Developing a Selection Strategy for Hybrid Recommendations

In Partial Fulfillment of the Requirements for the Degree of
Bachelors of Technology in Computer Science & Engineering
pursued under the guidance of

Mr. Prabhakar Sarma Neog
(Assistant Professor, Dept of CSE, NIT Silchar)

Members in Group:
Vikas Singh (11-1-5-097)
Shival Gupta (11-1-5-002)
Shishuvendra Tiwari (11-1-5-004)
Pramod Prasad (11-1-5-092)
Pabitra Pegu (11-1-5-091)
Manjil Brahma (11-1-5-065)

Class of 2015
Department of Computer Science and Engineering
National Institute of Technology Silchar – 788010

Table of Contents

Certificate

This is to certify that the Project Report entitled, “Developing a Selection Strategy for Hybrid Recommendations” submitted by **Shival Gupta, Vikas Singh, Shishuvendra Tiwari, Pramod Prasad, Pabitra Pegu, Manjil Brahma** to National Institute of Technology, Silchar, India, is a record of bonafide Project work carried out by them under my supervision and guidance and is worthy of consideration for the award of the degree of Bachelor of Technology in Computer Science and Engineering of the Institute.

Mr Prabhakar Sarma Neog
Assistant Professor
Department of Computer Science and Engineering
National Institute of Technology Silchar

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Shival Gupta
11-1-5-002

Vikas Singh
11-1-5-097

Shishuvendra Tiwari
11-1-5-004

Pramod Prasad
11-1-5-092

Pabitra Pegu
11-1-5-091

Manjil Brahma
11-1-5-065

Abstract

Recommender systems are being widely applied in many fields, such as e-commerce etc, to provide products, services and information to potential customers. Recommender systems represent user preferences for the purpose of suggesting items to purchase or examine.

They have become fundamental applications in electronic commerce and information access, providing suggestions that effectively prune large information spaces so that users are directed toward those items that best meet their needs and preferences. A variety of techniques havebeen proposed for performing recommendation, including content-based, collaborative, knowledge-based and other techniques.

To improve performance, these methods have sometimes been combined in hybrid recommenders. This project experiments among three hybridization strategies for combining content-based and collaborative filtering and finding the most suitable strategy for developing a hybrid recommender system for MovieLens Dataset. The other important aspect of the project is the application of Apache Lucene for implementing Content-based and Apache Mahout for Collaborative Filtering.

The use of Apache products make the approach to be industrially feasible.

1. Introduction and Literature Survey

1.1 Overview of Recommender Systems

Recommender systems or recommendation systems (sometimes replacing "system" with a synonym such as platform or engine) are a subclass of information filtering system that seek to predict the 'rating' or 'preference' that user would give to an item.

Recommender systems have become extremely common in recent years, and are applied in a variety of applications. The most popular ones are probably movies, music, news, books, research articles, search queries, social tags, and products in general. However, there are also recommender systems for experts, jokes, restaurants, financial services, life insurance, persons (online dating), and Twitter followers.

Recommender systems typically produce a list of recommendations in one of two ways - through collaborative or content-based filtering. Collaborative filtering approaches building a model from a user's past behavior (items previously purchased or selected and/or numerical ratings given to those items) as well as similar decisions made by other users; then use that model to predict items (or ratings for items) that the user may have an interest in. Content-based filtering approaches utilize a series of discrete characteristics of an item in order to recommend additional items with similar properties. These approaches are often combined (see Hybrid Recommender Systems).

The differences between collaborative and content-based filtering can be demonstrated by comparing two popular music recommender systems - Last.fm and Pandora Radio.

Pandora uses the properties of a song or artist (a subset of the 400 attributes provided by the Music Genome Project) in order to seed a "station" that plays music with similar properties. User feedback is used to refine the station's results, deemphasizing certain attributes when a user "dislikes" a particular song and emphasizing other attributes when a user "likes" a song. This is an example of a content-based approach.

Last.fm creates a "station" of recommended songs by observing what bands and individual tracks that the user has listened to on a regular basis and comparing those against the listening behavior of other users. Last.fm will play tracks that do not appear in the user's library, but are often played by other users with similar interests. As this approach leverages the behavior of users, it is an example of a collaborative filtering technique.
Each type of system has its own strengths and weaknesses. In the above example, Last.fm requires a large amount of information on a user in order to make accurate recommendations. This is an example of the cold start problem, and is common in collaborative filtering systems. While Pandora needs very little information to get started, it is far more limited in scope (for example, it can only make recommendations that are similar to the original seed).

Recommender systems are a useful alternative to search algorithms since they help users discover items they might not have found by themselves. Interestingly enough, recommender systems are often implemented using search engines indexing non-traditional data.

Montaner provides the first overview of recommender systems, from an intelligent agents perspective. Adomavicius provides a new overview of recommender systems. Herlocker provides an additional overview of evaluation techniques for recommender systems, and Beel et al. discuss the problems of offline evaluations. They also provide a literature survey on research paper recommender systems.

Recommender system is an active research area in the data mining and machine learning areas.

1.2 Approaches of concept

1.2.1 Content based Filtering

Another common approach when designing recommender systems is content-based

filtering. Content-based filtering methods are based on a description of the item and a profile of the user's preference. In a content-based recommender system, keywords are used to describe the items; beside, a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past (or is examining in the present). In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended. This approach has its roots in information retrieval and information filtering research.

To abstract the features of the items in the system, an item presentation algorithm is applied. A widely used algorithm is the tf–idf representation (also called vector space representation).

To create user profile, the system mostly focuses on two types of information: 1. A model of the user's preference. 2. A history of the user's interaction with the recommender system.

Basically, these methods use an item profile (i.e. a set of discrete attributes and features) characterizing the item within the system. The system creates a content-based profile of users based on a weighted vector of item features. The weights denote the importance of each feature to the user and can be computed from individually rated content vectors using a variety of techniques. Simple approaches use the average values of the rated item vector while other sophisticated methods use machine learning techniques such as Bayesian Classifiers, cluster analysis, decision trees, and artificial neural networks in order to estimate the probability that the user is going to like the item.

Direct feedback from a user, usually in the form of a like or dislike button, can be used to assign higher or lower weights on the importance of certain attributes (using Rocchio Classification or other similar techniques).

A key issue with content-based filtering is whether the system is able to learn user preferences from user's actions regarding one content source and use them across other

content types. When the system is limited to recommending content of the same type as the user is already using, the value from the recommendation system is significantly less than when other content types from other services can be recommended. For example, recommending news articles based on browsing of news is useful, but it's much more useful when music, videos, products, discussions etc. from different services can be recommended based on news browsing.

As previously detailed, Pandora Radio is a popular example of a content-based recommender system that plays music with similar characteristics to that of a song provided by the user as an initial seed. There are also a large number of content-based recommender systems aimed at providing movie recommendations, a few such examples include Rotten Tomatoes, Internet Movie Database, Jinni, Rovi Corporation, Jaman and See This Next.

1.2.2 Collaborative Filtering

Collaborative filtering (CF) is a technique used by some recommender systems. Collaborative filtering has two senses, a narrow one and a more general one. 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.

In the newer, narrower sense, collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on

an issue, A is more likely to have B's opinion on a different issue x than to have the opinion on x of a person chosen randomly. For example, a collaborative filtering recommendation system for television tastes could make predictions about which television show a user should like given a partial list of that user's tastes (likes or dislikes). Note that these predictions are specific to the user, but use information gleaned from many users. This differs from the simpler approach of giving an average (non-specific) score for each item of interest, for example based on its number of votes.

1.2.3 Types of Collaborative Filtering (CF)

Memory Based

This mechanism uses user rating data to compute similarity between users or items. This is used for making recommendations. This was the earlier mechanism and is used in many commercial systems. It is easy to implement and is effective. Typical examples of this mechanism are neighbourhood based CF and item-based/user-based top-N recommendations.

Model Based

Models are developed using data mining, machine learning algorithms to find patterns based on training data. These are used to make predictions for real data. There are many model-based CF algorithms. These include Bayesian networks, clustering models, latent semantic models such as singular value decomposition, probabilistic latent semantic analysis, Multiple Multiplicative Factor, Latent Dirichlet allocation and markov decision process based models.

This approach has a more holistic goal to uncover latent factors that explain observed ratings. Most of the models are based on creating a classification or clustering technique to identify the user based on the test set. The number of the parameters can be reduced based on types of principal component analysis.

There are several advantages with this paradigm. It handles the sparsity better than memory based ones. This helps with scalability with large data sets. It improves the prediction performance. It gives an intuitive rationale for the recommendations.

The disadvantages with this approach are in the expensive model building. One needs to

have a tradeoff between prediction performance and scalability. One can lose useful information due to reduction models. A number of models have difficulty explaining the predictions.

2. Hybrid Recommender Systems

2.1 Introduction

Recent research has demonstrated that a hybrid approach, combining collaborative filtering and content-based filtering could be more effective in some cases. Hybrid approaches can be implemented in several ways: by making content-based and collaborative-based predictions separately and then combining them; by adding content-based capabilities to a collaborative-based approach (and vice versa); or by unifying the approaches into one model (see[10] for a complete review of recommender systems). Several studies empirically compare the performance of the hybrid with the pure collaborative and content-based methods and demonstrate that the hybrid methods can provide more accurate recommendations than pure approaches. These methods can also be used to overcome some of the common problems in recommender systems such as cold start and the sparsity problem.

Netflix is a good example of hybrid systems. They make recommendations by comparing the watching and searching habits of similar users (i.e. collaborative filtering) as well as by offering movies that share characteristics with films that a user has rated highly (content-based filtering).

A variety of techniques have been proposed as the basis for recommender systems: collaborative, content-based, knowledge-based, and demographic techniques. Each of these techniques has known shortcomings, such as the well known cold-start problem for collaborative and content-based systems (what to do with new users with few ratings) and the knowledge engineering bottleneck[26] in knowledge-based approaches. A hybrid recommender system is one that combines multiple techniques together to achieve some synergy between them.

Collaborative: The system generates recommendations using only information about rating profiles for different users. Collaborative systems locate peer users with a rating history similar to the current user and generate recommendations using this neighborhood.

Content-based: The system generates recommendations from two sources: the features associated with products and the ratings that a user has given them. 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.

Demographic: A demographic recommender provides recommendations based on a demographic profile of the user. Recommended products can be produced for different demographic niches, by combining the ratings of users in those niches.

Knowledge-based: A knowledge-based recommender suggests products based on inferences about a user’s needs and preferences. This knowledge will sometimes contain explicit functional knowledge about how certain product features meet user needs.[27][28]

The term hybrid recommender system is used here to describe any recommender system that combines multiple recommendation techniques together to produce its output. There is no reason why several different techniques of the same type could not be hybridized, for example, two different content-based recommenders could work together, and a number of projects have investigated this type of hybrid: NewsDude, which uses both naive Bayes and kNN classifiers in its news recommendations is just one example.[27]

2.2 Hybridization techniques

2.2.1 Weighted

A weighted hybrid recommender is one in which the score of a recommended item is computed from the results of all of the available recommendation techniques present in the system. For example, the simplest combined hybrid would be a linear combination of recommendation scores. The P-Tango system (Claypool et al. 1999) uses such a hybrid. It initially gives collaborative and content-based recommenders equal weight, but gradually adjusts the weighting as predictions about user ratings are confirmed or disconfirmed. Pazzani’s combination hybrid does not use numeric scores, but rather treats the output of each recommender (collaborative, content-based and demographic) as a set of votes, which are then combined in a consensus scheme (Pazzani, 1999). The benefit of a weighted hybrid is that all of the system’s capabilities are brought to bear on the recommendation process in a straightforward way and it is easy to perform post-hoc credit assignment and adjust the hybrid accordingly. However, the implicit assumption in this technique is that the relative value of the different techniques is more or less uniform across the space of possible items. From the discussion above, we know that this is not always so: a collaborative recommender will be weaker for those items with a small number of raters.

2.2.2 Switching

A switching hybrid builds in item-level sensitivity to the hybridization strategy: the system uses some criterion to switch between recommendation techniques. The DailyLearner system uses a content/collaborative hybrid in which a content-based recommendation method is employed first. If the content-based system cannot make a recommendation with sufficient confidence, then a collaborative recommendation is attempted.3 This switching hybrid does not completely avoid the ramp-up problem, since both the collaborative and the content-based systems have the "new user" problem. However, DailyLearner's content-based technique is nearest-neighbor, which does not require a large number of examples for accurate classification. What the collaborative technique provides in a switching hybrid is the ability to cross genres, to come up with recommendations that are not close in a semantic way to the items previous rated highly, but are still relevant. For example, in the case of DailyLearner, a user who is interested in the Microsoft anti-trust trial might also be interested in the AOL/Time Warner merger. Content matching would not be likely to recommend the merger stories, but other users with an interest in corporate power in the high-tech industry may be rating both sets of stories highly, enabling the system to make the recommendation collaboratively. DailyLearner's hybrid has a "fallback" character – the short-term model is always used first and the other technique only comes into play when that technique fails. Tran & Cohen (1999) proposed a more straightforward switching hybrid. In their system, the agreement between a user's past ratings and the recommendations of each technique are used to select the technique to employ for the next recommendation. Switching hybrids introduce additional complexity into the recommendation process since the switching criteria must be determined, and this introduces another level of parameterization. However, the benefit is that the system can be sensitive to the strengths and weaknesses of its constituent recommenders.

2.1.3 Mixed

Where it is practical to make large number of recommendations simultaneously, it may be possible to use a "mixed" hybrid, where recommendations from more than one technique are presented together. The PTV system (Smyth and Cotter 2000) uses this approach to assemble a recommended program of television viewing. It uses content-based techniques

based on textual descriptions of TV shows and collaborative information about the preferences of other users. Recommendations from the two techniques are combined together in the final suggested program. The mixed hybrid avoids the “new item” start-up problem: the content-based component can be relied on to recommend new shows on the basis of their descriptions even if they have not been rated by anyone. It does not get around the “new user” start-up problem, since both the content and collaborative methods need some data about user preferences to get off the ground, but if such a system is integrated into a digital television, it can track what shows are watched (and for how long) and build its profiles accordingly. Like the fallback hybrid, this technique has the desirable “niche-finding” property in that it can bring in new items that a strict focus on content would eliminate. The PTV case is somewhat unusual because it is using recommendation to assemble a composite entity, the viewing schedule. Because many recommendations are needed to fill out such a schedule, it can afford to use suggestions from as many sources as possible. Where conflicts occur, some type of arbitration between methods is required – in PTV, content-based recommendation take precedence over collaborative responses. Other implementations of the mixed hybrid, ProfBuilder (Wasfi, 1999) and PickAFlick (Burke et al. 1997; Burke, 2000), present multiple recommendation sources side-by-side. Usually, recommendation requires ranking of items or selection of a single best recommendation, at which point some kind of combination technique must be employed.

3. Approach to Experiment

3.1 Introduction

The approach to experiment is as follows. It is intended to develop a theoretical approach using which we can develop a successful and efficient hybridization strategy for a hybrid recommender system. The components of our approach have two main facets, the collaborative model and the content model. By combining these two models in different capacities, we will try to unearth the best possible way in which such two systems can be combined.

Both models are sufficient independently for simpler systems but their accuracy is faltered with many factors. Using an hybrid approach helps cut down the inaccuracies and makes the process much more scientific and robust. The primary objective of the project is to find the best mechanism in which collaborative and content based systems are used in harmony.

3.2 Content Based Recommendation algorithm

In this approach a content-based movie recommendation method is used. We consider feature sets, such as actor, director, keyword etc. that describe a movie. For each feature in a feature set, based on the user's past viewed movies and the user's rating for each movie, we compute a feature weight. Each feature weight is calculated separately for each user. The features extracted from this movie are important, and their weights will be assigned accordingly. When a movie needs to be rated for a user, based on the features of the movie, we produce a different rating for each feature set and compare them. We also produce combined ratings which take into account all different feature sets

Following are the steps under this algorithm:

A. Feature Based Weight Calculation Method

The weight of feature j in feature set k for user u is calculated as:

$$w_k(u,j) = \frac{1}{\left|I_u^{train}\right|} \sum_{i \in I_u^{train}} x_{k,u}(i,j) r(u,i)$$

k represents the type of the feature set, k ∈ {actor, genre, director}. r(u,i) ∈ R is the implicit rating of user u for item i and $x_{k,u}$ (i , j) ∈ {0,1} jth feature of item i. I^{train}_u is the set of movies

watched by user u in the raining period. If movie i has feature j then $x_{k,u}(i,j)$ will be 1 and user rating for movie i will contribute to the sum.

B. Rating Prediction Method

The following equation is used for calculating the predicted rating specific to particular feature set.

$$r_k(u,i) = \sum_{j \in D_{k,i}} w_k(u,j)$$

$r_k(u,i)$ represents the rating that user u gives the movie i according to the k feature set. $D_{k,i}$ is the features that appear in movie i from feature set k.

C. Combining the rating of different feature sets.

$$r_k^N(u,i) = (r_k(u,i) - mR_{u,k}) / (MR_{u,k} - mR_{u,k})$$

In order to determine the total effect of all the features, the rating of each feature which is calculated needs to be normalized. In this work min-max normalization method is used it is conducted as follows: $mR_{u,k}$ indicates the minimum predicted rating of the user u calculated according to feature set k in the training set and $MR_{u,k}$ indicates the maximum predicted rating of the user u calculated according to feature set k in the training set.

Ratings are brought together with the use of two different methods. Actor, genre, director, time zone, channel, keyword, release year normalized ratings are summed in the first method.

$$r_{\text{sum}}^N(u,i) = \sum_{k=1}^{K} r_k^N(u,i)$$

\

3.3 Collaborative filtering algorithm

In our project we will be using item based recommendation algorithm for producing predictions to users. The item based approach looks into the set of item, the target user has rated and computes how similar they are to the target item i and then selects k most similar items $\{i_1, i_2, ..., i_k\}$. At the same time their corresponding similarities $\{s_{i1}, s_{i2}, ..., s_{ik}\}$are also computed. Once the most similar items are found the prediction is computed by taking the weighted average of the target users ratings on these similar items.

A. Adjusted Cosine Similarity

The similarity between items i and j is given by

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_u)(R_{u,j} - \bar{R}_u)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_u)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_u)^2}}.$$

Here $\bar{R}_u$ is the average of the u^{th} user's rating and $R_{u,i}$ is the rating given by uth user to the ith item.

B. Prediction Computation using Weighted Sum

As the name implies this method computes the prediction on an item i for a user u by computing the sum of the ratings given by the user on the items similar to i. Each rating is weighted by the corresponding similarity $s_{i,j}$ between items i and j. Using the following notation we can denote the prediction $P_{u,i}$ as

$$P_{u,i} = \frac{\sum_{\text{all similar items, N}}(s_{i,N} * R_{u,N})}{\sum_{\text{all similar items, N}}(|s_{i,N}|)}$$

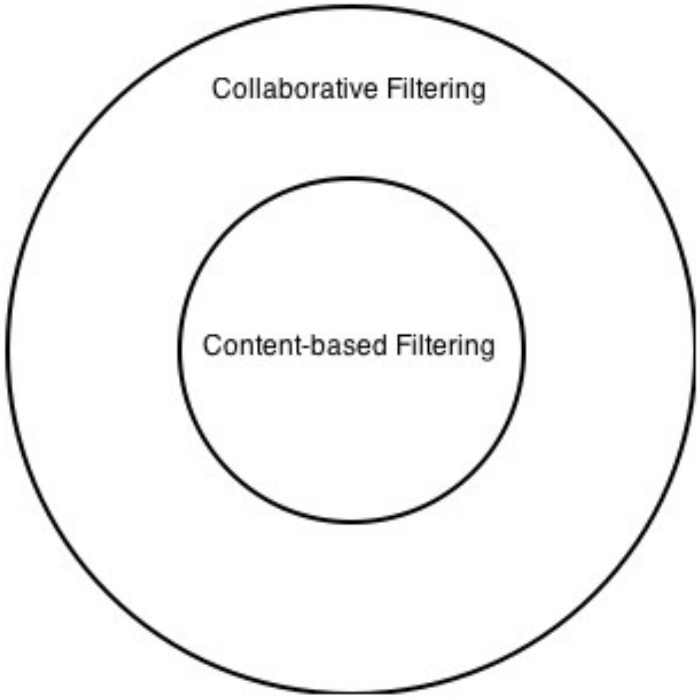
3.4 Approaches to hybrid recommendation

We are using one weighted and two switching approaches for combining the content based and collaborative filtering to develop a hybrid recommender system. The three approaches are as follows.

3.4.1 Content-based filtering centric hybridization approach. (Switching approach)

In this approach the procedure is followed as given below:

- First we apply content based filtering to generate recommendations.
- For the user-item pairs for which the rating could not be predicted using content based ideology, we apply collaborative strategy for exclusively those user-item pairs.

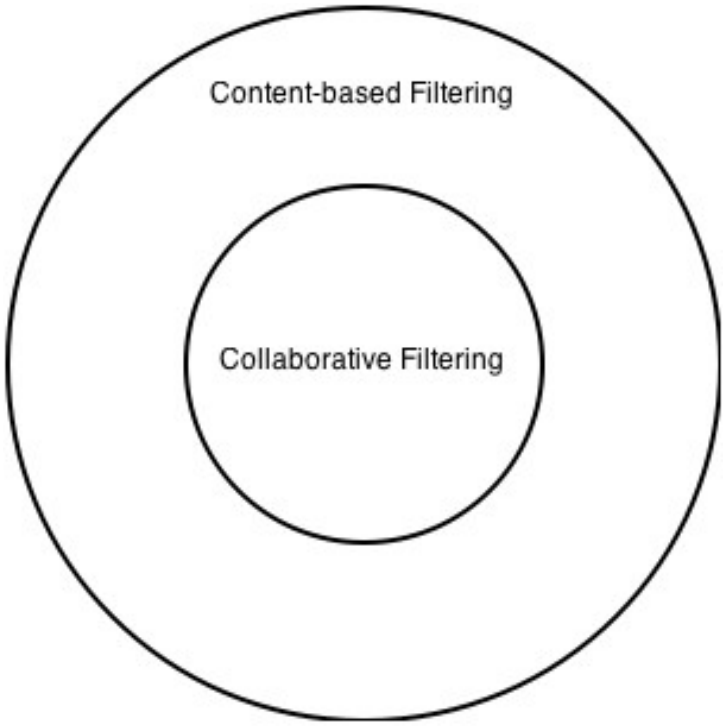


The abovementioned venn diagram symbolically displays the approach.

3.4.2 Collaborative filtering centric hybridization approach.

In this approach the procedure is given as follows:

- First we apply collaborative filtering to generate recommendations.
- For the user-item pairs for which the rating could not be predicted using collaborative ideology, we apply content-based strategy for exclusively those user-item pairs.

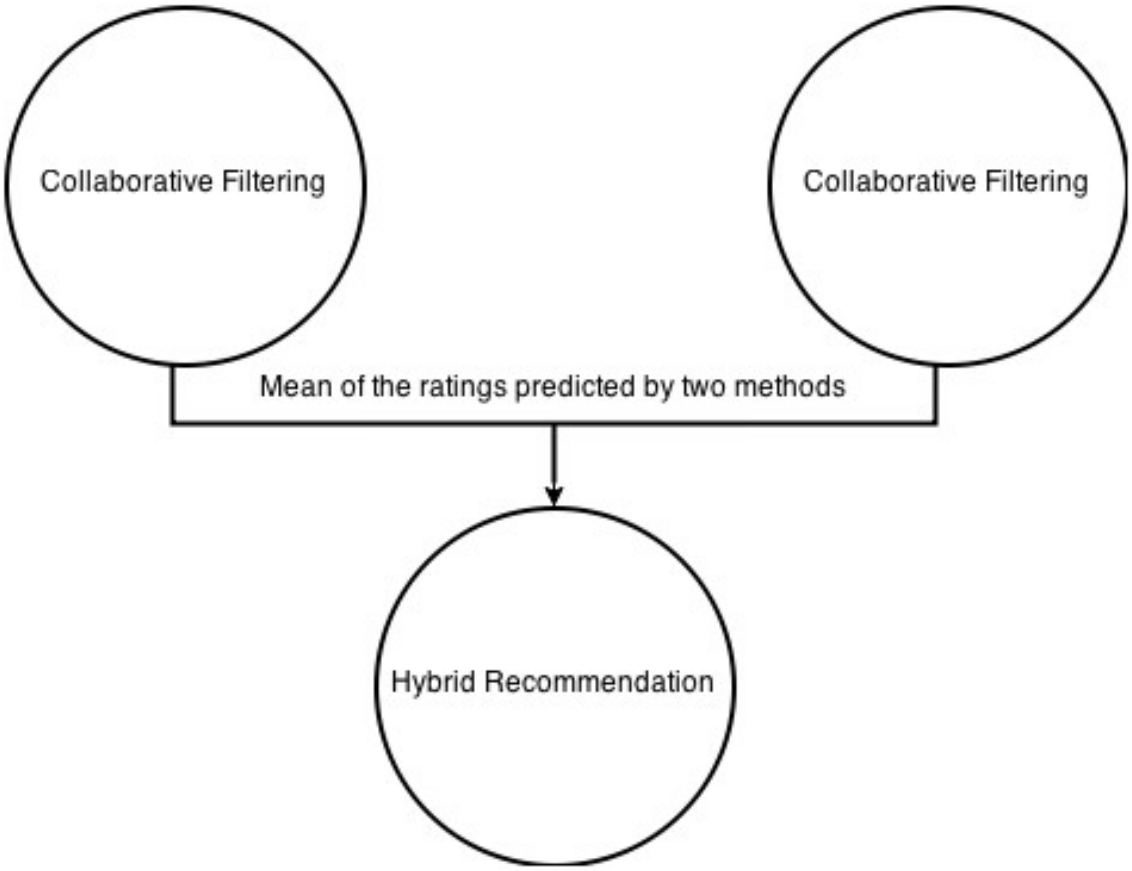


The abovementioned venn diagram depicts how the given strategy would appear symbolically.

3.4.3 Content Based and Collaborative bicentric hybridization approach

The procedure for this approach is as follows:

- In this approach we first take the content-based strategy and calculate recommendations.
- Then a collaborative approach is employed to calculate recommendations from a different perspective.
- Then the statistical mean of two recommendation values is taken and is considered to be the end result of the process.



The abovementioned flow diagram depicts this approach in theory.

3.5 Objective of the Experiment

The objective is to find the most promising strategy from the above three choices that generate qualitatively the best recommendations and to criticize and weigh each choice against each other to declare a winner for commercial use.

4. Tools in Use

4.1 Apache Mahout

Mahout is an open source machine learning library from Apache. The algorithms it implements fall under the broad umbrella of machine learning or collective intelligence. This can mean many things, but at the moment for Mahout it means primarily recommender engines (collaborative filtering), clustering, and classification.

Mahout aims to be the machine learning tool of choice when the collection of data to be processed is very large, perhaps far too large for a single machine. In its current incarnation, these scalable machine learning implementations in Mahout are written in Java, and some portions are built upon Apache's Hadoop distributed computation project. It's a Java library. It doesn't provide a user interface, a prepackaged server, or an installer. It's a framework of tools intended to be used and adapted by developers.

Recommender engines are the most immediately recognizable machine learning technique in use today. You'll have seen services or sites that attempt to recommend books or movies or articles based on your past actions. They try to infer tastes and preferences and identify unknown items that are of interest: Amazon.com is perhaps the most famous e-commerce site to deploy recommendations. Based on purchases and site activity, Amazon recommends books and other items likely to be of interest. Netflix similarly recommends DVDs that may be of interest, and famously offered a $1,000,000 prize to researchers who could improve the quality of their recommendations.

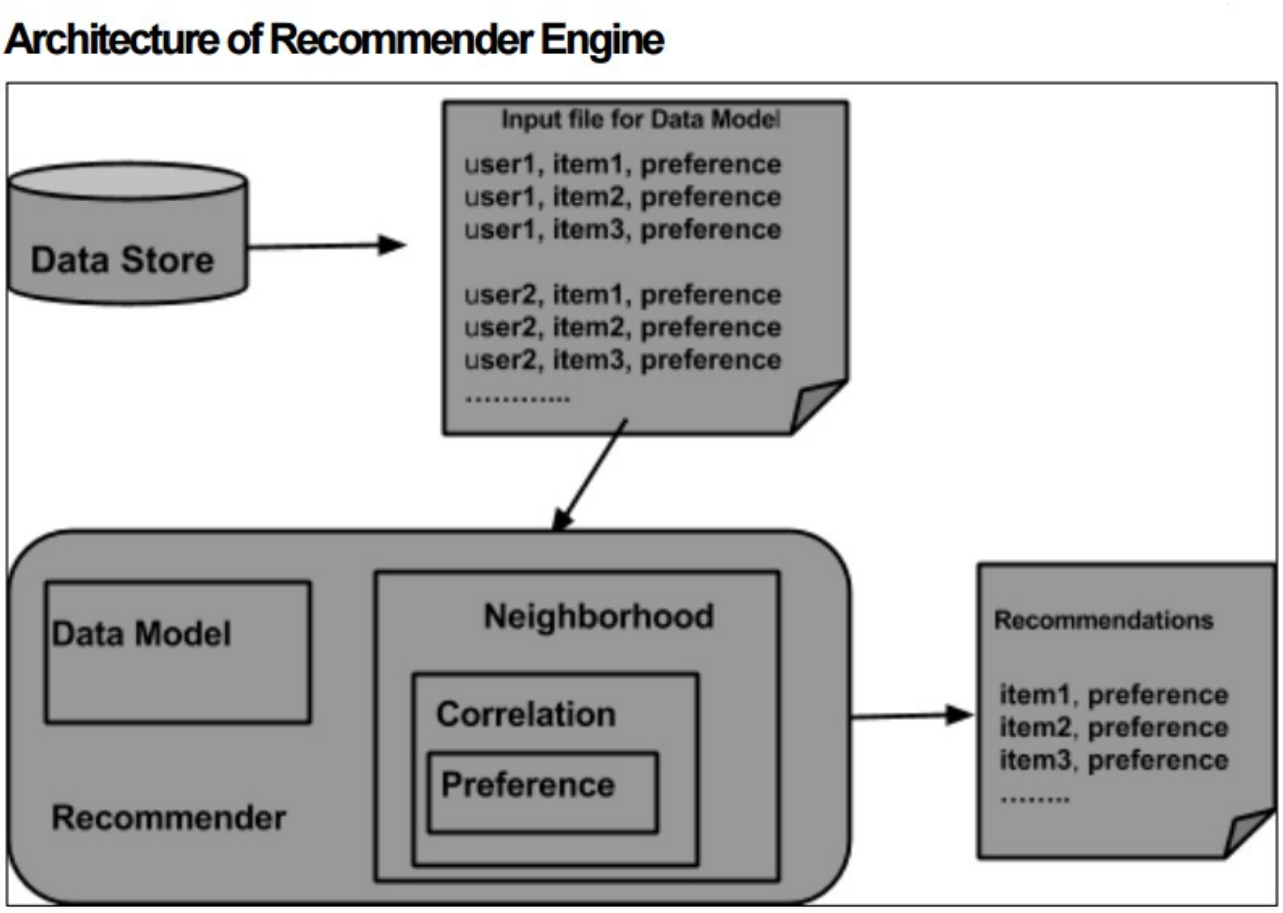
Dating sites like Líbímseti can even recommend people to people. Social networking sites like Facebook use variants on recommender techniques to identify people most likely to be as-yet-unconnected friends. As Amazon and others have demonstrated, recommenders can have concrete commercial value by enabling smart cross-selling opportunities.

The components provided by Mahout to build a recommender engine are as follows:

- DataModel
- UserSimilarity
- ItemSimilarity

- UserNeighborhood
- Recommender

From the data store, the data model is prepared and is passed as an input to the recommender engine. The Recommender engine generates the recommendations for a particular user. Given below is the architecture of recommender engine.

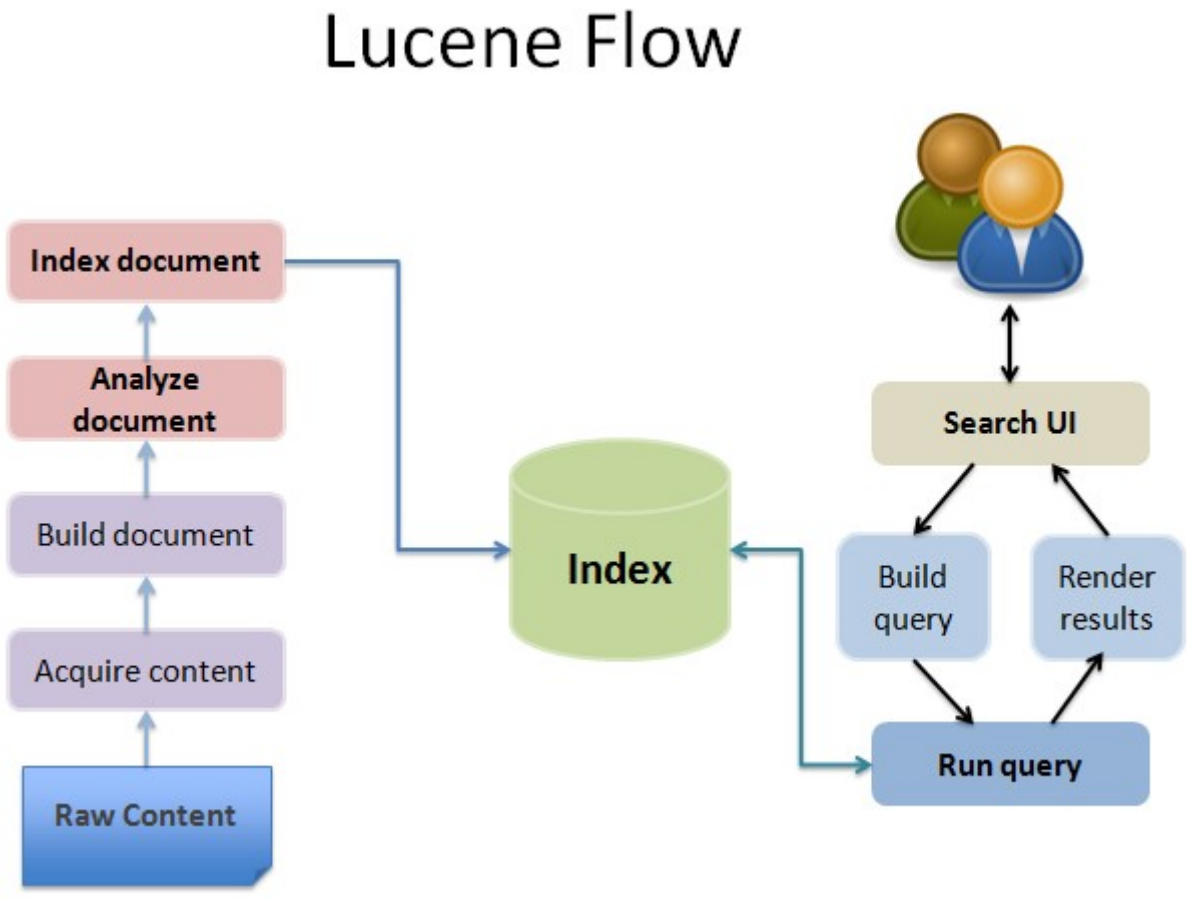


4.2 Apache Lucene

Lucene is a high performance, scalable Information Retrieval (IR) library. It lets you add indexing and searching capabilities to your applications. Lucene is a mature, free, open-source project implemented in Java; it's a member of the popular Apache Jakarta family of projects, licensed under the liberal Apache Software License. As such, Lucene is currently, and has been for a few years, the most popular free Java IR library.

As you'll soon discover, Lucene provides a simple yet powerful core API that requires

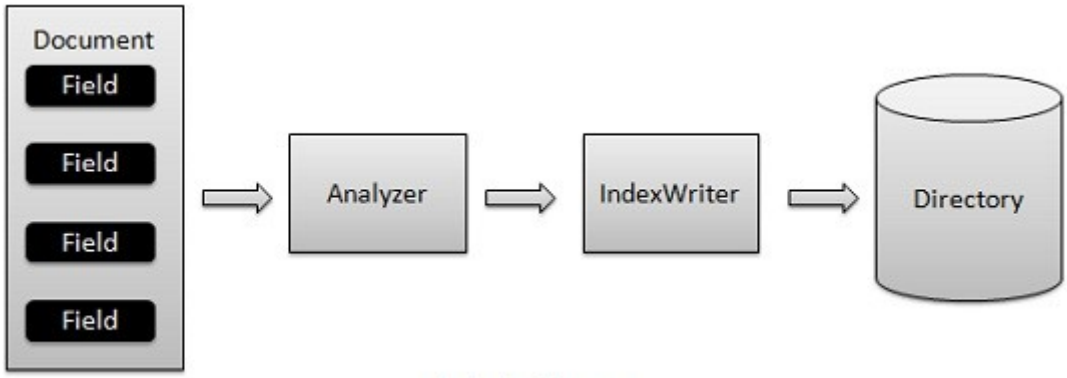
minimal understanding of full-text indexing and searching. You need to learn about only a handful of its classes in order to start integrating Lucene into an application. Because Lucene is a Java library, it doesn't make assumptions about what it indexes and searches, which gives it an advantage over a number of other search applications. People new to Lucene often mistake it for a ready-to-use application like a file-search program, a web crawler, or a web site search engine. That isn't what Lucene is: Lucene is a software library, a toolkit if you will, not a full-featured search application. It concerns itself with text indexing and searching, and it does those things very well. Lucene lets your application deal with business rules specific to its problem domain while hiding the complexity of indexing and searching implementation behind a simple-to-use API.



4.2.1 Indexing in Lucene

Indexing process is one of the core functionality provided by Lucene. Following diagram illustrates the indexing process and use of classes. IndexWriter is the most important and

core component of the indexing process.



Indexing Process

We add *Document*(s) containing *Field*(s) to *IndexWriter* which analyzes the *Document*(s) using the *Analyzer* and then creates/open/edit indexes as required and store/update them in a *Directory*. IndexWriter is used to update or create indexes. It is not used to read indexes.

Following is the list of commonly used classes during Indexing process:

- IndexWriter
- Directory
- Analyzer
- Document
- Field

4.2.2 Creating a document

- Create a method to get a lucene document from a text file.
- Create various types of fields which are key value pairs containing keys as names and values as contents to be indexed.
- Set field to be analyzed or not. In our case, only contents is to be analyzed as it can contain data such as a, am, are, an etc. which are not required in search operations.
- Add the newly created fields to the document object and return it to the caller method.
-

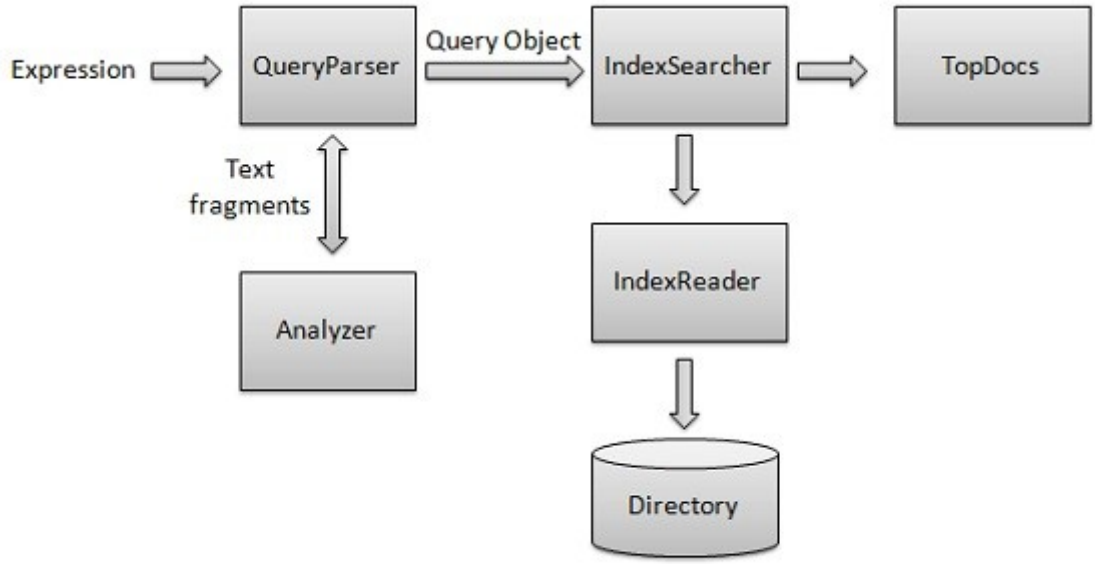
4.2.3 Searching in Lucene

Searching process is again one of the core functionality provided by Lucene. It's flow is similar to that of indexing process. Basic search of lucene can be made using following classes which can also be termed as foundation classes for all search related operations.

Following are the classes commonly used in searching:

- IndexSearcher
- Term
- Query
- TermQuery
- TopDocs

The following flowchart shows a typical searching process in Apache Lucene.



Searching Process

5. Experiment

5.1 Data Set for the Experiment

This dataset is an extension of MovieLens10M dataset, published by GroupLens research group.

It links the movies of MovieLens dataset with their corresponding web pages at Internet Movie Database (IMDb) and Rotten Tomatoes movie review systems. From the original dataset, only those users with both rating and tagging information have been mantained. The dataset is released in the framework of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011) at the 5th ACM Conference on Recommender Systems (RecSys 2011)

The salient features of the dataset are as follows.

2113	users
10197	Movies
20	Movie genres
20809	Movie genre assignments
4060	Directors
95321	Actors
72	Countries
10197	Country assignments
855598	ratings

We divided the dataset into two parts. One is declared as training and the other is kept reserved for testing. The training data set contains 90% of the ratings while the remaining 10% are reserved for testing. The distribution of the dataset into training and testing is randomly selected.

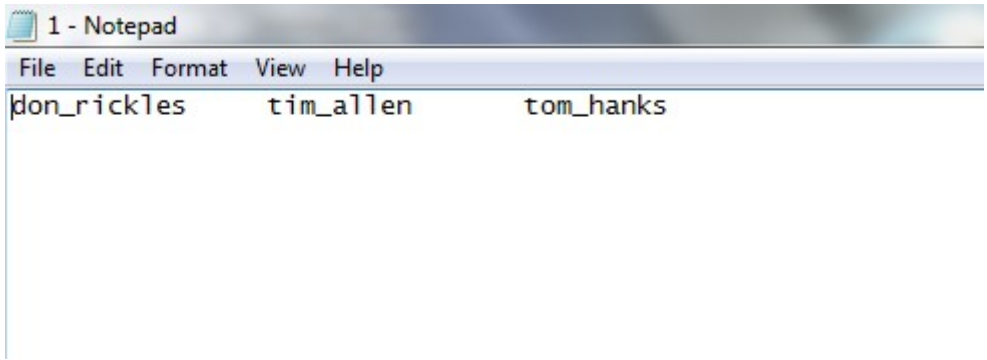
5.2 Acclimatizing the raw dataset into project specific format

The following steps were taken to acclimatize the raw dataset into a format which we can use for our calculations:

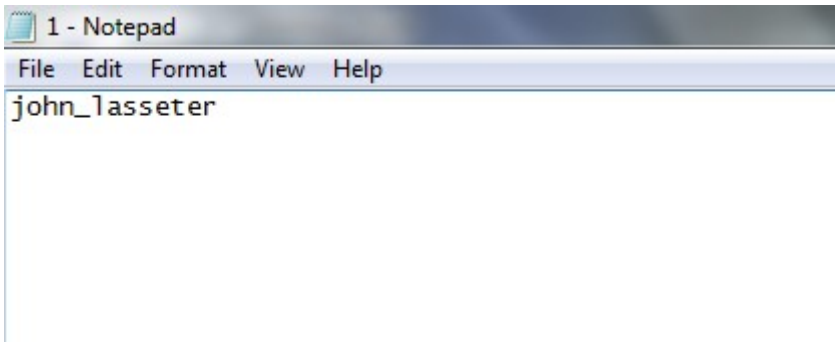
- For implementing the content-based filtering using different feature sets we

considered four feature sets from the data set, *viz*, actor, director, genre and country (in which the movie is based).

- For each feature set a movie profile is created that contains the attributes of that feature set that are present in the movie. The movie profile file is named by the movie ID of that particular movie.
    - Actor feature set: The attribute of the actor feature set are the actor ids of all the actors that are available in the dataset. For the actor profile of each movie, the profile contains the ids of three most important actors/actresses that have acted in the movie. The ids are in alphanumeric form.

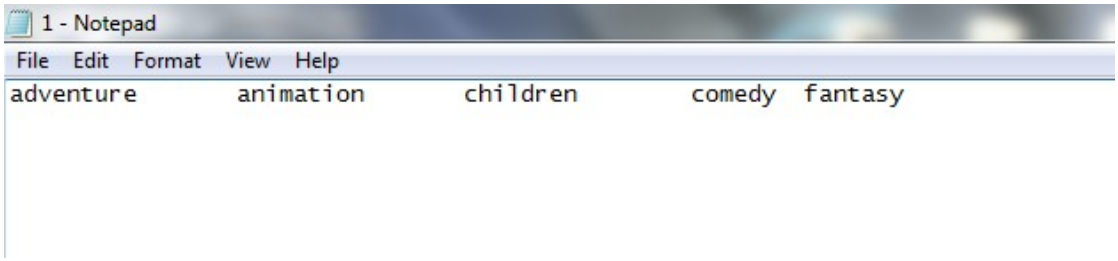


    - Director feature set; The attribute of the director feature set are the diretor ids of all the directors that are available in the dataset. For the director profile of each movie, the profile contains the id of the director that directed the movie. The ids are in alphanumeric form.

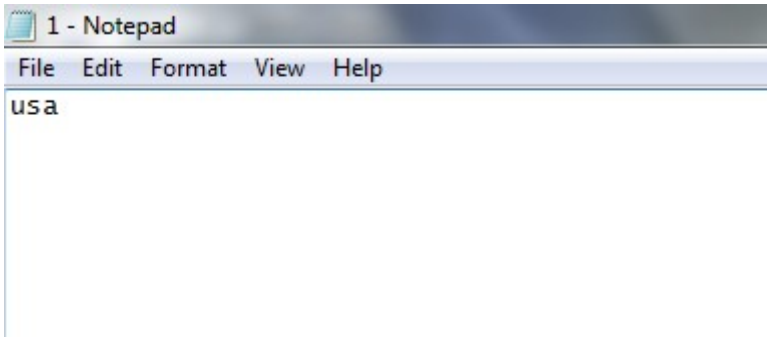


    - Genre feature set: The attribute of the genre feature set are all the genres

that a movie can belong to and are available in the dataset. Their are 20 movie genres.



- Country feature set: The attribute of the country feature set are the name of the countries that are available in the dataset. Their are 72 country names in the data set.

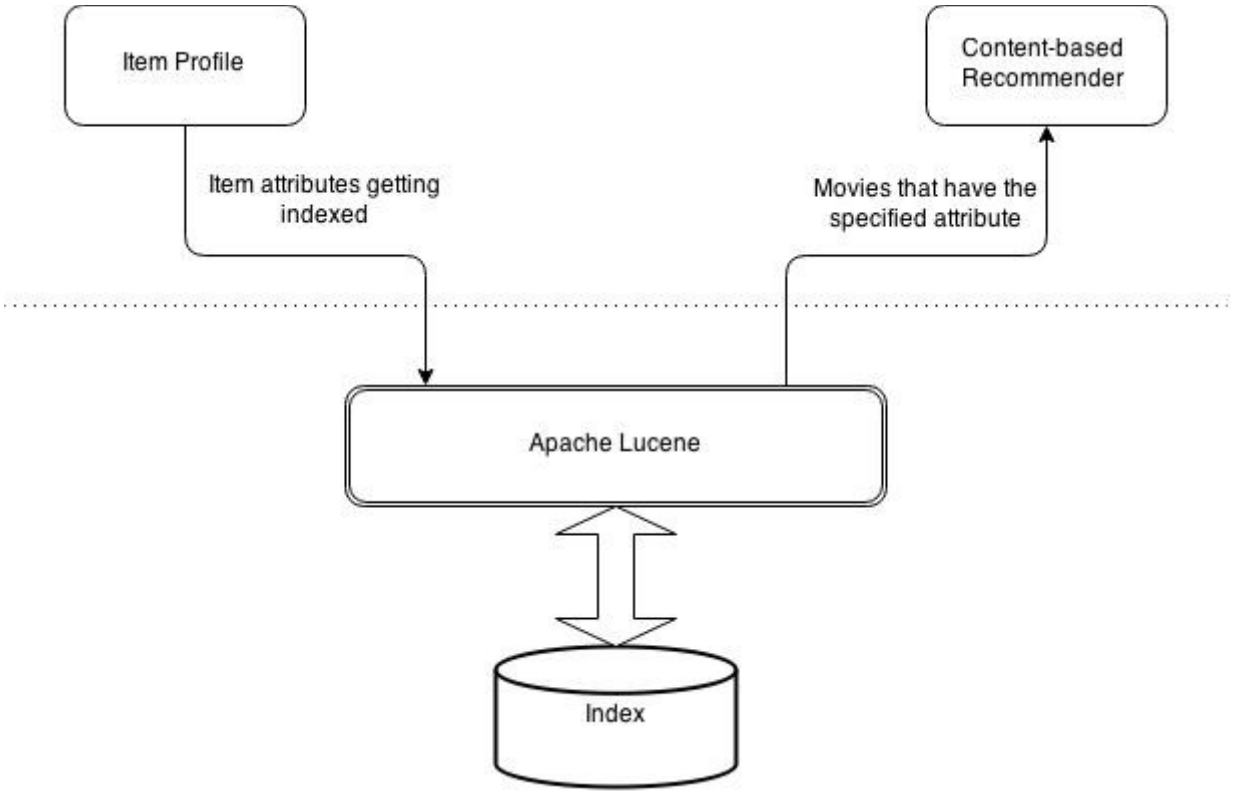


5.3 Implementing content based filtering using Apache Lucene

The following procedure was used to implement the content based filtering strategy using Apache Lucene

- The movie profiles files corresponding to various feature sets are indexed using Apache Lucene.
- To calculate the weight of a feature in the feature set for a user Apache Lucene is used for retrieving all those movies whose profile description contains that particular attribute (feature). Depending on the retrieved information about movies and among those the movies that have been rated by the concerned user the weight is calculated.

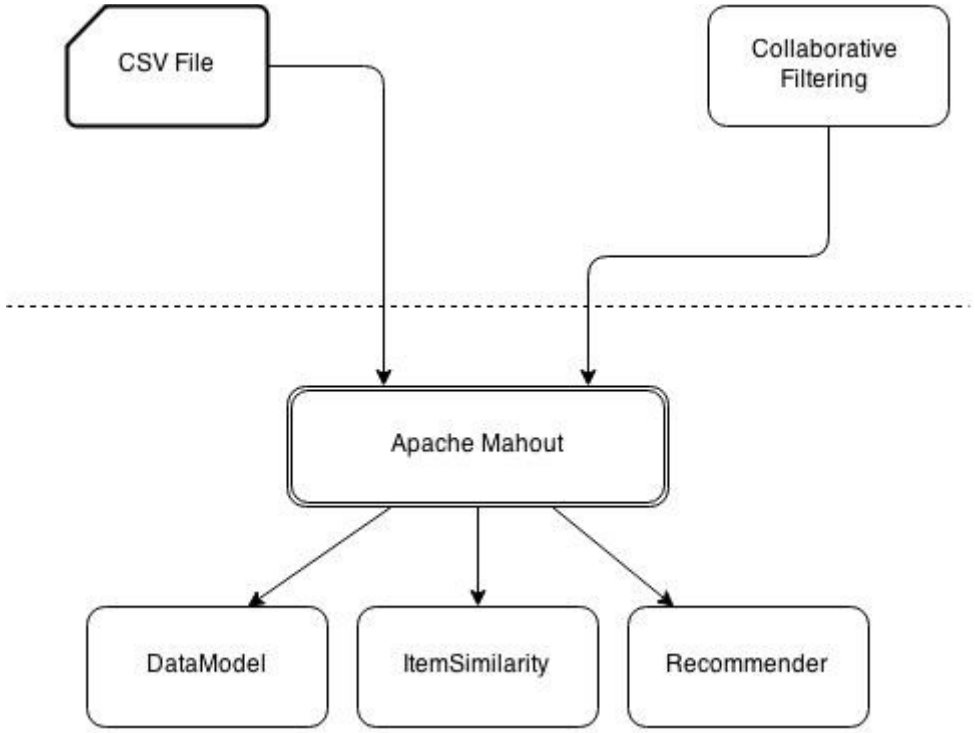
- Using those weights ratings specific to the feature set are predicted
- The above calculated ratings for various feature sets are combined together to obtain the final predicted rating.



5.4 Implementing collaborative filtering using Apache Mahout

The following procedure was used to implement the collaborative based filtering strategy using Apache Mahout:

- A data model is constructed on the dataset.
- Item similarities are computed between all item pairs.
- A weighted average of these similarities are taken to get the final predicted rating.



5.5 Implementing the hybrid approach.

The following procedure is used to implement the various hybrid approaches:

- The content and collaborative filtering implemented above are combined according to three hybrid strategies described in the section
    - Content-based filtering centric hybridization approach: First content based is applied and then for the ratings that are not predicted by content based collaborative is applied.
    - Collaborative filtering centric hybridization approach: First collaborative is applied and then for ratings that are not predicted by collaborative, content based is applied.
    - Bicentric hybridization approach: The mean of the ratings predicted by content based and collaborative gives the predicted rating for this hybrid approach.

6. Observations and Results

The analysis of various recommendation techniques done below involves two parameters – mean absolute error and number of ratings not predicted. The recommendation algorithms are applied on the training data set and the rating for the testing data set are predicted. Also the number of ratings in the testing dataset that couldn't be predicted is also evaluated.

The number of ratings in training data set 770,039 and the number of ratings in testing data set is 85,559.

6.1 Unhybridized approach

a) Content based filtering

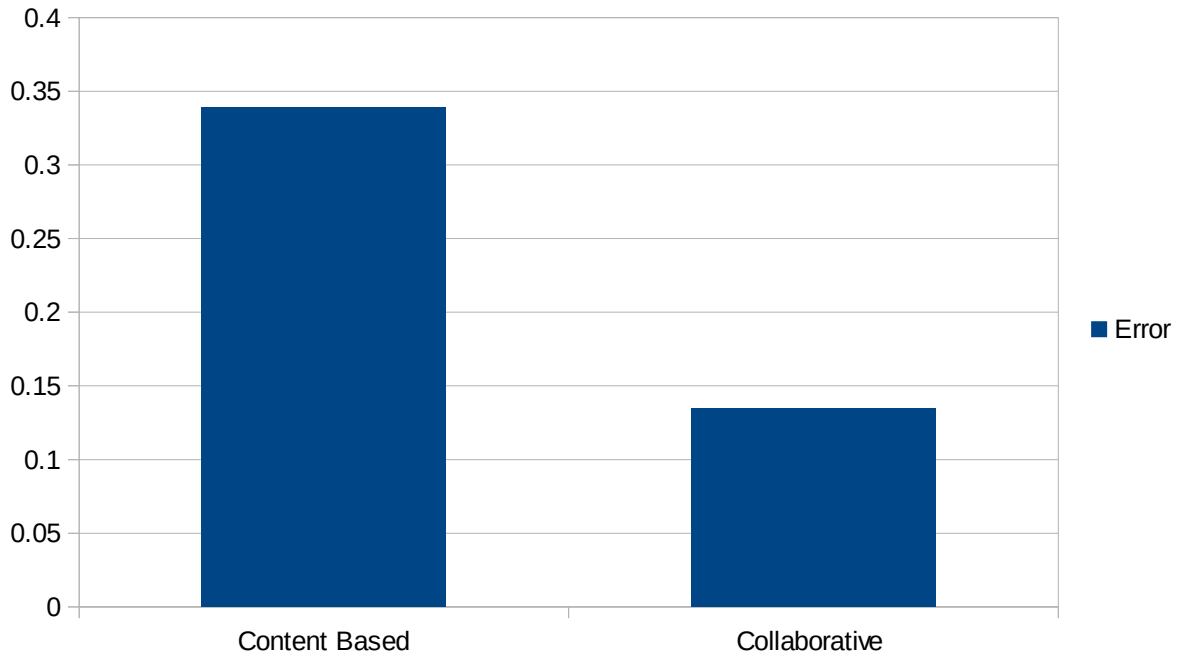
Mean absolute error = 0.339

Number of ratings not predicted = 7

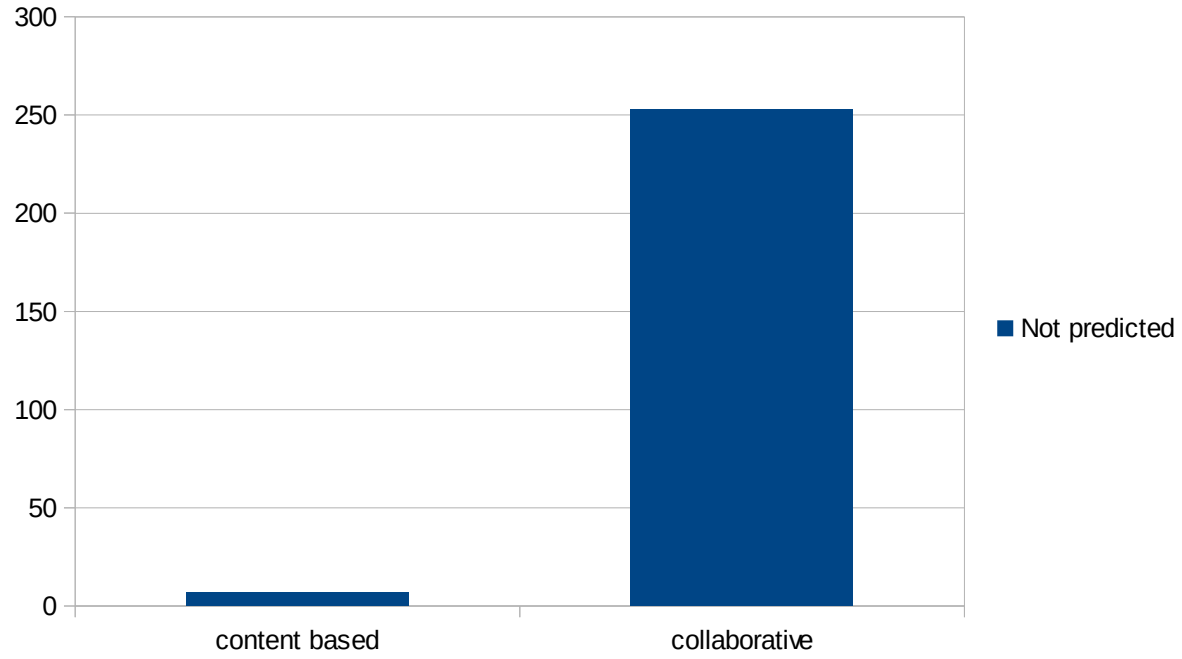
b) Collaborative filtering

Mean absolute error = 0.135

Number of ratings not predicted = 253



Comparing Mean Absolute Errors in Unhybridized Approach



Comparing unpredicted ratings in unhybridized approach

6.2 Hybridized approach

a) Content based filtering centric hybridization

Mean absolute error = 0.339

Number of ratings not predicted = 0

b) Collaborative filtering centric hybridization

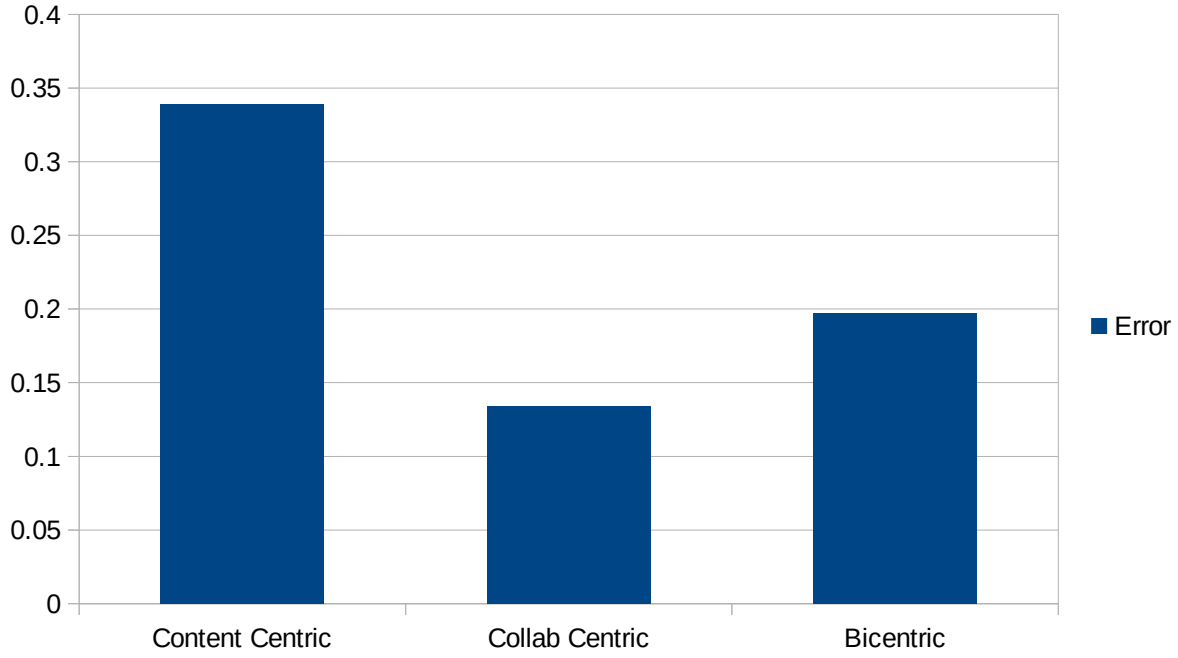
Mean absolute error = 0.134

Number of ratings not predicted = 0

c) Content based and Collaborative filtering Bicentric hybridization

Mean absolute error = 0.197

Number of ratings not predicted = 0



Comparison of various hybridized approaches

6.3 Results

The above analysis shows that *Collaborative filtering centric hybridization (Switching) approach* is the best strategy to build a hybridized recommender system in the given case as it minimizes both the mean absolute error and number of entries not predicted.

7. Conclusion

From the experiment it is evident that out of the three hybridization strategies - Content-based Centric Hybridization, Collaborative Filtering Centric Hybridization and Content-based and Collaborative Bicentric Hybridization approach, the Collaborative Filtering Centric Hybridization approach is the best strategy for developing a hybrid recommender system for MovieLens dataset as it minimizes the mean absolute error and the number of unpredicted ratings.

There is significant improvement in the number of unpredicted ratings as they have been reduced to zero in the hybridization approach. But when it comes to mean absolute error, the improvement in comparison to Collaborative is not much. The future work must aim at reducing the mean absolute error further to produce more quality recommendations by incorporating more effective ways of recommendation and trying to achieve a better combination of individual components. The Content-based and Collaborative Bicentric approach may involve varying degree of combination of two individual components to achieve better results.

8. Acknowldegements

This project couldn't have been done without the presence of Mr Prabhakar Sarma Neog, our respected project guide. He has been with us at each step of our journey. Motivating us when we were short of effort and helping us like careful mentor who has stake in their students' progress.

Also we would like to thank our department, Computer Science and Engineering, NIT Silchar for giving us the opportunity to investigate and research such a wonderful and amusing application of recommender system.

It was a great learning experience. We can assume that everyone in the team learned a lot. This project couldn't have been completed without a sincere team effort.

9. Bibliography

1. ContentBased Movie Recommendation Using Different Feature Sets. Mahiye Uluyagmur, Zehra Cataltepe and Esengul Tayfur.
http://www.iaeng.org/publication/WCECS2012/WCECS2012_pp517521.pdf

2. ItemBased Collaborative Filtering Recommendation Algorithms. Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl
http://files.grouplens.org/papers/www10_sarwar.pdf

3. Hybrid Recommender Systems: Survey and Experiments Robin Burke California State University, Fullerton http://kslab.kaist.ac.kr/kse643/Burke_umuai02.pdf

4. Hybrid Recommender Systems: http://en.wikipedia.org/wiki/Recommender_system

5. Apache Lucene: http://www.tutorialspoint.com/maven/index.htm

6. Apache Lucene: Lucene in Action by Erik Hatcher

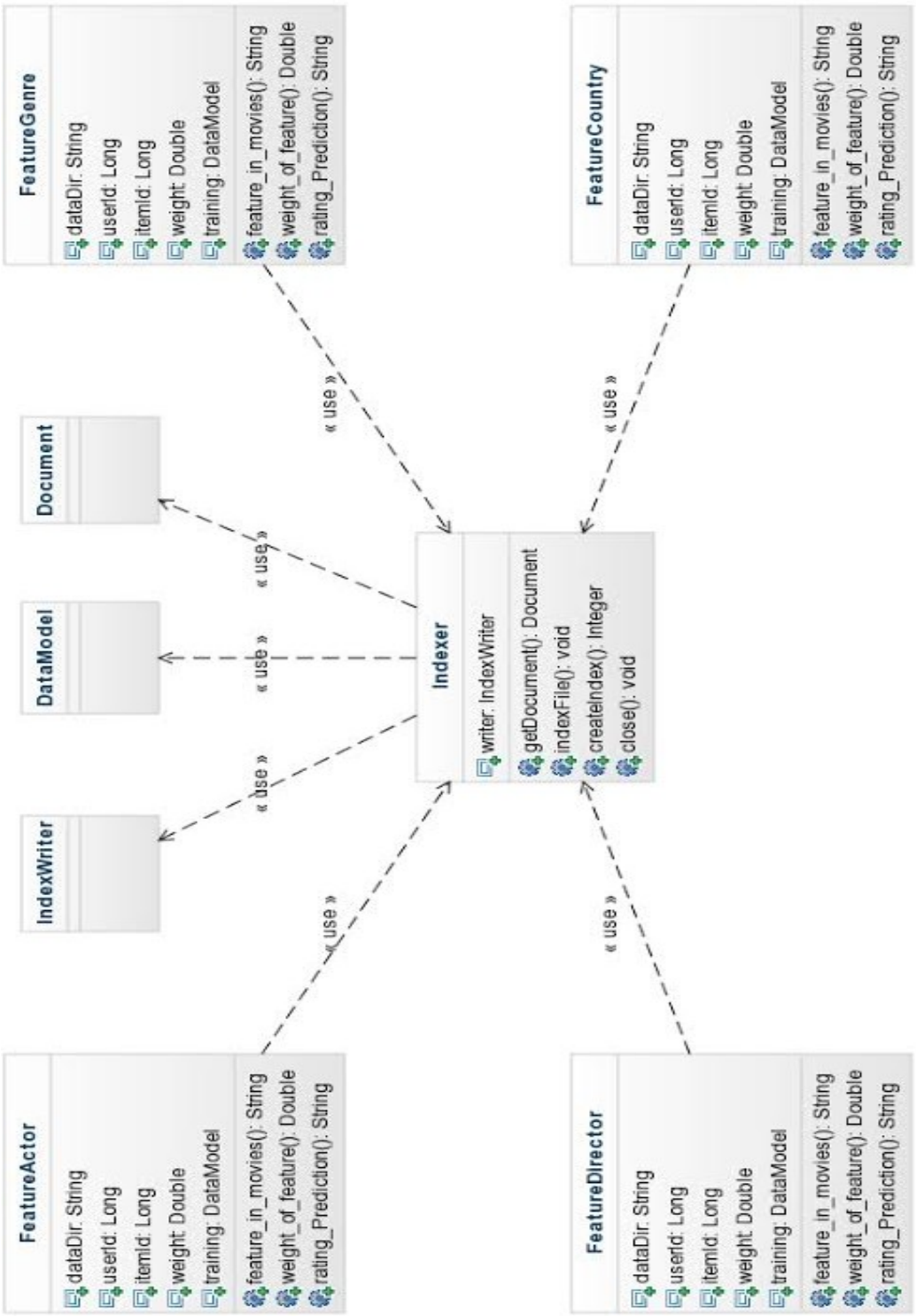
7. Apache Mahout: http://mahout.apache.org/

8. Apache Mahout: Mahout in Action by Sean Owen

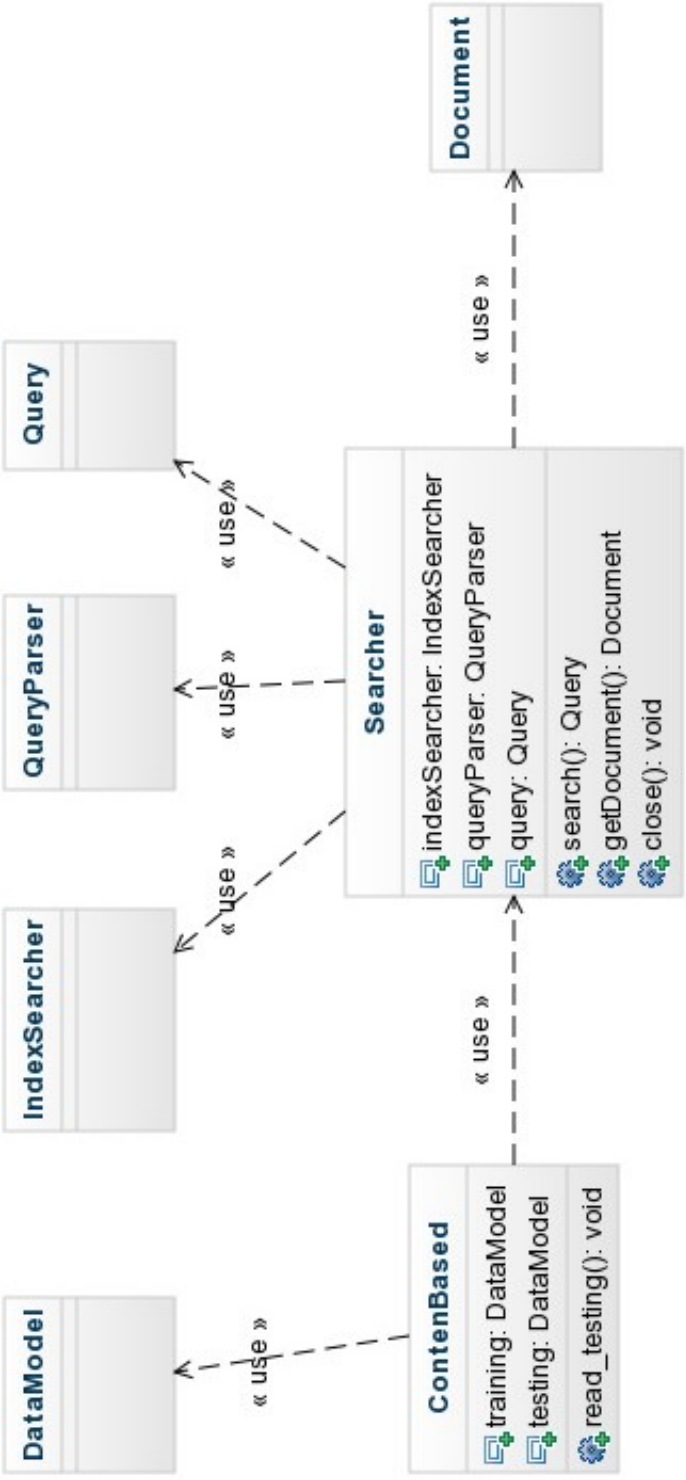
Appendix A

Class Diagrams

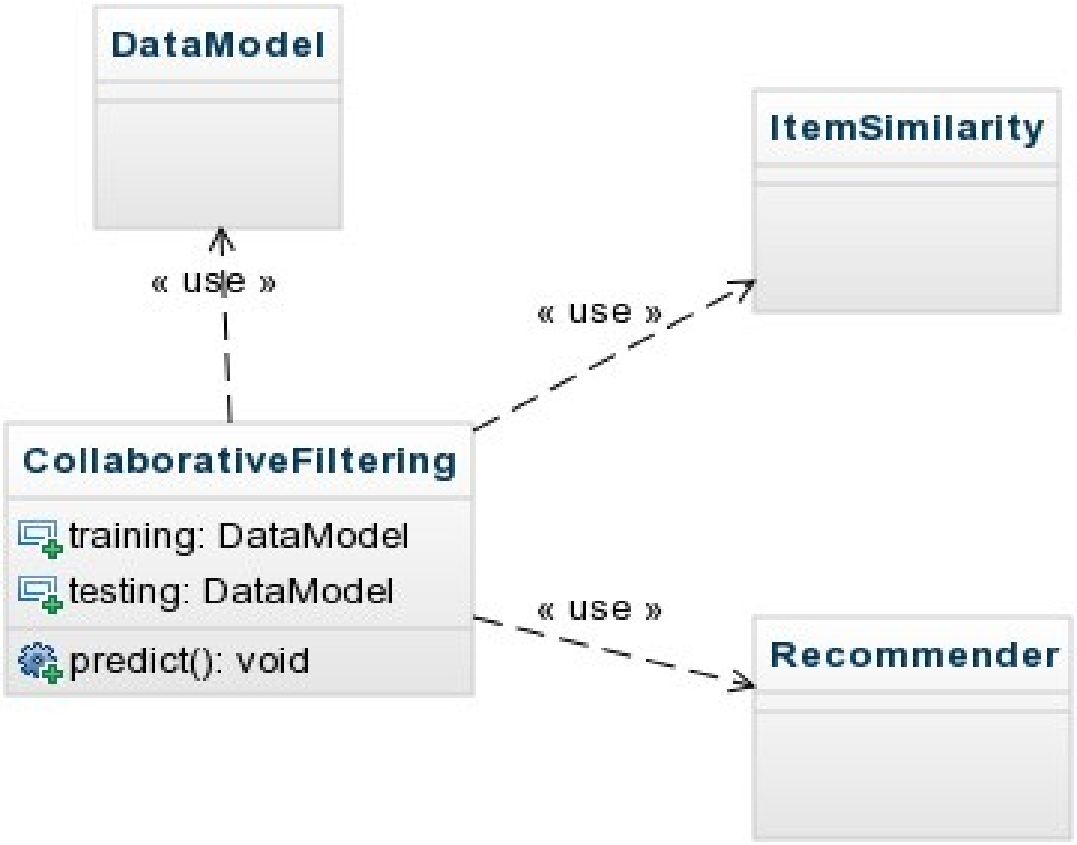
1) Indexing using Apache Lucene



2) Searching using Apache Lucene to implement Content-based recommender



3) Collaborative Filtering using Apache Mahout



Appendix B

Source Codes

1) Dataset.java

```
package dataset;
import java.io.*;
import java.util.*;
public class DataSet {
    public static void main(String ar[])throws Exception
    {
        actors();
        countries();
        directors();
        genres();
    }
    public static void actors()throws IOException
    {
        FileReader fr=new FileReader("src/data/movie_actors.dat");
        BufferedReader br=new BufferedReader(fr);
        TreeSet<String> ts=new TreeSet<String>();
        String line;
        String str="";
        int movie_id=1;
        line=br.readLine();
        while((line=br.readLine()) != null)
        {
            String[] values=line.split("\\t",-1);
            if(Integer.parseInt(values[3]) > 3)
                continue;
            String id="";
            values[1]=values[1].toLowerCase();
            for(int i=0 ; i < values[1].length() ; i++)
            {
                if( Character.isLetterOrDigit( values[1].charAt(i) ) )
                    id=id + values[1].charAt(i);
                else
                    id=id+"_";
            }
            ts.add(id);
            if(Integer.parseInt(values[0]) == movie_id)
```

```
            {
                str=str+id+"\t";
            }
            else
            {
                String temp="C:/data/actors/data/"+ (new
Integer(movie_id).toString()) +".txt" ;
                BufferedWriter bw=new BufferedWriter(new FileWriter(temp));
                if(str.endsWith("\t"))
                    str=str.substring(0,str.length()-1);
                bw.write(str);
                str=id+"\t";
                movie_id=Integer.parseInt(values[0]);
                bw.close();
            }
        }
        String temp="C:/data/actors/data/"+ (new Integer(movie_id).toString())
+".txt" ;
        BufferedWriter bw=new BufferedWriter(new FileWriter(temp));
        if(str.endsWith("\t"))
            str=str.substring(0,str.length()-1);
        bw.write(str);
        bw.close();
        br.close();
        bw=new BufferedWriter(new
FileWriter("C:/data/actors/actors_id_list.txt"));
        System.out.println(ts.size());
        bw.write(ts.size()+"\n");
        for(String i: ts)
            bw.write(i+"\n");
        bw.close();
    }
    public static void genres()throws IOException
    {
        FileReader fr=new FileReader("src/data/movie_genres.dat");
        BufferedReader br=new BufferedReader(fr);
        TreeSet<String> ts=new TreeSet<String>();
        String line;
        String str="";
        int movie_id=1;
        line=br.readLine();
```

```
        while((line=br.readLine()) != null)
        {
            String[] values=line.split("\\t",-1);
            String id="";
            values[1]=values[1].toLowerCase();
            for(int i=0 ; i < values[1].length() ; i++)
            {
                if( Character.isLetterOrDigit( values[1].charAt(i) ) )
                    id=id + values[1].charAt(i);
                else
                    id=id+"_";
            }
            ts.add(id);
            if(Integer.parseInt(values[0]) == movie_id)
            {
                str=str+id+"\t";
            }
            else
            {
                String temp="C:/data/genres/data/"+ (new
Integer(movie_id).toString()) +".txt" ;
                BufferedWriter bw=new BufferedWriter(new FileWriter(temp));
                if(str.endsWith("\t"))
                    str=str.substring(0,str.length()-1);
                bw.write(str);
                str=id+"\t";
                movie_id=Integer.parseInt(values[0]);
                bw.close();
            }
        }
        String temp="C:/data/genres/data/"+ (new Integer(movie_id).toString())
+".txt" ;
        BufferedWriter bw=new BufferedWriter(new FileWriter(temp));
        if(str.endsWith("\t"))
            str=str.substring(0,str.length()-1);
        bw.write(str);
        bw.close();
        br.close();
        bw=new BufferedWriter(new FileWriter("C:/data/genres/genres_list.txt"));
        System.out.println(ts.size());
        bw.write(ts.size()+"\n");
```

```
        for(String i: ts)
            bw.write(i+"\n");
        bw.close();
    }
    public static void directors()throws IOException
    {
        FileReader fr=new FileReader("src/data/movie_directors.dat");//
        BufferedReader br=new BufferedReader(fr);
        TreeSet<String> ts=new TreeSet<String>();
        String line;
        String str="";
        int movie_id=1;
        line=br.readLine();
        while((line=br.readLine()) != null)
        {
            String[] values=line.split("\\t",-1);
            String id="";
            values[1]=values[1].toLowerCase();
            for(int i=0 ; i < values[1].length() ; i++)
            {
                if( Character.isLetterOrDigit( values[1].charAt(i) ) )
                    id=id + values[1].charAt(i);
                else
                    id=id+"_";
            }
            ts.add(id);
            if(Integer.parseInt(values[0]) == movie_id)
            {
                str=str+id+"\t";//
            }
            else
            {
                String temp="C:/data/directors/data/"+ (new
Integer(movie_id).toString()) +".txt" ;//
                BufferedWriter bw=new BufferedWriter(new FileWriter(temp));
                if(str.endsWith("\t"))
                    str=str.substring(0,str.length()-1);
                bw.write(str);
                str=id+"\t";//
                movie_id=Integer.parseInt(values[0]);
                bw.close();
```

```
            }
        }
        String temp="C:/data/directors/data/"+ (new
Integer(movie_id).toString()) +".txt" ;//
        BufferedWriter bw=new BufferedWriter(new FileWriter(temp));
        if(str.endsWith("\t"))
            str=str.substring(0,str.length()-1);
        bw.write(str);
        bw.close();
        br.close();
        bw=new BufferedWriter(new
FileWriter("C:/data/directors/directors_id_list.txt"));
        System.out.println(ts.size());
        bw.write(ts.size()+"\n");
        for(String i: ts)
            bw.write(i+"\n");
        bw.close();
    }
    public static void countries()throws IOException
    {
        FileReader fr=new FileReader("src/data/movie_countries.dat");
        BufferedReader br=new BufferedReader(fr);
        TreeSet<String> ts=new TreeSet<String>();
        String line;
        String str="";
        int movie_id=1;
        line=br.readLine();
        while((line=br.readLine()) != null)
        {
            String[] values=line.split("\\t",-1);
            String id="";
            values[1]=values[1].toLowerCase();
            for(int i=0 ; i < values[1].length() ; i++)
            {
                if( Character.isLetterOrDigit( values[1].charAt(i) ) )
                    id=id + values[1].charAt(i);
                else
                    id=id+"_";
            }
            ts.add(id);
            if(Integer.parseInt(values[0]) == movie_id)
```

```
            {
                str=str+id+"\t";
            }
            else
            {
                String temp="C:/data/countries/data/"+ (new
Integer(movie_id).toString()) +".txt" ;
                BufferedWriter bw=new BufferedWriter(new FileWriter(temp));
                if(str.endsWith("\t"))
                    str=str.substring(0,str.length()-1);
                bw.write(str);
                str=id+"\t";
                movie_id=Integer.parseInt(values[0]);
                bw.close();
            }
        }
        String temp="C:/data/countries/data/"+ (new
Integer(movie_id).toString()) +".txt" ;
        BufferedWriter bw=new BufferedWriter(new FileWriter(temp));
        if(str.endsWith("\t"))
            str=str.substring(0,str.length()-1);
        bw.write(str);
        bw.close();
        br.close();
        bw=new BufferedWriter(new
FileWriter("C:/data/countries/countries_list.txt"));
        System.out.println(ts.size());
        bw.write(ts.size()+"\n");
        for(String i: ts)
            bw.write(i+"\n");
        bw.close();
    }
}
```

2) FeatureActor.java

```
package hybridrecommender;

import java.io.*;
import java.util.ArrayList;
import org.apache.mahout.cf.taste.model.DataModel;
```

```
public class FeatureActor
{
    String dataDir;
    long userId, itemId;
    double weight[];
    DataModel training;

    public FeatureActor(long userId, long itemId,DataModel training)
    {
        dataDir= "C:/data/actors/data";
        this.userId=userId;
        this.itemId=itemId;
        this.training=training;
    }

    public String[] features_in_movie(long itemId) throws Exception
    {
        String path=dataDir+"/"+itemId+".txt";
        BufferedReader br;
        try
        {
            br=new BufferedReader(new FileReader(path));
        }
        catch(FileNotFoundException e)
        {
            return null;
        }
        String line=br.readLine();
        String[] features=line.split("\\t",-1);
        weight=new double[features.length];
        return features;
    }

    public double rating_Prediction()throws Exception
    {
        String[] features=features_in_movie(itemId);
        if(features == null)
            return 0;
        else
        {
```

```
            LuceneSearch search=new LuceneSearch();
            search.dataDir= "C:\\data\\actors\\data";
            search.indexDir= "C:\\data\\actors\\index";
            int i=0;
            for(String feature : features)
            {
                ArrayList items=search.search(feature);
                weight[i++]=weight_of_feature(items);
            }
            double rating=0;
            for(i=0 ; i<weight.length ; i++)
                rating += weight[i];
            return rating;
        }
    }
    public double weight_of_feature(ArrayList<Long> items)throws Exception
    {
        double sum=0;
        for(Long item : items)
        {
            if(training.getPreferenceValue(userId, item) != null)
            {
                sum += training.getPreferenceValue(userId, item);
            }
        }
        int n=training.getItemIDsFromUser(userId).size();
        return sum/n;
    }
}
```

3) FeatureDirector.java

```
package hybridrecommender;

import java.io.*;
import java.util.ArrayList;
import org.apache.mahout.cf.taste.model.DataModel;

public class FeatureDirector
{
```

```
String dataDir;
long userId, itemId;
double weight[];
DataModel training;

public FeatureDirector(long userId, long itemId,DataModel training)
{
    dataDir= "C:/data/directors/data";
    this.userId=userId;
    this.itemId=itemId;
    this.training=training;
}

public String[] features_in_movie(long itemId) throws Exception
{
    String path=dataDir+"/"+itemId+".txt";
    BufferedReader br;
    try
    {
        br=new BufferedReader(new FileReader(path));
    }
    catch(FileNotFoundException e)
    {
        return null;
    }
    String line=br.readLine();
    String[] features=line.split("\\t",-1);
    weight=new double[features.length];
    return features;
}

public double rating_Prediction()throws Exception
{
    String[] features=features_in_movie(itemId);
    if(features == null)
        return 0;
    else
    {
        LuceneSearch search=new LuceneSearch();
        search.dataDir= "C:\\data\\directors\\data";
```

```
            search.indexDir= "C:\\data\\directors\\index";
            int i=0;
            for(String feature : features)
            {
                ArrayList items=search.search(feature);
                weight[i++]=weight_of_feature(items);
            }
            double rating=0;
            for(i=0 ; i<weight.length ; i++)
                rating += weight[i];
            return rating;
        }
    }
    public double weight_of_feature(ArrayList<Long> items)throws Exception
    {
        long sum=0;
        for (long item : items)
        {
            if(training.getPreferenceValue(userId, item) != null)
                sum += training.getPreferenceValue(userId, item);
        }
        int n=training.getItemIDsFromUser(userId).size();
        return 1.0*sum/n;
    }
}
```

4) FeatureGenre.java

```
package hybridrecommender;

import java.io.*;
import java.util.ArrayList;
import org.apache.mahout.cf.taste.model.DataModel;

public class FeatureGenre
{
    String dataDir;
    long userId, itemId;
    double weight[];
    DataModel training;
```

```
public FeatureGenre(long userId, long itemId,DataModel training)
{
    dataDir= "C:/data/genres/data";
    this.userId=userId;
    this.itemId=itemId;
    this.training=training;
}

public String[] features_in_movie(long itemId) throws Exception
{
    String path=dataDir+"/"+itemId+".txt";
    BufferedReader br;
    try
    {
        br=new BufferedReader(new FileReader(path));
    }
    catch(FileNotFoundException e)
    {
        return null;
    }
    String line=br.readLine();
    String[] features=line.split("\\t",-1);
    weight=new double[features.length];
    return features;
}

public double rating_Prediction()throws Exception
{
    String[] features=features_in_movie(itemId);
    if(features == null)
        return 0;
    else
    {
        LuceneSearch search=new LuceneSearch();
        search.dataDir= "C:\\data\\genres\\data";
        search.indexDir= "C:\\data\\genres\\index";
        int i=0;
        for(String feature : features)
        {
```

```
                ArrayList items=search.search(feature);
                weight[i++]=weight_of_feature(items);
            }
            double rating=0;
            for(i=0 ; i<weight.length ; i++)
                rating += weight[i];
            return rating;
        }
    }
    public double weight_of_feature(ArrayList<Long> items)throws Exception
    {
        double sum=0;
        for(Long item : items)
        {
            if(training.getPreferenceValue(userId, item) != null)
            {
                sum += training.getPreferenceValue(userId, item);
            }
        }
        int n=training.getItemIDsFromUser(userId).size();
        return sum/n;
    }
}
```

5) FeatureCountry.java

```
package hybridrecommender;

import java.io.*;
import java.util.ArrayList;
import org.apache.mahout.cf.taste.model.DataModel;

public class FeatureCountry
{
    String dataDir;
    long userId, itemId;
    double weight[];
    DataModel training;

    public FeatureCountry(long userId, long itemId,DataModel training)
    {
        dataDir= "C:/data/countries/data";
```

```
        this.userId=userId;
        this.itemId=itemId;
        this.training=training;
    }

    public String[] features_in_movie(long itemId) throws Exception
    {
        String path=dataDir+"/"+itemId+".txt";
        BufferedReader br;
        try
        {
            br=new BufferedReader(new FileReader(path));
        }
        catch(FileNotFoundException e)
        {
            return null;
        }
        String line=br.readLine();
        if(line == null)
            return null;
        String[] features=line.split("\\t",-1);
        weight=new double[features.length];
        return features;
    }

    public double rating_Prediction()throws Exception
    {
        String[] features=features_in_movie(itemId);
        if(features == null)
            return 0;
        else
        {
            LuceneSearch search=new LuceneSearch();
            search.dataDir= "C:\\data\\countries\\data";
            search.indexDir= "C:\\data\\countries\\index";
            int i=0;
            for(String feature : features)
            {
                ArrayList items=search.search(feature);
                weight[i++]=weight_of_feature(items);
```

```
            }
            double rating=0;
            for(i=0 ; i<weight.length ; i++)
                rating += weight[i];
            return rating;
        }
    }
    public double weight_of_feature(ArrayList<Long> items)throws Exception
    {
        double sum=0;
        for(Long item : items)
        {
            if(training.getPreferenceValue(userId, item) != null)
            {
                sum += training.getPreferenceValue(userId, item);
            }
        }
        int n=training.getItemIDsFromUser(userId).size();
        return sum/n;
    }
}
```

6) LuceneConstants.java

```
package hybridrecommender;

public class LuceneConstants {
   public static final String CONTENTS="contents";
   public static final String FILE_NAME="filename";
   public static final String FILE_PATH="filepath";
   public static final int MAX_SEARCH = 10000;
}
```

7) TextFileFilter.java

```
package hybridrecommender;

import java.io.File;
import java.io.FileFilter;

public class TextFileFilter implements FileFilter {

   @Override
```

```
   public boolean accept(File pathname) {
      return pathname.getName().toLowerCase().endsWith(".txt");
   }
}
```

8) Indexer.java

```
package hybridrecommender;

import java.io.File;
import java.io.FileFilter;
import java.io.FileReader;
import java.io.IOException;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.util.Version;

public class Indexer {

   private IndexWriter writer;

   public Indexer(String indexDirectoryPath) throws IOException{
      //this directory will contain the indexes
      Directory indexDirectory =
         FSDirectory.open(new File(indexDirectoryPath));

      //create the indexer
      writer = new IndexWriter(indexDirectory,
         new StandardAnalyzer(Version.LUCENE_36),true,
         IndexWriter.MaxFieldLength.UNLIMITED);
   }

   public void close() throws CorruptIndexException, IOException{
      writer.close();
   }
```

```
private Document getDocument(File file) throws IOException{
   Document document = new Document();

   //index file contents
   Field contentField = new Field(LuceneConstants.CONTENTS,
      new FileReader(file));
   //index file name
   Field fileNameField = new Field(LuceneConstants.FILE_NAME,
      file.getName(),
      Field.Store.YES,Field.Index.NOT_ANALYZED);
   //index file path
   Field filePathField = new Field(LuceneConstants.FILE_PATH,
      file.getCanonicalPath(),
      Field.Store.YES,Field.Index.NOT_ANALYZED);

   document.add(contentField);
   document.add(fileNameField);
   document.add(filePathField);

   return document;
}

private void indexFile(File file) throws IOException{
   System.out.println("Indexing "+file.getCanonicalPath());
   Document document = getDocument(file);
   writer.addDocument(document);
}

public int createIndex(String dataDirPath, FileFilter filter)
   throws IOException{
   //get all files in the data directory
   File[] files = new File(dataDirPath).listFiles();

   for (File file : files) {
      if(!file.isDirectory()
         && !file.isHidden()
         && file.exists()
         && file.canRead()
         && filter.accept(file)
```

```
         ){
            indexFile(file);
         }
      }
      return writer.numDocs();
   }
}
```

9) LuceneSearch.java

```
package hybridrecommender;

import java.io.IOException;
import java.util.ArrayList;

import org.apache.lucene.document.Document;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;

public class LuceneSearch {

   String indexDir;
   String dataDir;
   Indexer indexer;
   Searcher searcher;

   public ArrayList search(String searchQuery) throws IOException,
ParseException
   {
      searcher = new Searcher(indexDir);
      TopDocs hits = searcher.search(searchQuery);
      ArrayList<Long> items=new ArrayList<>();
      for(ScoreDoc scoreDoc : hits.scoreDocs)
      {
            Document doc = searcher.getDocument(scoreDoc);
            String temp=doc.get(LuceneConstants.FILE_NAME);
            temp=temp.substring(0,temp.indexOf('.'));
            items.add(Long.parseLong(temp));
      }
      searcher.close();
      return items;
```

```
    }
}
```

10) Searcher.java

```
package hybridrecommender;

import java.io.File;
import java.io.IOException;

import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.util.Version;

public class Searcher {

   IndexSearcher indexSearcher;
   QueryParser queryParser;
   Query query;

   public Searcher(String indexDirectoryPath)
      throws IOException{
      Directory indexDirectory =
         FSDirectory.open(new File(indexDirectoryPath));
      indexSearcher = new IndexSearcher(indexDirectory);
      queryParser = new QueryParser(Version.LUCENE_36,
         LuceneConstants.CONTENTS,
         new StandardAnalyzer(Version.LUCENE_36));
   }

   public TopDocs search( String searchQuery)
      throws IOException, ParseException{
```

```
      query = queryParser.parse(searchQuery);
      return indexSearcher.search(query, LuceneConstants.MAX_SEARCH);
   }

   public TopDocs search(Query query) throws IOException, ParseException{
      return indexSearcher.search(query, LuceneConstants.MAX_SEARCH);
   }


   public Document getDocument(ScoreDoc scoreDoc)
      throws CorruptIndexException, IOException{
      return indexSearcher.doc(scoreDoc.doc);
   }

   public void close() throws IOException{
      indexSearcher.close();
   }
}
```

11) HybridRecommender.java

```
package hybridrecommender;
public class HybridRecommender
{
    public static void main(String[] args)throws Exception
    {
        ContentBased recommender=new ContentBased();
        recommender.read_testing();
        CollaborativeFiltering recommender=new CollaborativeFiltering();
        recommender.predict();
    }
}
```

12) ContentBased.java

```
package hybridrecommender;
import java.io.*;
import java.util.ArrayList;
import org.apache.mahout.cf.taste.model.DataModel;
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;
import org.apache.mahout.cf.taste.impl.common.LongPrimitiveIterator;
```

```
public class ContentBased
{
DataModel training;
DataModel testing;
    public ContentBased()throws Exception
    {
        training=new FileDataModel(new File("src/dataset/training.csv"));
        testing=new FileDataModel(new File("src/dataset/testing.csv"));
    }
    public void read_testing()throws Exception
    {
        FeatureCountry object;
        BufferedWriter bw=new BufferedWriter(new
FileWriter("src/dataset/country.txt"));
        for(LongPrimitiveIterator users = training.getUserIDs();
users.hasNext();)
        {
            long userId = users.nextLong();
            ArrayList<Long> itemList=new ArrayList<>();
            ArrayList<Double> ratingList=new ArrayList<>();

            double min=100000 , max=-1;

            for(LongPrimitiveIterator items = training.getItemIDs();
items.hasNext();)
            {
                long itemId = items.nextLong();
                if(training.getPreferenceValue(userId, itemId) != null)
                {
                    object=new FeatureCountry(userId,itemId,training);
                    double rating=object.rating_Prediction();
                    if(rating > max)
                        max=rating;
                    else if(rating < min)
                       min=rating;
                }
            }

            for(LongPrimitiveIterator items = testing.getItemIDs();
items.hasNext();)
            {
                long itemId = items.nextLong();
```

```
                if(testing.getPreferenceValue(userId, itemId) != null)
                {
                    itemList.add(itemId);
                    object=new FeatureCountry(userId,itemId,training);
                    double rating=object.rating_Prediction();
                    //System.out.println(userId+" , "+itemId+" , "+rating);
                    ratingList.add(rating);
                    if(rating > max)
                        max=rating;
                    else if(rating < min)
                        min=rating;
                }
            }
            //System.out.println(min+"    "+max);

            if( min != max )
            {
                int size=ratingList.size();
                for(int i=0 ; i < size ; i++)
                {
                    if(testing.getPreferenceValue(userId, itemList.get(i)) !=
null)
                    {
                        double j=ratingList.get(i);
                        j=(j-min)/(max-min);
                        //ratingList.set(i,j);
                        bw.write(userId+","+itemList.get(i)+","+j+"\n");
                        //System.out.println(userId+","+itemList.get(i)
+","+ratingList.get(i)+","+j);
                    }
                }
            }
            else if(min != 0)
            {
                int size=ratingList.size();
                for(int i=0 ; i < size ; i++)
                {
                    if(testing.getPreferenceValue(userId, itemList.get(i)) !=
null)
                    {
                        bw.write(userId+","+itemList.get(i)+",1.0"+"\n");
```

```
                    }
                }
            }
            else
            {
                int size=ratingList.size();
                for(int i=0 ; i < size ; i++)
                {
                    if(testing.getPreferenceValue(userId, itemList.get(i)) !=
null)
                    {

                        bw.write(userId+","+itemList.get(i)+",0.0"+"\n");

                    }
                }
            }
            System.out.println(userId);
        }
        bw.close();
    }
}
```

13) CollaborativeFiltering.java

```
package hybridrecommender;

import java.io.*;
import org.apache.mahout.cf.taste.common.NoSuchItemException;
import org.apache.mahout.cf.taste.model.DataModel;
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;
import org.apache.mahout.cf.taste.impl.common.LongPrimitiveIterator;
import org.apache.mahout.cf.taste.impl.neighborhood.ThresholdUserNeighborhood;
import org.apache.mahout.cf.taste.impl.recommender.GenericItemBasedRecommender;
import org.apache.mahout.cf.taste.impl.recommender.GenericUserBasedRecommender;
import org.apache.mahout.cf.taste.impl.similarity.PearsonCorrelationSimilarity;
import org.apache.mahout.cf.taste.neighborhood.UserNeighborhood;
import org.apache.mahout.cf.taste.recommender.Recommender;
import org.apache.mahout.cf.taste.similarity.ItemSimilarity;
import org.apache.mahout.cf.taste.similarity.UserSimilarity;
public class CollaborativeFiltering
```

```
{
DataModel training;
DataModel testing;
    public CollaborativeFiltering()throws Exception
    {
        training=new FileDataModel(new File("src/dataset/training.csv"));
        testing=new FileDataModel(new File("src/dataset/testing.csv"));
    }
    public void predict()throws Exception
    {
        BufferedWriter bw=new BufferedWriter(new
FileWriter("src/dataset/collaborative_prediction1.txt"));
        //ItemSimilarity itemSimilarity =new
PearsonCorrelationSimilarity(training);
        //Recommender recommender =new GenericItemBasedRecommender(training,
itemSimilarity);
        UserSimilarity similarity = new PearsonCorrelationSimilarity(training);
        UserNeighborhood neighborhood = new ThresholdUserNeighborhood(0.1,
similarity, training);
        Recommender recommender =new GenericUserBasedRecommender(training,
neighborhood, similarity);
        for(LongPrimitiveIterator users = testing.getUserIDs();
users.hasNext();)
        {
            long userId = users.nextLong();
            for(LongPrimitiveIterator items = testing.getItemIDs();
items.hasNext();)
            {
                long itemId = items.nextLong();
                double rating=0;
                if(testing.getPreferenceValue(userId, itemId) != null)
                {
                    try
                    {
                        rating=recommender.estimatePreference(userId, itemId);
                    }
                    catch(NoSuchItemException e)
                    {
                        rating=0;
                    }
                    if(Double.isNaN(rating))
                        rating=0;
                    bw.write(userId+","+itemId+","+rating+"\n");
```

```
                }
            }
        }
        bw.close();
    }
}
```

14) Error_Collaborative.java

```
package hybridrecommender;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Error_Collaborative {
    public static void main(String ar[])throws IOException
    {
        BufferedReader br1=new BufferedReader(new
FileReader("src/dataset/collaborative_prediction.txt"));
        BufferedReader br2=new BufferedReader(new
FileReader("src/dataset/testing.csv"));
        String line;
        double error=0;
        int count=0;
        while((line=br1.readLine()) != null)
        {
            String[] values=line.split(",",-1);
            int userId=Integer.parseInt(values[0]);
            int itemId=Integer.parseInt(values[1]);
            double r1=Double.parseDouble(values[2]);
            line=br2.readLine();
            values=line.split(",",-1);
            double r2=Double.parseDouble(values[2]);
            error=error + Math.abs( r2/5 - r1/5 );
            if(r1 == 0)
                count++;
        }
        System.out.println("Mean Absolute Error= "+error/85559);
        System.out.println("Number of Ratings not Predicted= "+count);
        br1.close();
```

```
        br2.close();
    }

}
```

15) Error_Content.java

```
package hybridrecommender;

import java.io.*;
public class Error_Content {
    public static void main(String ar[])throws IOException
    {
        BufferedReader br1=new BufferedReader(new
FileReader("src/dataset/actor.txt"));
        BufferedReader br2=new BufferedReader(new
FileReader("src/dataset/director.txt"));
        BufferedReader br3=new BufferedReader(new
FileReader("src/dataset/genre.txt"));
        BufferedReader br4=new BufferedReader(new
FileReader("src/dataset/country.txt"));
        BufferedReader br5=new BufferedReader(new
FileReader("src/dataset/testing.csv"));
        String line;
        double error=0;
        int count=0;
        while((line=br1.readLine()) != null)
        {
            String[] values=line.split(",",-1);
            int userId=Integer.parseInt(values[0]);
            int itemId=Integer.parseInt(values[1]);
            double r1=Double.parseDouble(values[2]);
            line=br2.readLine();
            values=line.split(",",-1);
            double r2=Double.parseDouble(values[2]);
            line=br3.readLine();
            values=line.split(",",-1);
            double r3=Double.parseDouble(values[2]);
            line=br4.readLine();
            values=line.split(",",-1);
            double r4=Double.parseDouble(values[2]);
            line=br5.readLine();
```

```
            values=line.split(",",-1);
            double r5=Double.parseDouble(values[2]);
            error=error + Math.abs( r5/5 - (r1+r2+r3+r4)/4 );
            if(r1+r2+r3+r4 == 0)
                count++;
        }
        System.out.println("Mean Absolute Error= "+error/85559);
        System.out.println("Number of Ratings not Predicted= "+count);
        br1.close();
        br2.close();
        br3.close();
        br4.close();
        br5.close();
    }
}
```

16) Error_Hybrid.java (bicentric)

```
package hybridrecommender;

import java.io.*;
public class Error_Hybrid {
    public static void main(String ar[])throws IOException
    {
        BufferedReader br1=new BufferedReader(new
FileReader("src/dataset/actor.txt"));
        BufferedReader br2=new BufferedReader(new
FileReader("src/dataset/director.txt"));
        BufferedReader br3=new BufferedReader(new
FileReader("src/dataset/genre.txt"));
        BufferedReader br4=new BufferedReader(new
FileReader("src/dataset/country.txt"));
        BufferedReader br5=new BufferedReader(new
FileReader("src/dataset/collaborative_prediction.txt"));
        BufferedReader br6=new BufferedReader(new
FileReader("src/dataset/testing.csv"));
        String line;
        double error=0;
        int count=0;
        while((line=br1.readLine()) != null)
        {
            String[] values=line.split(",",-1);
```

```
            int userId=Integer.parseInt(values[0]);
            int itemId=Integer.parseInt(values[1]);
            double r1=Double.parseDouble(values[2]);
            line=br2.readLine();
            values=line.split(",",-1);
            double r2=Double.parseDouble(values[2]);
            line=br3.readLine();
            values=line.split(",",-1);
            double r3=Double.parseDouble(values[2]);
            line=br4.readLine();
            values=line.split(",",-1);
            double r4=Double.parseDouble(values[2]);
            line=br5.readLine();
            values=line.split(",",-1);
            double r5=Double.parseDouble(values[2]);
            line=br6.readLine();
            values=line.split(",",-1);
            double r6=Double.parseDouble(values[2]);
            error=error + Math.abs( r6/5 - ((r1+r2+r3+r4)+ r5)/9 );
            if(r1+r2+r3+r4+r5 == 0)
                count++;
        }
        System.out.println("Mean Absolute Error= "+error/85559);
        System.out.println("Number of Ratings not Predicted= "+count);
        br1.close();
        br2.close();
        br3.close();
        br4.close();
        br5.close();
    }

}
```

17) Error_Hybrid_Content.java (content centric)

```
package hybridrecommender;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
```

```
public class Error_Hybrid_Content {
    public static void main(String ar[])throws IOException
    {
        BufferedReader br1=new BufferedReader(new
FileReader("src/dataset/actor.txt"));
        BufferedReader br2=new BufferedReader(new
FileReader("src/dataset/director.txt"));
        BufferedReader br3=new BufferedReader(new
FileReader("src/dataset/genre.txt"));
        BufferedReader br4=new BufferedReader(new
FileReader("src/dataset/country.txt"));
        BufferedReader br5=new BufferedReader(new
FileReader("src/dataset/collaborative_prediction.txt"));
        BufferedReader br6=new BufferedReader(new
FileReader("src/dataset/testing.csv"));
        String line;
        double error=0;
        int count=0;
        while((line=br1.readLine()) != null)
        {
            String[] values=line.split(",",-1);
            int userId=Integer.parseInt(values[0]);
            int itemId=Integer.parseInt(values[1]);
            double r1=Double.parseDouble(values[2]);
            line=br2.readLine();
            values=line.split(",",-1);
            double r2=Double.parseDouble(values[2]);
            line=br3.readLine();
            values=line.split(",",-1);
            double r3=Double.parseDouble(values[2]);
            line=br4.readLine();
            values=line.split(",",-1);
            double r4=Double.parseDouble(values[2]);
            line=br5.readLine();
            values=line.split(",",-1);
            double r5=Double.parseDouble(values[2]);
            line=br6.readLine();
            values=line.split(",",-1);
            double r6=Double.parseDouble(values[2]);
            double a=(r1+r2+r3+r4)/4;
            if(a > 0)
```

```
                error=error + Math.abs(r6/5 - a);
            else
                error=error + Math.abs(r6/5 - r5/5);
            if(r1+r2+r3+r4+r5 == 0)
                count++;
        }
        System.out.println("Mean Absolute Error= "+error/85559);
        System.out.println("Number of Ratings not Predicted= "+count);
        br1.close();
        br2.close();
        br3.close();
        br4.close();
        br5.close();
    }
}
```

18) Error_Hybrid_Collaborative.java (collaborative centric)

```
package hybridrecommender;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Error_Hybrid_Collaborative {
        public static void main(String ar[])throws IOException
    {
        BufferedReader br1=new BufferedReader(new
FileReader("src/dataset/actor.txt"));
        BufferedReader br2=new BufferedReader(new
FileReader("src/dataset/director.txt"));
        BufferedReader br3=new BufferedReader(new
FileReader("src/dataset/genre.txt"));
        BufferedReader br4=new BufferedReader(new
FileReader("src/dataset/country.txt"));
        BufferedReader br5=new BufferedReader(new
FileReader("src/dataset/collaborative_prediction.txt"));
        BufferedReader br6=new BufferedReader(new
FileReader("src/dataset/testing.csv"));
        String line;
        double error=0;
        int count=0;
        while((line=br1.readLine()) != null)
        {
```

```
            String[] values=line.split(",",-1);
            int userId=Integer.parseInt(values[0]);
            int itemId=Integer.parseInt(values[1]);
            double r1=Double.parseDouble(values[2]);
            line=br2.readLine();
            values=line.split(",",-1);
            double r2=Double.parseDouble(values[2]);
            line=br3.readLine();
            values=line.split(",",-1);
            double r3=Double.parseDouble(values[2]);
            line=br4.readLine();
            values=line.split(",",-1);
            double r4=Double.parseDouble(values[2]);
            line=br5.readLine();
            values=line.split(",",-1);
            double r5=Double.parseDouble(values[2]);
            line=br6.readLine();
            values=line.split(",",-1);
            double r6=Double.parseDouble(values[2]);
            double a=(r1+r2+r3+r4)/4;
            if(r5 > 0)
                error=error + Math.abs(r6/5 - r5/5);
            else
                error=error + Math.abs(r6/5 - a);
            if(r1+r2+r3+r4+r5 == 0)
                count++;
        }
        System.out.println("Mean Absolute Error= "+error/85559);
        System.out.println("Number of Ratings not Predicted= "+count);
        br1.close();
        br2.close();
        br3.close();
        br4.close();
        br5.close();
    }
}
```