tom.temp
    SET score = CASE tempid
        WHEN 1 THEN 12
        WHEN 2 THEN 42
        WHEN 3 THEN 32
        WHEN 4 THEN 51
    END
WHERE user_user_id = userid
```

3.8. Making a JSON output for Android device

In order for the android application to update its listview with the data gotten from the web-server, it has to be sent in a format that is compatible or useful to android. The format that is in use is JavaScript Object Notation (JSON).

Below is the SQL query to capture the document URLs and their scores for a user’s search:
sql = "SELECT * FROM tom.temp WHERE user_user_id="+userid+" ORDER BY score DESC";

The string or JSON output generated by the system is the final process of the recommendation, the JSON is response to the request made by the user from the android device. With this, the ListView will be populated.

Below is a sample of the JSON output that the server sends to the client as a response:

```json
{
  "ranklist": [
    {"title":"title1","url":"url1","score":"score1"},
    {"title":"title2","url":"url2","score":"score2"},
    {"title":"title3","url":"url3","score":"score3"}
  ]
}
```

3.9. Android Application

The Android application is the interface for this case study. The application contains the following:

- EditText which is used to accept user’s search query which is sent to the web-server
- Search button to send the HTTP request with the content of the EditText.
- ListView that is populated by the response from the server, and it is clickable.
- WebView displays the HTML view of a URL that is on the ListView.
Figure 43 Android application populated with URL recommendations from server
3.9.1. Adding Item to database

This application send all the requests required to send to the server. Items selected by the user form the listview has to be added to the database.

When a user clicks on one of the listitems in the listview, the system does 3 thing:

i. Loads the WebView with the link gotten from the ListItem
ii. Sets the RatingBar to Zero
iii. Sends a request to the server in the background to add the URL in the table “item”

Example:

&query=query

In the servlet code, the ‘opr’ parameter is grabbed and the corresponding method is called. For the case of the above query, the AddItem method is called, and the method takes in 2 parameters.
AddItem.add(url, query)

The Java method above adds the URL to the table ‘item’, and uses the terms in the query to assign tags to the URL.

![ER diagram to show the Item-Tag-Term relationship](image)

3.9.1.1. Removing Stop-words

Consider the labels “Where do you want to go?” and “When do you want to travel?”. The words ‘do’, ‘to’, ‘when’, ‘where’ and ‘you’ are commonly regarded as not conveying any significant semantics to the texts or phrases they appear in. Consequently, they are discarded. This kind of words are called stop words [8].

Before adding tags to a URL, we have to determine which word is worth tagging to a URL, so the system filters out the terms that are considered stop-words.

The array of stop-words used are the follow:

There is not one definite list of stop words which all tools use and such a filter is not always used. Some tools specifically avoid removing them to support phrase search.

3.9.2. Adding Rating to database

The rating of an Item (URL) is essential to the collaborative filtering aspect of this case study. The Android application adds a rating to the database, when the listview is populated with links
to webpages. When a user clicks on the listview, it loads the webView which in turn adds the URL into the table ‘item’, but to rate the URL, the ratingBar is used for this purpose.

![Figure 46 Rating in Android Application and in Server back-end](image)

The application makes a HTTP request to the server to rate the item. The HTTP request looks like the text below:

```
```
On the server side the server gets the ‘opr’ parameter to get the method to call. In this case, the system will call an AddRating method that takes 3 parameters ‘item’, ‘user’, ‘rating-score’.

3.9.3. Adding Users to database
To use the system, you have to be a registered user of the system. To register a user to the system, the application provides an android activity to register a new credential. Without the proper identification of a user in the system, there will be trouble giving recommendations to other users based on activities of a user that cannot be identified.

![Register Activity in Popup form](image1)

*Figure 47 Register Activity in Popup form*

![Failed: Try changing the username](image2)

*Figure 48 When a user picks a username that exists on the database*
The application makes a HTTP request to the server to add the new user. The HTTP request looks like the text below:

http://localhost:8080/servlet/process?opr=adduser&username=amanze&password=1

The server identifies the method to call from the ‘opr’ parameter, and calls the AddUser method that takes 2 parameters ‘username’, ‘password’.

3.9.4. Authenticating user

The Android application prevents users that are outside the database from logging-in. The helps the system to keep track of user’s web activities.
The application sends the following HTTP request to the server, and the server responds with either a success message or a failed login message.

```
http://localhost:8080/servlet/process?opr=login&username=amanze&password=1
```

![Figure 51 Login Activity in form of a Popup](image1)

![Figure 52 Successful login in the Login Activity](image2)
3.9.5. Screenshots

**Figure 53** Android application waiting for server’s response

**Figure 54** Adding URL to database and viewing it on the Web-View, by clicking a List-Item
Figure 55 Rating an item (URL) using a rating bar
4. Conclusion

4.1. Work done

In this project work, I have been able to get URLs from search results, since our focus is to recommend webpages that are close to the preference of users. The system was also able to score each of the documents, and by their score, we have ranked the documents and displayed them on an Android device for the users to use as they choose. Links chosen by the user are added to the database. Even the ratings made by users on item are added to the database, and this aids the collaborative aspect of the recommendation system.

A hybrid-recommender (Content-based + Collaborative) at a young stage of its existence will provide recommendation, but mainly from the content-based section. The result from the collaborative portion will most likely be irrelevant to the context at the early stages, but as the amount of data increases, the collaborative recommender will strongly complement the content-based.

4.2. Contribution & Limitations

This project was able to make recommendation of webpages that were of plain text, and not that of special text like PDF, DOC, DOCX, EPUB etc.

The speed of operation is fairly slow when gathering data from the internet. This depends heavily on the speed of the internet that the host machine is connected to.

The project provides more insight on the fact that when recommender is young, the collaborative aspect of it provide little or no recommendation. Most of the time the recommendations presented by the recommender might not be useful to the context at hand.

The approach of getting webpages from search engines is tedious, but if there is a better way e.g. a schools repository, then the process will be less tedious, and straightforward.

4.3. Perspective

This process of recommending documents can be used in many forms. Schools or organizations with wealth of information can make use of the filtering processes and recommend material to their users.
It can also be used to decipher what is important from what is not. Given the user or the system the choice using or discarding irrelevant materials.
5. References


[4]. Cataldo Musto, Ph.D. Apache Mahout –Tutorial, University of Bari, 2014


[10]. Ho Chung Wu And Robert Wing Pong Luk The Hong Kong Polytechnic University, Kam Fai Wong The Chinese University of Hong Kong, And Kui Lam Kwok Queens College, City University of New York, Interpreting TF-IDF Term Weights as Making Relevance Decisions, 2008.

[11]. Kazunari Sugiyama, Kenji Hatano, Masatoshi Yoshikawa, and, Shunsuke Uemura, Improvement in TF-IDF Scheme for Web Pages Based on the Contents of Their Hyperlinked Neighboring Pages, Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma, Information Technology Center, Nagoya University, Nagoya, 464-8601 Japan, 2005.


[17]. Prem Melville and Raymond J. Mooney and Ramadass Nagarajan, Content-Boosted Collaborative Filtering for Improved Recommendations, Department of Computer Sciences, University of Texas, {melville, mooney, ramdas}@cs.utexas.edu, American Association for Artificial Intelligence, www.aaai.org, 2002.

[18]. Robin van Meteren and Maarten van Someren, Using Content-Based Filtering for Recommendation, NetlinQ Group, Gerard Brandtstraat 26-28, 1054 JK, Amsterdam, The Netherlands, robin@netlinq.nl. University of Amsterdam, Roeterstraat 18, The Netherlands, marten@swi.psy.uva.nl.


Figure 56 Class Diagram of Server
Appendix A2

Figure 57 Class Diagram for Server Cont.
Appendix A3

Figure 58 Class Diagram of Android Application