Visvesvaraya Technological University

A Project Report
on

A Weight based Personalized Recommendation using Idiocentric and Collaborative Recommendation

submitted in partial fulfillment of the requirements for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE & ENGINEERING

by

Vijesh M- 1PI09CS119
Sanjay G Huilgol- 1PI09CS083
Vijay Mahantesh SM- 1PI09CS118

under the guidance of
Dr. Kavi Mahesh

January 2013 - May 2013

Department of Computer Science & Engineering
PES Institute of Technology
(An autonomous institute under VTU, Belgaum and UGC, New Delhi)
100 Feet Ring Road, BSK-III Stage, Bangalore-560085

PES Institute of Technology

(An autonomous institute under VTU, Belgaum and UGC, New Delhi)

100 Feet Ring Road, BSK-III Stage, Bangalore-560085

Department of Computer Science & Engineering

CERTIFICATE

Certified that the project work entitled **"A Weight based Personalized Recommendation using Idiocentric and Collaborative Recommendation"** is bonafide work carried out by

Vijesh M, 1PI09CS119
Sanjay G Huilgol, 1PI09CS083
Vijay Mahantesh SM, 1PI09CS118

in partial fulfillment for the award of degree of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the academic semester January 2013 - May 2013. It is certified that all corrections and suggestions indicated for internal assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the said Bachelor of Engineering degree.

Signature of the Guide	Signature of the HOD	Signature of the Principal
Dr. Kavi Mahesh	Prof. Nitin V Pujari	Dr. K N B Murthy

External Viva

Name of the examiners	Signature with date
1.	....
2.	....

Acknowledgement:

The thrill and excitement of finishing a project can be felt in the last stage when the report is being written. When we look back from where it all started we recollect all the people who brought this project from infancy to its completion.

At the outset, we would like to express our heartfelt thanks towards our Principal **Dr. K N Balasubramanya Murthy** for providing us with a congenial environment to carry out our academic projects.

We express our deep sense of gratitude to our HOD, **Prof. Nitin V Pujari** for his invaluable support and guidance and providing all the necessary resources, which always encourages us to excel.

We consider it a privilege to acknowledge and express gratitude to our guide, **Dr. Kavi Mahesh** for his continuous support and motivational guidance throughout.

We also express our sincere thanks to all the teaching and non-teaching staff of the Department of Computer Science, PESIT for their kind support and ever helping nature. We would also like to thank our project coordinator, **Mr. Badri Prasad**, for all his cordial support during the project timeline.

A friend in need is a friend indeed is rightly said. We take this opportunity to thank all our friends for their constant and invaluable feedback which encouraged us to a great extent. In particular, we would like to thank **Sandeep Raju P** and **Rakesh Kumar R** for their invaluable effort in helping us with designing the approach and building the user interface.

Lastly, we are very grateful to several web resources like Google, Wikipedia and other useful technical articles without which this project would not have seen the light of day.

- The Team

Abstract:

Recommending items to users is one of the most frequently stumbled upon requirements in the e-commerce, or the internet in general. Many techniques have been developed for recommending items. The techniques can be broadly classified into three categories: Collaborative Filtering, Content Based Filtering and Hybrid Recommendations. In this project, we have developed a Graph-based recommendation algorithm that uses both the idiocentric and collaborative behavior of the user to recommend the items back to the users. We have developed methods to carry out unsupervised dimensionality reduction on the dataset. We have deduced some of the characteristic qualities of the users and have constructed a custom built profile for each of them. We have used the movielens_1m dataset to test the accuracy of our algorithm. A survey paper on movielens dataset by Bao and Xia, from autumn 2012 Machine Learning Course at Stanford University, compares four parameterized approaches for rating prediction. The least RMS error from all the approaches was found to be 0.917. Using the non-parameterized approach that we have developed, we were able to achieve an RMS error of 0.903.

Contents

Chapter 1

Introduction

In this age of information overload, people use a variety of strategies to make choices about what to buy, how to spend their leisure time, what movies to watch, etc. Recommender systems automate some of these strategies with the goal of providing affordable, personal, and high-quality recommendations. Recommender Systems are primarily directed towards individuals who lack sufficient personal experience or competence to evaluate the potentially overwhelming number of alternative items that a Web site, for example, may offer. Since recommendations are usually personalized, different users or user groups receive diverse suggestions.

In their simplest form, personalized recommendations are offered as ranked lists of items. In performing this ranking, recommender systems try to predict what the most suitable products or services are, based on the user's preferences and constraints. In order to complete such a computational task, they collect from users their preferences, which are either explicitly expressed, e.g., as ratings for products, or are inferred by interpreting user actions. For instance, navigation to a particular product page can be considered as an implicit sign of preference for the items shown on that page.

The major problems faced in recommender systems are as follows:

- In-homogeneity in properties of items An item is associated with a set of properties. Given a set of items, there is no constraint that all the items should possess the same set of properties. Generally, recommendation systems assume homogeneity in properties and are not applicable to inhomogeneous item datasets.
- Dimensionality Reduction and Feature Selection on the item dataset In machine learning and statistics, dimensionality reduction is the process of reducing the number of features under consideration, and can be divided into feature selection and feature extraction. Feature selection approaches try to find a subset of the original features. Feature extraction transforms the data in the high-dimensional space.
- User Profiling A user profile indicates the information needs or preferences on items that users are interested in. There are a lot of recommendation techniques that utilize the user profiles to personalize the recommendations.
- Combining Idiocentric and Collaborative Recommendation Idiocentric recommendation involves recommending items to users based only on the user's history. Collaborative recommendation involves recommending items to users based on user-user similarity or item-item similarity. Each of the methods give varied results and combining them is a challenging task.

In our approach to the solution, we are addressing the problems mentioned above. We verify the accuracy of our approach on the movielens dataset. We also compare the results that we've obtained against the results presented in one of the survey papers from the Machine Learning Course, Stanford University, Autumn 2012.

The solution that we propose is robust in the perspective of in-homogeneity in properties of items. That is, it is not mandatory that all the items have the same set of properties. We also perform unsupervised dimensionality reduction on the item dataset, relative to the user dataset. Considering the pattern in which the users have consumed the items, we also perform user profiling and determine the weight that the users inherently have towards attribute of the items. From the usage pattern, we deduce the extent of idiosyncrasy and use it to combine the results from idiocentric and collaborative recommendation.

Chapter 2

Problem Definition

We frequently stumble upon applications in e-commerce and the Internet that requires recommending items to users. From ages, many techniques have been proposed for recommending items to users, but as it involves a large dataset most of the techniques are costly in terms of space and time. Often, the dataset available are inhomogeneous in terms of item properties. If the item contains large number of properties i.e. a very high dimensionality, it becomes tedious for the system user to select features that have a significant impact factor. Good personalized recommendations demand an efficient and intuitive user profiling. So an effective way of solving this recommendation problem is of great importance.

Chapter 3

Literature Survey

Recommendation systems have gained popularity in web based systems since the appearance of papers on collaborative filtering in the 1990s [7, 11, 5].

- [7]explains the concept of collaborative filters. They introduce GroupLens as a system for collaborative filtering of netnews, to help people find articles they will like in the huge stream of available articles. They hypothesize that the users who have agreed on a certain aspect in the past will probably agree again.
- [11]describes a technique for making personalized recommendations to a user based on the similarities between the interest profile of that user and those of other users. They also test and compare four different algorithms for making recommendations using social information filtering.
- [5]presents an approach for collaborative recommendation where in the history of other users is used in the automation of a social method for informing choices to the user. Their results show that the communal history-of-use data can serve as a powerful resource for use in interfaces.
- [6]determines the potential predictive utility for Usenet news. They develop a specially modified news browser that accepts ratings and displays predictions on a 1-5 scale. They compute the predictions using collaborative filtering techniques and compare the results with non collaborative approaches.
- [8]describes several algorithms designed for collaborative filtering, including techniques based on correlation coefficients, vector-based similarity calculations, and statistical Bayesian methods. They compare the predictive accuracy of the various methods in a set of representative problem domains. Over the past decade, web systems have moved towards personalized recommendations for better user experience.
- [3]describes a tag-based system for personalized recommendation. They propose an approach which extends the basic similarity calculus with external factors such as tag popularity, tag representativeness and the affinity between user and tag.
- [1]investigates user modeling strategies for inferring personal interest profiles from social web interactions. They analyze individual micro-blogging activities on twitter and compare different strategies for creating user profiles based on the twitter messages.
- [10]presents a personalization algorithm for recommendation in folksonomies, which relies on hierarchical tag clusters.
- [9]explores and analyzes different item-based collaborative techniques. They look into different techniques for computing item-item similarities and various techniques to obtain recommendations from them. Significant developments in learning using graph data has led to recent advances in recommendation techniques.
- [2]presents a recommendation algorithm that includes different types of contextual information. They model the browsing process of a user on a movie database by taking random walks over the contextual graph.

- [4]models personalized tag recommendation as a ”query and ranking” problem. They also propose a novel graph-based ranking algorithm for interrelated multitype objects.

Chapter 4

Project Requirement Definition

4.1 Target Users

The primary target users are academic researchers and firms that use some form of recommendation in their product. The algorithm is open sourced and the users are free to contribute towards the development of a more efficient algorithm.

4.2 Software Requirements

4.2.1 Python

Python is a general-purpose, high-level programming language whose design philosophy emphasizes code readability. It is an expressive language which provides language constructs intended to enable clear programs on both a small and large scale with its own built-in memory management and good facilities for calling and cooperating with other programs. Python supports multiple programming paradigms, including object-oriented, imperative and functional programming styles. Python packages that we use in our implementation are:

- networkx - Used for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.
- pickle - pickle is the standard mechanism for the object serialization and de-serialization. It was developed as a pure python pickle module in its beta version. We use pickle to store graph objects.
- numpy - numpy is an extension to the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

4.2.2 Gephi

Gephi is an interactive visualization and exploration platform for all kinds of networks and complex systems, dynamic and hierarchical graphs. We use it to visualize the Item graph while displaying the output.

4.2.3 Pencil

Pencil is an open-source GUI prototyping tool that we used to create UI mockup.

4.2.4 Git Versioning System

In software development, Git is a distributed revision control and source code management (SCM) system with an emphasis on speed. Every Git working directory is a full-fledged repository with complete history and full revision tracking capabilities, not dependent on network access or a central server. We use it to effectively control our software development.

4.2.5 Qt

Qt is a cross-platform application framework that is widely used for developing application software with a graphical user interface (GUI) (in which cases Qt is classified as a widget toolkit), and also used for developing non-GUI programs such as command-line tools and consoles for servers. Qt uses standard C++ but makes extensive use of a special code generator (called the Meta Object Compiler, or moc) together with several macros to enrich the language. Qt can also be used in several other programming languages via language bindings. It runs on the major desktop platforms and some of the mobile platforms. It has extensive internationalization support. Non-GUI features include SQL database access, XML parsing, thread management, network support, and a unified cross-platform application programming interface (API) for file handling.

4.3 Hardware Requirements

This being partially a research project, we are coming up with an algorithmic solution to the recommendation problem. So the hardware requirements are minimal and basic. The execution demands a linux-based operating system. The exact configuration of the system is highly application dependent. Larger the dataset, more powerful the system should be.

4.4 Assumptions and Dependencies

As a user of our recommendation system, we assume that he/she has the knowledge about the following aspects:

- The user should possess the working knowledge of linux-based system.
- The user should know how to execute projects in Qt.
- The user is expected to know the abstract flow of the algorithm.
- The user is expected to know what a dataset item and a dataset user is.
- The user is expected to know what the properties of the items are.
- The user should have knowledge about the format of our input dataset, as specified in the input requirements.
- The user should know what the terminologies idiocentric and collaborative recommendations represent.
- The user should be able to interpret what the relative attributive importance and relative value importance are.
- The user should know what dimensionality reduction is.
- The user should have a basic knowledge in the concepts involved in similarity and clustering.
- If the user is keen on understanding how the recommendations were given, he/she should have knowledge about graphs, data-mining and machine learning concepts.

As a researcher or a developer who wishes to extend our system, we assume that he/she has the knowledge about the following aspects:

- All the assumptions specified for a normal user in the above sub-section hold good for a researcher/developer.
- The researcher is expected to have deep understanding about graphs, data-mining and machine learning concepts.
- The researcher should be aware of the previous research works carried out in this domain.
- The researcher should have a deep understanding about our algorithm.

- The researcher should have deep understanding about the concepts of similarity and clustering (overlapping clustering in specific).
- The researcher should understand the difficulties in unsupervised dimensionality reduction.
- The researcher should be able to identify the improvements in various sections of the algorithm.
- The developer is expected to be knowledgeable about the following technologies:
  - Python
  - Qt
  - Git
  - Linux Based Operating System
- The developer should be able to recognize and use the existing interfaces to plug-in custom algorithms.

4.5 Languages Used

- Python
- Qt for GUI Programming

Chapter 5

System Requirements Specification

5.1 Functional Requirements

There are minimal functional requirements for using this system.

- The UI is provided wherein there is a space to input the datasets.
- The input is a set of two JSON files where one is item dataset and the other is user dataset. For our ease we have defined the input item dataset JSON file to be of a particular format where the first line gives the information about the type of all attributes.

5.2 Non Functional Requirements

- This is a partially research based open source project. The project and its source code are licensed under GNU General Public License Version 3. The exact terms and conditions can be found at http://www.gnu.org/licenses/gpl.html.
- The source code can be found at https://github.com/vijeshm/recommendation-algorithm-fyp.
- The system is accessible and available to everyone without any discrimination among the users.
- The present version is not platform independent. It requires a linux based operating system and the exact configuration is use case dependent.

Chapter 6

Gantt Chart

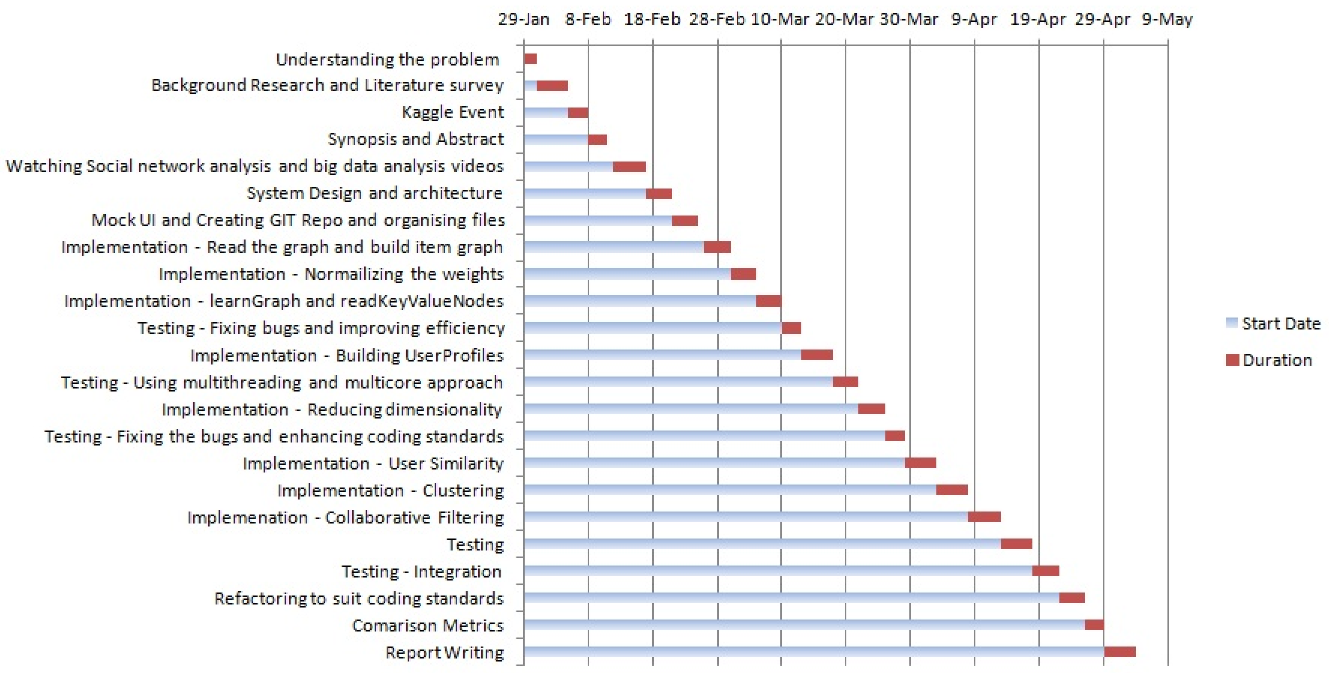


Figure 6.1: Gantt chart for the project

Figure 6.1 presents the Gantt Chart for the timeline of our project.

Chapter 7

System Design

7.1 Component Diagram

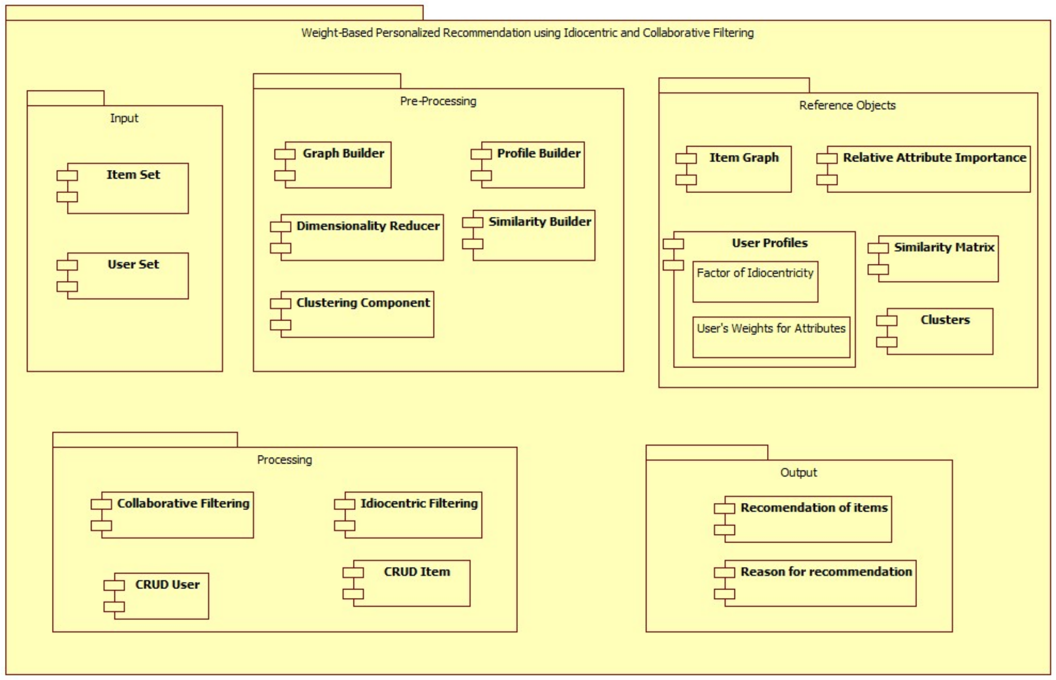


Figure 7.1: Component diagram for the project

Figure 7.1 shows different components or our recommendation system. There are four different components, viz. Input, Pre processing, Processing and Output. The details of each component are as shown in the above figure.

7.2 Activity Diagram

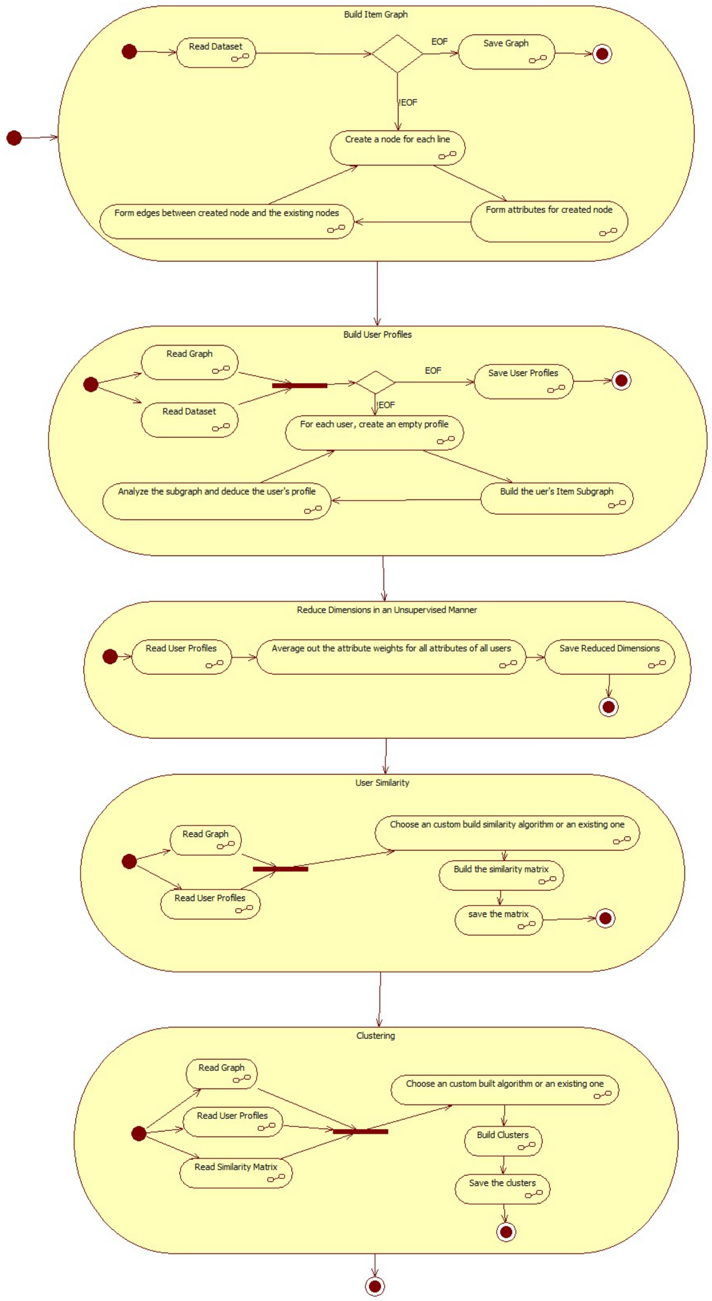

Build Item Graph
Read Dataset
EOF
Save Graph
!EOF
Create a node for each line
Form edges between created node and the existing nodes
Form attributes for created node
Build User Profiles
Read Graph
Read Dataset
EOF
Save User Profiles
!EOF
For each user, create an empty profile
Analyze the subgraph and deduce the user's profile
Build the uer's Item Subgraph
Reduce Dimensions in an Unsupervised Manner
Read User Profiles
Average out the attribute weights for all attributes of all users
Save Reduced Dimensions
User Similarity
Read Graph
Read User Profiles
Choose an custom build similarity algorithm or an existing one
Build the similarity matrix
save the matrix
Clustering
Read Graph
Read User Profiles
Read Similarity Matrix
Choose an custom built algorithm or an existing one
Build Clusters
Save the clusters

Figure 7.2: Activity diagram for the project

Col1	**Col2**	**Col3**
value 1		Sample embedded table:

Table 7.1: Sample Table

Chapter 8

Detailed Design

The following section describes the design and architecture of the system in detail.

8.1 Building Item Graph

As shown in the activity diagram, the flow starts with building the item graph from the item dataset. We build a graph with each node representing an item and each edge representing the common attributes between them. The common attributes that bind the two nodes with its value will be used as an attribute for the edge. Each item is associated with a set of properties. Each of those properties is mapped on to the graph as the properties of the node. Figure 8.1 illustrates the concept explained above.

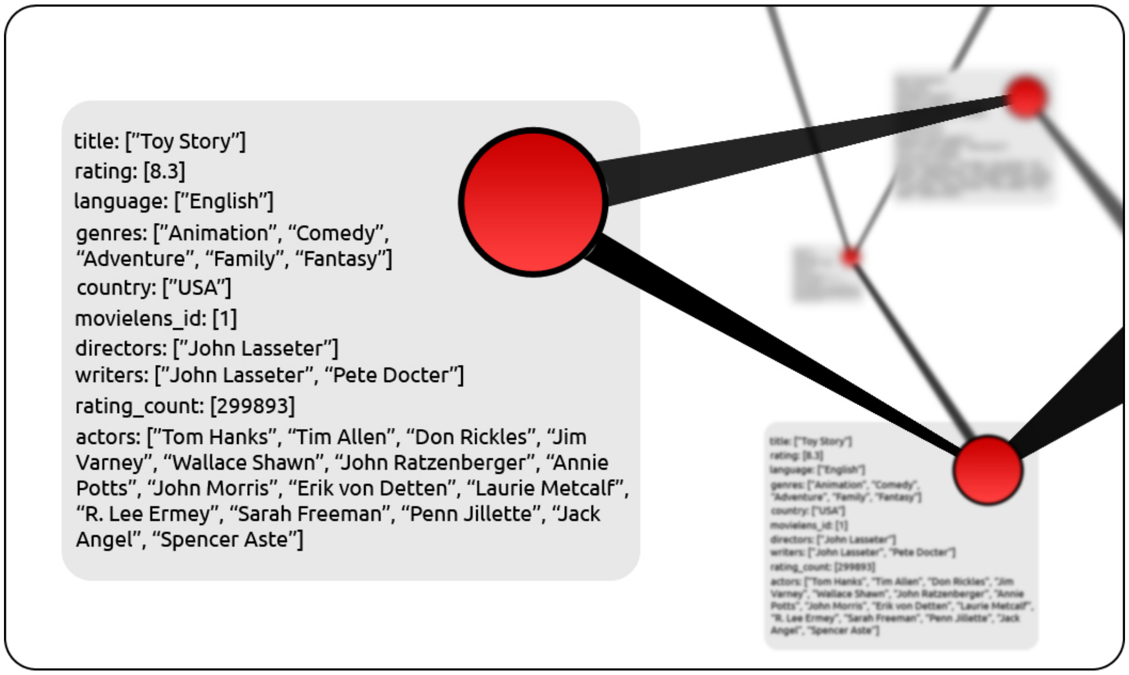

title: ["Toy Story"]
rating: [8.3]
language: ["English"]
genres: ["Animation", "Comedy", "Adventure", "Family", "Fantasy"]
country: ["USA"]
movielens_id: [1]
directors: ["John Lasseter"]
writers: ["John Lasseter", "Pete Docter"]
rating_count: [299893]
actors: ["Tom Hanks", "Tim Allen", "Don Rickles", "Jim Varney", "Wallace Shawn", "John Ratzenberger", "Annie Potts", "John Morris", "Erik von Detten", "Laurie Metcalf", "R. Lee Ermey", "Sarah Freeman", "Penn Jillette", "Jack Angel", "Spencer Aste"]

Figure 8.1: The structure of the graph

8.2 Building User Profiles

A user is an entity who consumes an item. In this section, we build a profile for each user. The user profile consists of characteristic parameters of the user that quantifies his behavior. In real world, many factors influence the decision of the user to consume a particular item.

These factors include the personal choices of the user and recommendation by other users. This scenario is analogous to a customer at a restaurant. The customer might place an order due to recommendations by his friends. His decision is also influenced by what he personally prefers to eat. Generally, the customer does not have equal preferences towards all properties of the food, such as sour, salt, hot and sweet. He has varied levels of liking towards various attributes. Also, the customer might choose to place the order according to his preference or others' recommendation, with a certain probability. We generalize and quantify these behaviors of the customer in order to deduce the characteristic parameters of the customer.

Each user profile is built by recognizing the relationship among the items that the user has consumed. This relationship is captured in the sub-graph of those items from the item graph. For each user a set of properties are deduced that convey the orientation of the user towards the various attributes of the items. This orientation is quantified in terms of relative attributive importance and relative value importance. This derived knowledge about the user is used to recommend items to that user in both idiocentric and collaborative techniques.

8.3 Unsupervised Dimensionality Reduction

In this section, we present an empirical approach to perform dimensionality reduction on the dataset. We observe the users' pattern in consuming the items and deduce the importance of an attribute. The importance of an attribute in the item dataset increases as number of users who give higher relative importance to that attribute increases. Hence, the importance of an attribute a in the item set I can be quantified as the mean relative importance of a, taken over all the users u belonging to U. In the previous section, we created a profile for each user. The characteristic parameter w consists of relative weights of attributes for u. The relative importance of an attribute a in the item set I can be computed as follows:

$$weight_a = \frac{\sum_{u_p \in U} w_{u_p}[a]}{Number\ of\ Users}$$

Figure 8.2: formula for relative attribute importance

A similar argument holds well in case of relative value importance.

$$weight_v = \frac{\sum_{a \in A} w_{u_p}[a][v]}{Number\ of\ Values}$$

Figure 8.3: formula for relative value importance

8.4 User Similarity

This section explains the collaborative filtering part of the solution. The techniques involved in Collaborative Filtering are based on analyzing a large amount of data on users' activities and predicting what a given user might like. The recommendations can be based on similarity between users or similarity between items. Our algorithm uses similarity between users to perform collaborative filtering. Some of the popular similarity measures are Euclidean Distance, Cosine Similarity, Pearson Correlation and Jaccard Similarity.

Jaccard similarity is a measure used for comparing the similarity of sample sets. Jaccard similarity between two sets A and B is mathematically defined as follows:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Figure 8.4: formula for jaccard similairty

Each user is associated with a set of items that he has consumed. The similarity between users can be computed just by applying the simple Jaccard similarity between the sets of items for a pair of users. But there is an intrinsic drawback associated with this approach. The approach treats all the items to be equivalent. It is ignorant towards the properties of the items themselves and the relations between them. To include these considerations, we modify the simple formula for Jaccard similarity.

The idea behind the modification is as follows. To compute the similarity between two users up and uq, we compute two induced subgraphs of G. One of them contains the intersecting items of the two users, and the other contains the union of items of the two users. From the two induced subgraphs, we make two aggregated lists of edge attributes, one for each subgraph. Note that the attributes in these lists can repeat. The similarity is calculated as a fraction. To compute the numerator, we iterate through the attributes in the aggregated list for item intersection subgraph. Counting the number of occurrences of these attributes would implicitly mean that we are considering all the attribute as equals. But, the users give varied levels of importance to the attributes. Hence, for each attribute in the aggregated list, we sum up the corresponding weights from the users up and uq and multiply the sum by the relative frequency of occurrence of the attribute in the aggregated list. We follow a similar procedure to compute the denominator, with the only difference being that we consider the item union subgraph.

$$Intrscn = [a \mid a \in edge\ attribute\ of\ subgraph(items\ of\ u_p \cap items\ of\ u_q, G)]$$

Figure 8.5: formula for intersection

$$Union = [a \mid a \in edge\ attribute\ of\ subgraph(items\ of\ u_p \cup items\ of\ u_q, G)]$$

Figure 8.6: formula for union

$$sim(u_p, u_q) = \frac{\sum(w_{u_p}[a] + w_{u_q}[a]) * relFreq(a, Intrscn)\ \forall\ a \in set(Intrscn))}{\sum(w_{u_p}[a] + w_{u_q}[a]) * relFreq(a, Union) \forall\ a \in set(Union))}$$

Figure 8.7: formula for modified jaccard similairty

8.5 Clustering

One of the most useful statistical analysis techniques for understanding the user behavior is clustering. This group behavior will help in recommending items that might be of interest to that cluster. The task of grouping users is carried out in such a way that the users belonging to the same group (referred to as a cluster) will have a coherent list of similarity values. These values, in turn, are calculated as explained in the previous section. Here, we use an overlapping clustering approach, where a given user may belong to multiple clusters. Overlapping clustering captures the user interests in vivid attributes.

In the first step, we find out the average similarity between all pairs of users. We then create a trivial cluster for each user and progressively add other users in accordance to the condition that the similarity between a base user and all other users is greater than the average value. On completing this step, we find the clusters exist in duplicates. The duplicates are then removed to obtain the final set of clusters.

8.6 Idiocentric Recommendation

The idiocentric recommendation algorithm works as follows. We initialize the similarity scores of all the items to zero. Given a user u, we analyze the list of items that u has previously consumed. If a user has consumed an item i, it is very likely that the user might be interested to consume the neighbors of i, but not to equal extents. The extent to which the user might be interested to consume a neighbor not only depends upon the similarity between the items, but also upon the weight that the user gives to the common features between the items. The common features between the items are captured as the property of the edge between them. Hence, we increment the score of each neighbor by the weight that the user gives to each property of the edge. After updating all the nodes, we create an associative array of items and their score. Note that the associative array does not contain the items that are already consumed by the user. We then normalize the scores by dividing all the scores by the maximum score.

8.7 Collaborative Filtering

Collaborative Filtering (CF) is a collection of techniques that are used in recommending items to users based on the ratings of other users. CF techniques are by far the most successful techniques in recommending items to users. The core idea behind CF techniques is the concept of similarity. The recommender systems generally use two forms of similarities: user-based similarities and item-based similarities. These techniques generally use a standard metric, such as Euclidean Distance, Cosine Similarity, Pearson Correlation or Jaccard Similarity. Once a similarity matrix is built, these systems use various algorithms to generate recommendations. In this project, we have used an extended version of Jaccard similarity. The algorithm that we have developed uses user-based similarity. Our algorithm begins by iterating through all the clusters. A cluster houses many users. Each user is associated with a list of items that he has consumed. For each user up in each cluster c, we increment the score of each item i consumed by up, by the similarity between u and up. Note that the clusters we obtained are overlapping. That is, a single user can belong to multiple clusters. If the given user u1 is very similar to another user u2, then u2 will appear in majority of the clusters where u1 is present. This implies that the score for the items consumed by u2 is naturally boosted, since u2 is presented in most of u1's clusters and the similarity between u1 and u2 is high.

8.8 A weighted combination for recommendation

The previous sections dealt with generating egocentric and collaborative recommendations for the user. Both the algorithms predict a rating for each item that a user has not consumed. Hybrid Recommendation involves combining the idiocentric and the collaborative scores list of hybrid scores, by means of weighted combination of corresponding items from both the lists. Since we are merging the lists that signify idiocentric behavior and group behavior, we choose alpha as the weight. In one of the previous sections, we had defined alpha as the probability with which the user up decides upon an item purely based on his preferences. alpha quantifies how idiocentric a user is. Hence, assign a weight alpha to idiocentric ratings and (1 - alpha) to collaborative ratings.

We propose two methods by which the lists can be combined:

- Rank based weighted average:
  In this approach, we take a weighted average of ranks of each item and sort the items in the decreasing order of their composite ranks. As mentioned earlier, we assign a weight alpha to idiocentric rating and (1 - alpha) to collaborative rating. The following formula mathematically describes the operation:

$$compositeRank_i = \alpha_{u_p} * rank(i, scores\_ego) + (1 - \alpha_{u_p}) * rank(i, scores\_collab)$$

Figure 8.8: composite rank-based

- Score based weighted average:
  In this approach, we take a weighted average of ratings of each item and sort the items

in the decreasing order of their composite ratings. As mentioned earlier, we assign a weight alpha to idiocentric ratings and (1 - alpha) to collaborative ratings. The following formula mathematically describes the operation:

$$compositeScore_i = \alpha_{u_p} * scores\_ego[i] + (1 - \alpha_{u_p}) * scores\_collab[i]$$

Figure 8.9: composite score-based

Rank based weighted average method can be used to provide recommendations to the user in the order of their ranks. But predicting the rating that the user might give to an item tends to be relatively hard. Consequently, evaluating the accuracy of our algorithm also tends to be relatively hard. Hence, in our experiments, we use the score based weighted average method to predict the rating that a user might give to a previously unseen item.

Chapter 9

Hurdles: Design Decisions and Optimization

9.1 Git Versioning System

In software development, Git is a distributed revision control and source code management (SCM) system with an emphasis on speed. In our earlier projects, we had used Git in a minimalistic manner, where we used only local repositories to maintain our code. In this project we had to learn more about handling remote repositories on github. We encountered many problems in branching and merging of the code as we had to simultaneously handle many datasets. Our code, commits and branches can be found at https://github.com/vijeshm/recommendation-algorithm-fyp.

9.2 MockUp UI

We had two tool choices, MockUp and Pencil for creating the UI mockup on Ubuntu. We initially built the UI mockup using Mockup tool but it wasn't sufficiently powerful for our requirements. Hence we modified the mockup using Pencil tool.

9.3 Choosing Language

In order to quickly and easily implement our algorithm, we had to choose a language designed for code readability, simplicity and robustness. Hence we had three choices, Python, Jython, Cython and R. Finally we decided to use Python as our implementation language for the reasons and tradeoffs mentioned below:

- Python v/s R: The R is widely used among statisticians and data-miners for data analysis. In contrast to Python, R has less support in terms of extensibility, libraries and communities. More insights about other tradeoffs can be found at pandas.python.org.
- Python v/s Jython: We had initially thought of using a database to store the reference structures that we generated. Jython was primarily considered as an option as it could be linked with the databases easily. Since Jython uses compiled codes, it was expected to be relatively faster. But we chose against going with databases, as discussed in next section.

- Python v/s Cython: Using Cython would have made the execution faster, but it takes lot of time to code and optimize it. Due to project timeline, we could not implement it. Later we found out that integrating the Cython code with the UI code was a huge task by itself.

9.4 Databases

We had initially thought of using a graph database to store the dataset and the reference structures. Referring to http://en.wikipedia.org/wiki/Graph
_database, HyperGraph and ArangoDb were the most suited ones to our application. But we decided to against the databases, since the read and write time for a database is far greater than that of an in-memory object. The available machines today are capable of managing huge in-memory objects.

9.5 Genericity

Generally, the development that takes place for these kind of problems are dataset specific. During the development, we identified the parts which can be generalized and abstracted them to work for any generic dataset. In order to bring this about, we put certain constraints on the input dataset formats, such as mandatory "id" field for each item in the dataset. The details of more such constraints can be found in Input requirements section.

9.6 32-bit Addressable Woes

During the initial stages of development, we had used a 32-bit addressable OS to execute our code. Apparently the size of our graph object exceeded 4GB and there was a Memory Error exception. This motivated us to optimize the algorithm and data-structures. In the course of 3 days, we realized that we were using a 32-bit addressable OS. We solved the problem by porting our code to a 64-bit OS.

9.7 Graph

9.7.1 Writing huge memory objects as a pickle file

The pickle module takes a lot of memory to stringify objects. We tried to stringify the huge graph objects on an 8GB RAM machine but the memory was still insufficient and the graph object by itself took a lot of space. The solution to the problem was to write only the required information about the graph manually onto a file.

9.7.2 Graph Formats

A graph object can be stored on a disk in many formats such as:

- GEXF
- GDF
- GML
- GraphML
- Pajek NET
- GraphViz DOT
- CSV

We tried to export the graph objects onto these formats, but we encountered the same problems as we did in the pickle export.

9.7.3 Graph Storage

Due to the problems mentioned above, we decided to export the graph as an edgelist along with its properties. Each line in the file is of the form:
node1 node2 properties
This file consumed 1.5 GB of space on the disk. On enumeration of the attributes and values of the edge properties, we were able to reduce the file size to 982 MB. But this still proved to be very large on converting it back to an in memory object. To overcome these problems, we had to come up with the new and innovative solution. We exploited the graph structure and used the technique of reverse indexing to get the size down to

2.0 MB. The reverse indexed structure is as follows.
insert keyValueNode structure here*

9.8 Categorization of non-categorical Data

Attributes that are of integers, floating point and time data types fall under numerical or non-categorical data. But our algorithm is designed to work for categorical data. Hence there is a need to convert this data into categories. We referred "Data Clustering Theory, Algorithms and Applications by Guojun Gan, Chaoqun Ma, Jianhoug Wu" in order to achieve this. The method that they propose uses K-Means clustering to categorize the numerical data iteratively. This proved to be a time costly operation. Hence we came up with time efficient clustering solution that serves our needs.

9.9 Parallelization of Algorithms

The algorithm was designed keeping in mind the possibility of parallelization. Many parts of the approach were inherently parallelizable. We came up with 3 solutions for parallelization of building user profiles, as listed below:

For each user, a new process is spawned for building the corresponding user profile. This induced a highly coarse granularity between the processes. In turn, there was a drastic increase in the CPU usage, RAM and Swap space. Figure 9.1 illustrates the point.
We then used a limited pool of 4 processes (using the Pool module from multiprocessing), and assigned jobs to these 4 processes. This method couldn't be used to recursively parallelize the algorithm. The problem with this technique was sharing the data. When we passed a reference to a data structure, the data present in the object could be read. But when we write the data back, the action was not being reflected in the parent process. Hence, this approach wasn't suitable either. The memory consumed decreased and the CPU usage was reduced considerably (approximately 2GB and 40
% usage on 4 cores).

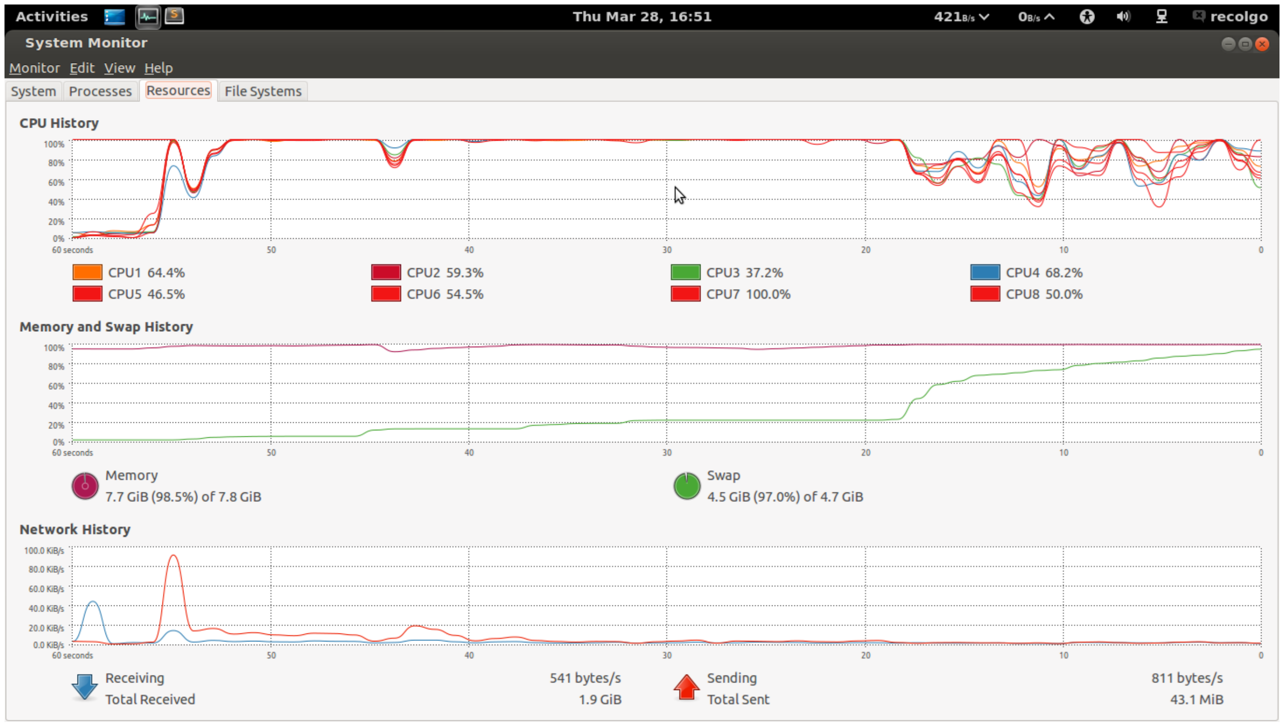


Figure 9.1: state of the system during execution

Execution time for building basic user profiles:
Linear execution: 11s
multithreaded: 44s
i.e. a factor of 0.25

Another solution was to create multiple threads. This would allow us to share data easily, and doesn't create much overhead in terms of inter-process communication. The procedures can be recursively parallelized. But this method also introduced a coarse granularity.

Considering all the above mentioned factors, we decided to go with the linear execution of the algorithm, though it can be parallelized.

Chapter 10

Implementation

The following section describes the implementation details of the design we have come up with.

10.1 Item Graph

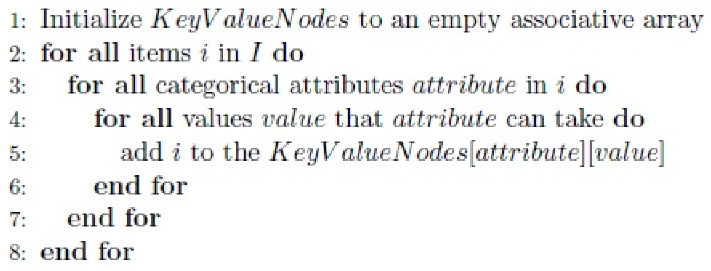
In our initial phase, we had built a graph object of all the items as nodes with the common attributes between them as connecting edges. But this method proved costly in terms of space. The whole graph object used to get stored in memory and hence to make it more memory efficient we thought of changing the way of data representation.

The data structure we came up with to represent the item dataset was of key-value-node pair. By reverse indexing the data set, we observed that the amount of space it occupied was lesser. The key here mean the attribute and the value is the value of the attribute. Node describes the items that are associated with that particular attribute-value pair.

For example, say we have item1 and item2 having common attributes a2 with value v1 and similarly item3 and item4 with a2 and v3 respectively. In our key-value-node pair, this can be represented as: *insert keyValueNodes structure here*

The item dataset consists of both categorical and non-categorical data. The categorical data will be either string or boolean type (Eg: Director and Like, respectively). Non-categorical data includes all the numerical attributes. (Eg: rating, year, timestamp). The following algorithms explain the approach for the two types of data:

- For categorical data:



```
Initialize $KeyValueNodes$ to an empty associative array
for all items $i$ in $I$ do
    for all categorical attributes $attribute$ in $i$ do
        for all values $value$ that $attribute$ can take do
            add $i$ to the $KeyValueNodes[attribute][value]$
        end for
    end for
end for
```

Figure 10.1: building keyValueNode structure for categorical data

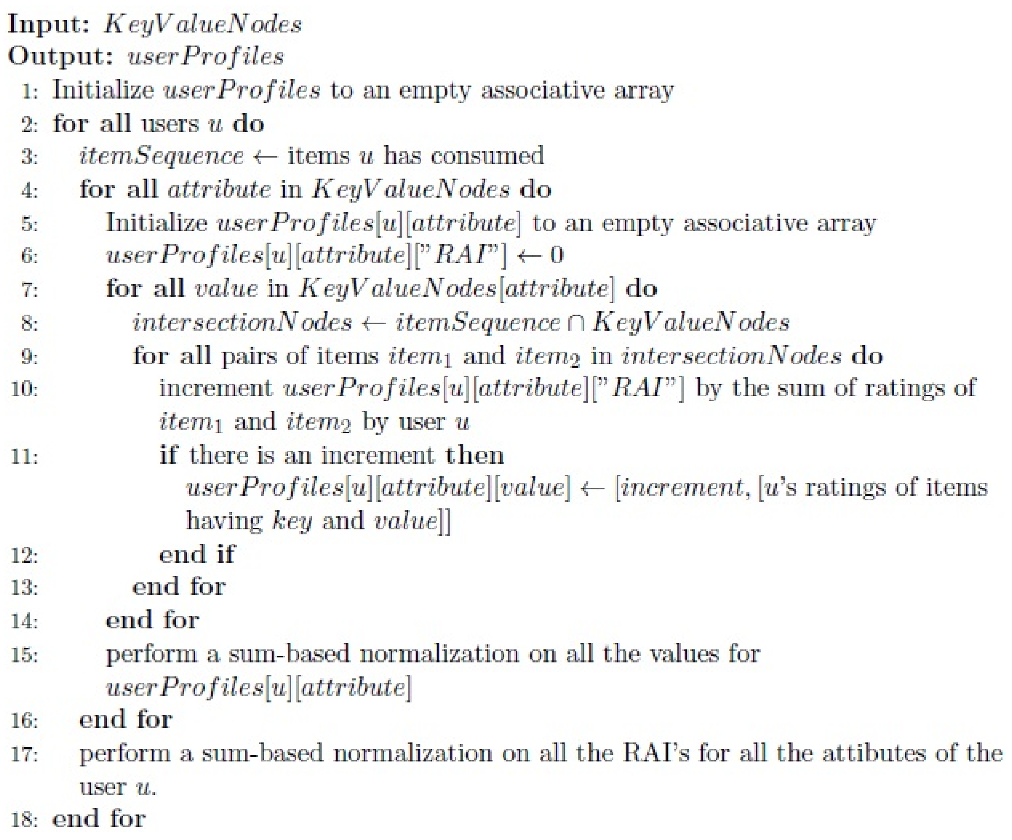
- For non-categoricla data:

```
Initialize $D$ to an empty associative array
for all non-categorical attributes $attribute$ in the dataset do
    set D[$attribute$] to empty list
end for
for all item $i$ in I do
    for all non-categorical attributes $attribute$ of $i$ do
        D[attribute] $\leftarrow$ D[attribute] $\cup$ values taken by $attribute$ for $i$
    end for
end for
$clusters \leftarrow$ empty associative array
for all $attribute$ in D do
    $cutpoints \leftarrow$ []
    sort D[$attribute$]
    $differences \leftarrow$ list of differences between consecutive values of D[$attribute$]
    $avrg \leftarrow$ average of $differences$
    populate $cutpoints$ with the indices for which the difference is greater than $avrg$
    populate $clusters$[$attribute$] with list of values that fall between the consecutive
    indices
end for
for all item $i$ in I do
    for all non-categorical attribute $attribute$ of $i$ do
        clust $\leftarrow$ cluster that the value of $attribute$ belongs to
        add $i$ to the $KeyValueNodes$[$attribute$][$clust$]
    end for
end for
```

Figure 10.2: building keyValueNode structure for categorical data

10.2 User Profile

Building user profiles from the user dataset is a major task as it eventually helps in idiocentric recommendation. User profiles are built by the data obtained from the user dataset. It takes into account the orientation of the user towards the value of a particular attribute as well as the attribute itself. The relative attributive importance and the relative value importance together define the attitude of the user towards the attributes in the item dataset. The personalized user profiles will be built using the key-value-node reference structure generated on building the item graph. The algorithm starts with initializing an empty associative array for each user. The relative attributive importance of each attribute is computed eventually for all the users. In computing the profiles for each user, we consider the ratings of items in the intersection between the user's consumption list and the list of key-value-nodes in the way illustrated by the following algorithm.



```
Input: $KeyValueNodes$
Output: $userProfiles$
 1: Initialize $userProfiles$ to an empty associative array
 2: for all users $u$ do
 3:    $itemSequence \leftarrow$ items $u$ has consumed
 4:    for all $attribute$ in $KeyValueNodes$ do
 5:       Initialize $userProfiles[u][attribute]$ to an empty associative array
 6:       $userProfiles[u][attribute]["RAI"] \leftarrow 0$
 7:       for all $value$ in $KeyValueNodes[attribute]$ do
 8:          $intersectionNodes \leftarrow itemSequence \cap KeyValueNodes$
 9:          for all pairs of items $item_1$ and $item_2$ in $intersectionNodes$ do
10:             increment $userProfiles[u][attribute]["RAI"]$ by the sum of ratings of
                $item_1$ and $item_2$ by user $u$
11:             if there is an increment then
                   $userProfiles[u][attribute][value] \leftarrow [increment,$ [$u$'s ratings of items
                   having $key$ and $value$]]
12:             end if
13:          end for
14:       end for
15:       perform a sum-based normalization on all the values for
          $userProfiles[u][attribute]$
16:    end for
17:    perform a sum-based normalization on all the RAI's for all the attibutes of the
       user $u$.
18: end for
```

Figure 10.3: Algorithm to build userProfile

10.3 Reduce Dimensionality

The reduction in the dimensionality of the dataset is performed for both attributes and values. Attribute dimensionality reduction is the relative importance of an attribute taken over all the users. In other terms it defines the mean relative importance of an attribute taken over all the users. In the same manner, value dimensionality reduction is also computed, which quantifies the mean relative importance of a value of an attribute taken over all the users. The following algorithms describe the two concepts in a programmatic manner.

10.4 User Similarity

To compute the similarity between two users up and uq, we compute two induced sub-graphs of G. One of them contains the intersecting items of the two users, and the other contains the

```
Input: $userProfiles$
Output: $reducedAttrWeights$
 1: Initialize $reducedAttrWeights$ to an empty associative array
 2: for all $profile$ in $userProfiles$ do
 3:   for all $attribute$ in $profile$ do
 4:     $reducedAttrWeights[attribute] += userProfiles[profile]["weights"][attribute]$
 5:   end for
 6: end for
 7: for all $attribute$ in $reducedAttrWeights$ do
 8:   $reducedAttrWeights[attribute] = \frac{reducedAttrWeights[attribute]}{numberOfUsers}$
 9: end for
```

Figure 10.4: Algorithm to reduce dimensionality of attributes

```
Input: $userProfiles$
Output: $reducedValueWeights$
 1: Initialize $reducedValueWeights$ to an empty associative array
 2: for all $profile$ in $userProfiles$ do
 3:   for all $attribute$ in $profile$ do
 4:     for all $value$ in $profile[attribute]$ do
 5:       $reducedValueWeights[attribute][value] += userProfiles[profile]["weights"][attribute][value]$
 6:     end for
 7:   end for
 8: end for
 9: for all $attribute$ in $reducedValueWeights$ do
10:   for all $value$ in $reducedValueWeights[attribute]$ do
11:     $reducedValueWeights[attribute][value] = \frac{reducedValueWeights[attribute][value]}{numberOfUsers}$
12:   end for
13: end for
```

Figure 10.5: Algorithm to reduce dimensionality of values

union of items of the two users. From the two induced sub-graphs, we make two aggregated lists of edge attributes, one for each sub-graph. Note that the attributes in these lists can repeat. The similarity is calculated as a fraction. To compute the numerator, we iterate through the attributes in the aggregated list for item intersection sub-graph. Counting the number of occurrences of these attributes would implicitly mean that we are considering all the attribute as equals. But, the users give varied levels of importance to the attributes. Hence, for each attribute in the aggregated list, we sum up the corresponding weights from the users up and uq and multiply the sum by the relative frequency of occurrence of the attribute in the aggregated list. We follow a similar procedure to compute the denominator, with the only difference being that we consider the item union sub-graph. The algorithm below shows how it is implemented:

```
Input: $userSequence$, $keyValueNodes$, $userProfiles$
Output: $userSimilarityMatrix$
 1: Initialize $userSimilarityMatrix$ to an empty 2d matrix of size $numberOfUsers$ x
    $numberOfUsers$
 2: for all $attribute$ in $KeyValueNodes$ do
 3:   for all $value$ in $KeyValueNodes[attribute]$ do
 4:     Initialize $ItemPairs$ to list of combination of Items taking 2 at a time.
 5:     for all $item1$ and $item2$ in $ItemPairs$ do
 6:       $userList1 \leftarrow$ list of users who have consumed item1
 7:       $userList2 \leftarrow$ list of users who have consumed item2
 8:       $usersIntersectionSet \leftarrow userList1 \cap userList2$
 9:       $usersUnionSet \leftarrow userList1 \cup userList2$
10:       $userPairsIntersectionSet \leftarrow$ list of combination of $usersIntersectionSet$
          taking 2 at a time.
11:       $userPairsUnionSet \leftarrow$ list of combination of $usersUnionSet$ taking 2 at a
          time.
12:       for all $user1$ and $user2$ in $userPairsIntersectionSet$ do
13:         $score \leftarrow userProfiles[\mathrm{user1}]["weights"][attribute] +$
            $userProfiles[\mathrm{user2}]["weights"][attribute]$
14:         $userSimilarityMatrix[\mathrm{user1}][\mathrm{user2}]['numerator'] += score$
15:       end for
16:       for all $user1$ and $user2$ in $userPairsUnionSetSet$ do
17:         $score \leftarrow userProfiles[\mathrm{user1}]["weights"][attribute] +$
            $userProfiles[\mathrm{user2}]["weights"][attribute]$
18:         $userSimilarityMatrix[\mathrm{user1}][\mathrm{user2}]['denominator'] += score$
19:       end for
20:     end for
21:   end for
22: end for
```

Figure 10.6: Algorithm to compute user similarity

10.5 Idiocentric Recommendation

In this section, we predict the ratings that a user would give to an item considering idiocentric behavior. userWeights is an associative array that contains the weights given by that user for different attributes as well as values taken by that attribute. itemList is the list of items for which rating has to be predicted. The algorithm computes the scoreIdio for each item in the itemList using the userWeights associated with the attribute and their corresponding values of the item. Finally, to predict the rating of a given item, we take the dot product of score and ratings and divide it by sum of ratings in order to normalize. The algorithm is as follows:

```
Input: $user$, $userWeights$, $itemList$
Output: $recommendedItemsIdeo$
1: Initialize $recommendedItemsIdeo$ to an empty list
2: Initialize $scoreIdeo$ to an associative array
3: for all $item$ in $itemList$ do
4:    for all $attribute$ in $itemList[item]$ do
5:       for all $value$ in $itemList[item][attribute]$ do
6:          $weight \leftarrow \left(\frac{1}{userWeights[attribute]*userWeights[value]}\right)^2$
7:          append weight to $scoreIdeo[item]['weight']$ list
8:          $avgRating \leftarrow$ average of ratings given by the user for all the items he has
            consumed with $attribute$, $value$ combination
9:          append $avgRating$ to $scoreIdeo[item]['rating']$ list
10:      end for
11:   end for
12: end for
13: Initialize $predictedRatingIdeo$ to an associative array
14: for all $item$ in $itemList$ do
15:    $predictedRatingIdeo[item] \leftarrow \frac{scoreIdeo[item]['weight'] \bullet scoreIdeo[item]['rating']}{\sum scoreIdeo[item]['weight']}$
16: end for
17: $recommendedItemsIdeo \leftarrow$ sort the items according to $predictedRatingIdeo$ in
    descending order
```

Figure 10.7: Algorithm to produce idiocentric recommendation

10.6 Clustering

In the clustering algorithm, we cluster the users based on interest towards a particular attribute. Each user may belong to more than one cluster. i.e, we are performing an overlapping clustering technique. The method is as follows. The first step involves calculating averageSimilarity between all the users in order to set the threshold. This is done by averaging all the values in the userSimlarityMatrix computed in previous step. In the second step, a pair of users is combined to form one cluster if their similarity value is greater or equal to threshold. The algorithm is as follows:

```
Input: $userSimilarityMatrix$
Output: $userClusters$
 1: Initialize $userClusters$ to an empty list
 2: Initialize $userClustersDict$ to an empty associative array.
 3: $userPairs \leftarrow$ list of combination of $usersLists$ taking 2 at a time.
 4: $averageSimilarity \leftarrow$ average of all the similarity values in $userSimilarityMatrix$
 5: for all $user1$ and $user2$ in $userPairs$ do
 6:   if $userSimilarityMatrix[user1][user2] \geq averageSimilarity$ then
 7:     add $user2$ in $userClustersDict[user1]$ list
 8:   end if
 9: end for
10: for all $user$ in $userClustersDict$ do
11:   add $user$ to $userClustersDict[user]$ list
12:   add $userClustersDict[user]$ to $userClusters$ list
13: end for
14: remove duplicate $cluster$ if any present in $userClusters$
```

Figure 10.8: Algorithm to cluster the users

10.7 Collaborative Filtering

Collaborative filtering determines the rating a user might give to an item, considering the ratings given by other users who are similar to user under consideration. The algorithm takes userSimilarityMatrix, clusters, itemList as input. itemList is the list of items for which rating has to be predicted. For each item, collaborative filtering algorithm builds a vector of ratings (given by other users) and a vector of similarity values between the given user and other users. Finally, collaborative rating is predicted by taking a dot product of these vectors and normalizing by sum. The algorithm is as follows:

```
Input: $user$, $userSimilarityMatrix$, $clusters$, $itemList$
Output: $recommendedItemsColl$
 1: Initialize $recommendedItemsColl$ to an empty list
 2: Initialize $scoreColl$ to an associative array
 3: for all $cluster$ in $clusters$ do
 4:   if user $\in$ $cluster$ then
 5:     for all $otherUser$ in $cluster$ do
 6:       for all $item$ in $itemList$ do
 7:         if $otherUser$ has consumed the $item$ then
 8:           $rating \leftarrow$ rating given the $otherUser$ for that item.
 9:           append $rating$ to $scoreColl[item]['rating']$ list
10:           $similarity \leftarrow userSimilarityMatrix[user][otherUser]$
11:           append $similarity$ to $scoreColl[item]['similarity']$ list
12:         end if
13:       end for
14:     end for
15:   end if
16: end for
17: Initialize $predictedRatingColl$ to an associative array
18: for all $item$ in $itemList$ do
19:   $predictedRatingColl[item] \leftarrow \frac{scoreColl[item]['rating'] \bullet score[item]['similarity']}{\sum score[item]['similarity']}$
20: end for
21: $recommendedItemsColl \leftarrow$ sort the items according to $predictedRatingColl$ in
    descending order
```

Figure 10.9: collaborative filtering algorithm

Chapter 11

and so on...

so forth...