University POLITEHNICA of Bucharest

Faculty of Automatic Control and Computers,
Computer Science and Engineering Department

BACHELOR THESIS

Article Recommender System

Scientific Adviser:
Ing. Mihai Alexandru Ciorobea
Sl.Dr.Ing. Razvan Deaconescu

Author:
Theodor-Cosmin Didii

Bucharest, 2015

I would like to thank my project supervisor,
Mihai Alexandru Ciorobea, that helped and guided me
through the realisation of this project.

Also, I would like to thank Razvan Deaconescu
for his unending support in the redactation
of the bachelor thesis.

Abstract

This thesis presents a recommendation system for articles that are rich in content and have certain attributes. This system was needed because a tool that exploited all the properties of an article, at their maximum potential, was not that accessible. The first step toward choosing an appropriate algorithm for your problem is to decide upon which attributes of your entities you want to focus. Since this is a recommendation system for articles, the main focus is going to be on article content and article categorization. Other recommendation systems take into account only the content of the articles and do not give any importance to the date, author, language, and ratings of an article. Also, most of the already existing recommendation systems create and maintain the data in their own database, thus doubling the required storage space for an application.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recommendation systems have become a major research area since the appearance of the first paper on collaborative filtering in the mid 1990s and have become popular in both commercially and research communities. The first step toward choosing an appropriate algorithm for your problem is to decide upon which attributes of your entities you want to focus. Since this is a recommendation system for articles the main focus in going to be on article content and categorization. Other recommendation systems take into account only the text of the articles and do not give any importance to the date, author, language, and ratings of an article. Also, most of the already existing recommendation systems create and maintain the data in their own database, thus doubling the required storage space for an application. In this thesis we will describe a recommender system that solves all the mentioned problems of the existing systems.

1.1 Motivation

Adobe is working on a new project that helps content producers like National Geographic and Fast Company bring their content on the web. This project does not have a recommender system. At the moment the recommendations are made by hand. Thus, a specialized recommender system was needed for their articles and specific attributes. Recommender systems that use a part of the attributes already exist, but none of them use all the attributes and are not as easy to extend. Because not all the attributes are used, the recommendations given are not as good.

1.2 Objectives

The main objective is to build a stand-alone application the acts as a Restful API and can offer multiple recommendation options. There should be two types of recommendations:

1. Recommendations based on content (related articles), thus, solving the cold start problem.

   This type of recommendations may use collaborative filtering if specified by the user of the system.

   This type of recommendations should make use of all the attributes of an article and give precise recommendations.

   This type of recommendations should make use of all the attributes of an article and give precise recommendations.

Each attribute should be given a certain importance in the classifying of the articles.

2. Recommendations based on user history and ratings, giving personalized recommendations for a certain user.

1.3 Related Work

A great amount of research has been carried out on the topic of recommendation systems, both in the academia and in the industry. There is a great deal of diversity in the solutions that currently make up the state of the art in the field, both in regard to algorithms used and the recommendation strategies employed.

In the following sections, I will present a couple of the most successful recommendation systems at the moment.

1.3.1 Search Engines (Google, Yahoo, Bing, etc)

These are the most famous type of recommendation systems. They are based on both the content of a site and the particular preferences of an user, determined by their previous search history.

All these systems offer an user the possibility to query all the indexed websites using a certain phrase or combination of words.

The one that offers a tool most similar to an articles recommendation system is represented by advanced Google search. It offers the possibility to find sites that are similar to a web address you already know, but the recommended pages are not that good of a match. By using certain keywords and doing a search based on them, the resulted articles were more alike.

Also, most of the search engines are held by private companies and act like black boxes. You do not have any access to their data and algorithms.

1.3.2 Apache Solr

Solr is an open source enterprise search platform, written in Java, from the Apache Lucene project. Its major features include full-text search, hit highlighting, faceted search, real-time indexing, dynamic clustering, database integration, NoSQL features and rich document handling. Providing distributed search and index replication, Solr is highly scalable and fault tolerant. Solr is the most popular enterprise search engine.

Solr is written in Java and runs as a standalone full-text search server. Solr uses the Lucene Java search library at its core for full-text indexing and search, and has REST-like HTTP/XML and JSON APIs that make it usable from most popular programming languages. Solr's powerful external configuration allows it to be tailored to many types of application without Java coding, and it has a plugin architecture to support more advanced customization.

This solution needs to do its own indexing before any recommendations can be made and it can't be directly integrated with another database. Thus, requiring the duplication of data. This system does not take into account the previous user search history and does not make personalized recommendations.

1.3.3 Duine Recommender

The Duine recommender is a software module that calculates how interesting information items are for a user. The resulting interest is quantified by a number, called prediction, ranging from -1 (not interesting) to +1 (interesting). When applied in, for example, an electronic TV Guide the Duine recommender can calculate how interesting each TV program is for a particular user. These predictions can be used in various ways: e.g. to provide a user with a list of the top 10 most interesting items, to sort a list of items with the items with the highest prediction at the top, or to present an indication of the interest to the user for each item (e.g. using a number of stars).

The Duine recommender also processes and stores ratings that users give to an information item and interests of the users in aspects of the information (categories, genres, people, topics etc). All data associated with a user is stored in a user profile.

The Duine recommender has learning capabilities. When a user rates an information item, the recommender extracts data from this item (e.g. keywords or genres of a TV program description) to determine the interests of the user. By using smart learning algorithms the recommender slightly adapts the user profile after each rating, based on these interests.

One of the disadvantages of Duine is that it does not offer recommendations based only on content so, it wouldn't be a good fit for the presented problem.Also, it doesn't solve the cold start problem very well, since it's based only on learning and prediction algorithms.

Table 1.1: Solution Comparison

Recommendation System	Collaborative Filtering	Content based	Hybrid	Open Source	Integrates with Existing Database
Search Engines	✓	✓	✓	✗	✗
Apache Solr	✗	✓	✗	✓	✗
Duine	✓	✗	✗	✓	✓
Our Recommender	✓	✓	✓	✓	✓

TODO DELETE THIS: E VREO PROBLEMA CA M-AM CONCERNTRAT PE CE REZOLV EU SI NU REZOLVA EI SI NU INVERS?

Chapter 2

Recommendation Algorithms

There are 3 types of recommendation algorithms:

1. Recommendations based on content

   These type of recommendations are solely based on the content of the item to be recommended.

   It uses various text based algorithms in order to calculate a similarity between two items.

2. Recommendations based on collaborative filtering

   These type of recommendations are based on previous user history and ratings.

   The main purpose is to find users with simillar interests to a certain user and recommend items that that are preferred by most of the users.

3. Hybrid recommendations

   These type of recommendations combine the previous presented.

All 3 types of algorithms can be used, independently or by combining the obtained results.

In the following sections we are going to present the implemmented algorithms.

2.1 Algorithms for Related Articles

The related articles recommendation returns a list of articles related to a certain article.

We have implemmented both a hybrid and a content based approach.

A user may choose to use only one approach or combine them for better results.

2.1.1 Collaborative Filtering

In order to obtain better results, we may use collaborative filtering to do a kind of article grouping by user preferences. Usually, users fit into a certain profile and will read articles that may be related to each other. Using this technique we can reduce the number of articles on which we are going to apply the article similarity algorithm. We use the following iterative algorithm[2] to find articles that are related to article A1:

```
For each user U that read A1
    For each article A2 read by user U
        Save in common article list that a user read both A1 and A2
For each article A2 in common article list
    Compute the similarity between A1 and A2
    Save the computed similarity in related article list
Sort the related article list by similarity in descending order
Return the related article list
```

Listing 2.1: Item to item collaborative filtering

2.1.2 Computing Article Similarity

In order to obtain the best related article list we have to take advantage of all the attributes of the articles at our disposal. We compute the similarity using the:

- Date created: used to chose between two articles that have the same related score with the current article
- Title: find the similarities in the titles of two articles by using natural language processing
- Short title: find the similarities in the short titles of two articles by using natural language processing
- Department: check if the articles belong to the same department by comparing the strings
- Category: check if the articles belong to the same category by comparing the strings
- Importance: articles with the same importance are shown upper in the related articles
- Publication: if the articles belong to the same publication the similarity is going to be greater
- Language: the articles should be written in the same language in order to be related
- Author: if the articles have the same author the similarity is going to be greater
- Keywords: if the articles have more common keywords determined by natural language processing analysis then the similarity is going to be greater
- Ratings: if the articles have more readers and a higher mean rating, the similarity is going to be greater
- Collection Reference: if the articles belong to the same collections the similarity is going to be greater
- TFIDF (Term Frequency Inverse Document Frequency) similarity

Each of these fields has a certain importance in the final resulted similarity between articles.

2.1.2.1 Natural Language Processing

In order to compute and improve the similarity between two strings we employ the following three methods:

1. Levenshtein Distance

   We compute the number of characters needed to change one string to another and divide this value by the length of the bigger string. We obtain a value between 0 and 1.

   This method is faster than the next one and is better suited for generating related articles on the fly.

2. By comparing character pairs

   We compute the number of common character pairs of the two strings, multiply it by two and divide it by the number of combined character pairs. In the end, we will obtain a value between 0 and 1.

   This method gives better results than the previous one.

3. Lemmatization

   In order to obtain better TFIDF results we use the Stanford NLP (Natural Language Processing) framework for lemmatization.

   Lemmatization is the process of doing a vocabulary and morphological analysis on a text in order to reduce the words to their proper base form.

   It is important to not confuse lemmatization with stemming.

   Stemming usually refers to a crude heuristic process that just chops off the end of a word in order to obtain good results, most of the time. This is usually achieved through the removal of the derivational affixes.

   For instance:

   (a) The word "*am*" will be reduced through lemmatization to the base form "*be*". This form can not be deducted through stemming, as it requires a dictionary look-up.

   (b) The word "*meeting*" can either be a noun, in which case the lemma is going to be "*meeting*", or the ing form of the verb "*meet*". This difference can be deducted by the lemmatizer through the understanding of what type of speech the word actually is from the context. Stemming will reduce the word to the form "*meet*", which may not be correct in certain contexts.

   (c) The word "*walking*" has the lemma "*walk*", form which can be deducted both through stemming and lemmatization

   TODO Write more about it

2.1.2.2 Term Frequency Inverse Document Frequency (TFIDF)

In order to compute the TFIDF similarity we use the following formulas:

1. Computing the term frequency of a term T in a document, D

$$tf(T,D) = \sqrt[2]{\frac{tfreq(T,D)}{tnum(D)}}. \tag{2.1}$$

   Where tfreq (T, D) is the number of occurrences of a term T in the document D and tnum (D) is the total number of terms in D. A tf value depends on the number of term occurrences. This method cannot characterize documents accurately because it is not able to distinguish important words from trivial words sufficiently. In order to obtain a better TFIDF value we also use stemming and lemmatization on the available words.

2. Computing the inverse document frequency of a term T in a document.

$$idf(t, D) = \log \frac{N}{|t \in D|}. \tag{2.2}$$

   Where N is the total number of documents in the corpus and $|t \in D|$ is the number of documents where the term t appears. If the term is not in the corpus, this will lead to a division-by-zero. It is therefore common to adjust the denominator to $1 + |t \in D|$

3. Computing the TFIDF [4]

$$tfidf(T, D_1) = tf(T, D_1) \times idf(T, D_1) \tag{2.3}$$

4. Computing the TFIDF similarity

   We use the Cosine similarity in order to do this.

$$\begin{aligned} CosineSimilarity(D_1, D_2) = \cos(\theta) &= \frac{D_1 \times D_2}{||D_1|| \times ||D_2||} \\ &= \frac{\sum_{i=1}^{n} tfidf(T_i, D_1) \times tfidf(T_i, D_2)}{\sqrt[2]{\sum_{i=1}^{n} tfidf(T_i, D_1)^2} \times \sqrt[2]{\sum_{i=1}^{n} tfidf(T_i, D_2)^2}} \end{aligned} \tag{2.4}$$

   Because the Cosine similarity doesn't take into account the number of common words or the article size, it does not give good results on it's own, so, we use the following formula, where D1 is the document for which we want the related article list:

$$\begin{aligned} similarity(D_1, D_2) = (CosineSimilarity(D_1, D_2)) \times \\ \frac{(NumberOfCommonWords(D_1, D_2))}{max(NumberOfWordsInD_1, NumberOfWordsInD_2)} \times \\ \frac{1}{\sqrt[5]{NumberOfWordsInD_2}} \end{aligned} \tag{2.5}$$

2.2 Algorithms for Recommended Articles

An algorithm used to find users with similar interests and recommend items.

In order to find users with similar interests, we can use the Pearson correlation:

$$P_{a,u} = \frac{\sum_{i=1}^{m} (r_{a,i} - \overline{r_a}) \times (r_{u,i} - \overline{r_u})}{\sqrt[2]{\sum_{i=1}^{m} (r_{a,i} - \overline{r_a})^2 \times \sum_{i=1}^{m} (r_{u,i} - \overline{r_u})^2}} \tag{2.6}$$

Where r $_{a,i}$ is the rating given by user a to item i and $\overline{r_a}$ is the mean rating given by user a to the corated items.

In order to find items that may interest the user, we can use the following formula:

$$p_{a,i} = \overline{r_a} + \frac{\sum_{u=1}^{n} (r_{u,i} - \overline{r_u}) \times P_{a,u}}{\sum_{u=1}^{n} P_{a,u}} \tag{2.7}$$

Where $p_{a,i}$ is the predicted rating of user a for item i.

Chapter 3

Database Architecture

The system uses Hbase as it's default database for storing data, but can be easily integrated with any kind of database by passing a java class for working with the database entities.

HBase is an open source, non-relational, distributed database modeled after Google's BigTable and written in Java. It is developed as part of Apache Software Foundation's Apache Hadoop project and runs on top of HDFS (Hadoop Distributed Filesystem), providing BigTable-like capabilities for Hadoop. That is, it provides a fault-tolerant way of storing large quantities of sparse data (small amounts of information caught within a large collection of empty or unimportant data, such as finding the 50 largest items in a group of 2 billion records).

Hbase is divided into tables. Each table has a name. Then, each table has multiple row keys. Each row key can have multiple families (columns). Each family can have multiple qualifiers. Each qualifier has a value.

Table 3.1: Hbase database format

Row key	Family		Family		Family	
	Qualifier	Qualifier	Qualifier	Qualifier	Qualifier	Qualifier
	Value	Value	Value	Value	Value	Value
Row key	Family		Family		Family	
	Qualifier	Qualifier	Qualifier	Qualifier	Qualifier	Qualifier
	Value	Value	Value	Value	Value	Value

We are working with hbase because it grants us fast access to our article data.

3.1 Database entities

We have mapped our database over the attributes that are needed for the recommendation and related algorithms.

3.1.1 Users

- user id - String
- user preferred categories - array of strings
- top friends - array of user ids

- item history - array of item ids
- items recommended directly by the user - array of item ids
- items recommended for the user - array of item ids (caching)

3.1.2 Items

- item id - String
- name - String
- contenturl - String
- date created - date
- title - String
- short title - String
- keywords - array of strings
- department - String
- category - String
- collection references - array of collection URLs
- author - String
- rating - double

3.2 Database Design

Besides the basic information, we need to store some extra data in order to speed up the processing time of the recommended and related articles.

3.2.1 User Table

Table 3.2: User table format

Row key				
User id	Preferred categories	Top friends	Item history	Items recommended directly
	Preferred categories	Top friends	Item history	Items recommended directly
	array list of categories	array list of user ids	array list of item ids	array list of items

TODO

- Explain why you need each item
- Explain why you need this

Table 3.3: Item table format

Row key							
Item id	Name	Content URL	Date created	Title	Short Title	Keywords	Department
Item id	Name	Content URL	Date created	Title	Short Title	Keywords	Department
String	String	String	Long	String	String	Array of String	String

Row key						
Item id	Category	Collection references	Author	Ratings		
Item id	Category	Collection references	Author	User id	User id	User id
String	String	Array of collection URLs	String	Double	Double	Double

Row key				
Item id	TFIDF			Content
Item id	Word	Word	Word	Content
String	Double	Double	Double	String

3.2.2 Item Table

TODO

- Explain why you need each item
- Explain why you need this

3.2.3 TFIDF Table

Table 3.4: TFIDF table

Row key
Total File Appearences
Total File Appearences
Integer

TODO

- Explain why you need each item
- Explain why you need this

3.3 Database Operations

TODO

- Present the calls that each endpoint requires(ge all articles, users, etc)

Chapter 4

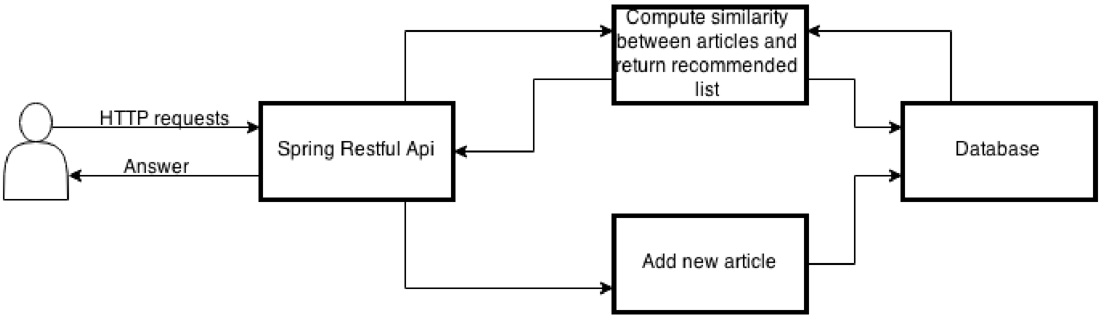
System Architecture and Workflow

In the following sections we are going to talk about the various architecture and technologies choices.

Also, we are going to present the workflows of the system.

4.1 Architecture

Figure 4.1: System architecture


Compute similarity between articles and return recommended list
HTTP requests
Answer
Spring Restful Api
Database
Add new article

TODO

- Update architecture
- Present Spring and why we chose it
- Present Java and why use chose it
- Present HBASE and why we chose it(not sure about this. Small presentation in database design. Can bring it from there)
- Present Stanford lemmatizer and why we chose it.

4.2 Workflow

- Add the workflow
- Add all the workflows(use cases) and explain them in depth

Chapter 5

Testing and Validation

In order to test and validate my solution I needed an article database. To have that, I filled my database with random values, so that I could check that I have configured it correctly and to have a solid base on which I could test my recommendation algorithms.

5.1 Testing

5.1.1 Basic Operations

TODO

- Add the test cases for the basic operations, unrelated to the next two sections

5.1.2 Related Articles Operations

TODO

- Add the test cases for related articles.

5.1.3 Recommended Articles Operations

TODO

- Add the test cases for recommended articles.

5.2 Evaluation

TODO

- Write why it is hard to evaluate the recommender system.
- Write the evaluation scenarios that do not depend on related or recommended.

5.2.1 Related Articles

5.2.1.1 Final Result

Because it was almost impossible to check if I got good and relevant results with a random database I had to take a database from a previous project of Adobe, which was used by one of their clients, fast company.

After I populated my database with their data, I started running tests on it by changing the importance of each attribute and printing the top 100 results

In order to check that I was giving a good result I chose an article and google searched its title on fast company's site. I then classified the outputs in 3 categories, by relevance and saved the data in an expected file.

Using the expected data I then binary searched for the best importance of each attribute. Because the expected data that I chose may have been determined by my own personality, I decided to test my recommendation system by using solr.

Using the best importance values I determined by using the expected file, I got my 10 most related articles in solr top 25 related.

In order to further confirm that my resulted articles are really related I built a simple web page in which users chose one of 4 degrees of relatedness and the data was saved on a server.

5.2.2 Recommended Articles

5.2.2.1 Final Result

In order to test that the system gives good recommendations we took the dataset of MovieLens which provided 100,000 ratings from 1000 users on 1700 movies. Using that data we then used our algorithms to predict what rating would a user give to a certain movie, already rated, and compared it to the actual given rating. By doing this we obtained a deviation from the removed rating of about 15%.

Chapter 6

Conclusions and Future Development

6.1 Conclusions

We proved in this thesis that we built a stand-alone application that can give good recommendations and can be easily integrated with any database.

We built a system that takes full advantage of all the existing data and gives the best recommendations for the presented problem.

Also, the importance of each considered attribute can be easily changed and thus we can easily make recommendations of articles made by a certain author, belong to a certain category, belong to the same collection, etc.

We can combine the recommended articles for a certain user with the related articles, in order to solve the cold start problem and to obtain better results.

6.2 Future Development

In order to further improve the recommendations we could use a neural network to predict what ratings a user would give to the unrated articles[3]. This way, we could better predict the users with similar interests. TODO

- Add more about the neural network.

Bibliography

[1] Drakoulis Martakos Charalampos Vassiliou, Dimitris Stamoulis and Sortiris Athanassopoulo. A recommender system framework combining neural networks and collaborative filtering. http://www.wseas.us/e-library/conferences/2006hangzhou/papers/531-656.pdf, April 2006.

[2] Brent Smith Greg Linden and Jeremy York. Amazon.com recommendations item-to-item collaborative filtering. http://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf, February 2003.

[3] Yehuda Koren Robert M. Bell and Chris Volinsky. The bellkor 2008 solution to the netflix prize. http://www2.research.att.com/~volinsky/netflix/Bellkor2008.pdf, June 2008.

[4] Michael R. Smith and Tony Martinez. A hybrid latent variable neural network model for item recommendation. http://arxiv.org/pdf/1406.2235.pdf, June 2014.