# 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:

### 1.1 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.

### 1.2 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.

## 1.3 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.

## 1.4 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:

1. The user should possess the working knowledge of linux-based system.
2. The user should know how to execute projects in Qt.
3. The user is expected to know the abstract flow of the algorithm.
4. The user is expected to know what a dataset item and a dataset user is.
5. The user is expected to know what the properties of the items are.
6. The user should have knowledge about the format of our input dataset, as specified in the input requirements.
7. The user should know what the terminologies idiocentric and collaborative recommendations represent.
8. The user should be able to interpret what the relative attributive importance and relative value importance are.
9. The user should know what dimensionality reduction is.
10. The user should have a basic knowledge in the concepts involved in similarity and clustering.
11. 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:

1. All the assumptions specified for a normal user in the above sub-section hold good for a researcher/developer.
2. The researcher is expected to have deep understanding about graphs, data-mining and machine learning concepts.

3. The researcher should be aware of the previous research works carried out in this domain.
4. The researcher should have a deep understanding about our algorithm.
5. The researcher should have deep understanding about the concepts of similarity and clustering (overlapping clustering in specific).
6. The researcher should understand the difficulties in unsupervised dimensionality reduction.
7. The researcher should be able to identify the improvements in various sections of the algorithm.
8. The developer is expected to be knowledgeable about the following technologies:
    a. Python
    b. Qt
    c. Git
    d. Linux based operating system
9. The developer should be able to recognize and use the existing interfaces to plug-in custom algorithms.

## 4.5 Languages Used:

1. Python
2. Qt for GUI Programming

# Chapter 5

# System Requirements Specification

## 5.1 Functional Requirements:

There are minimal functional requirements for using this system.

5.1.1 The UI is provided wherein there is a space to input the datasets.

5.1.2 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:

5.2.1 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.

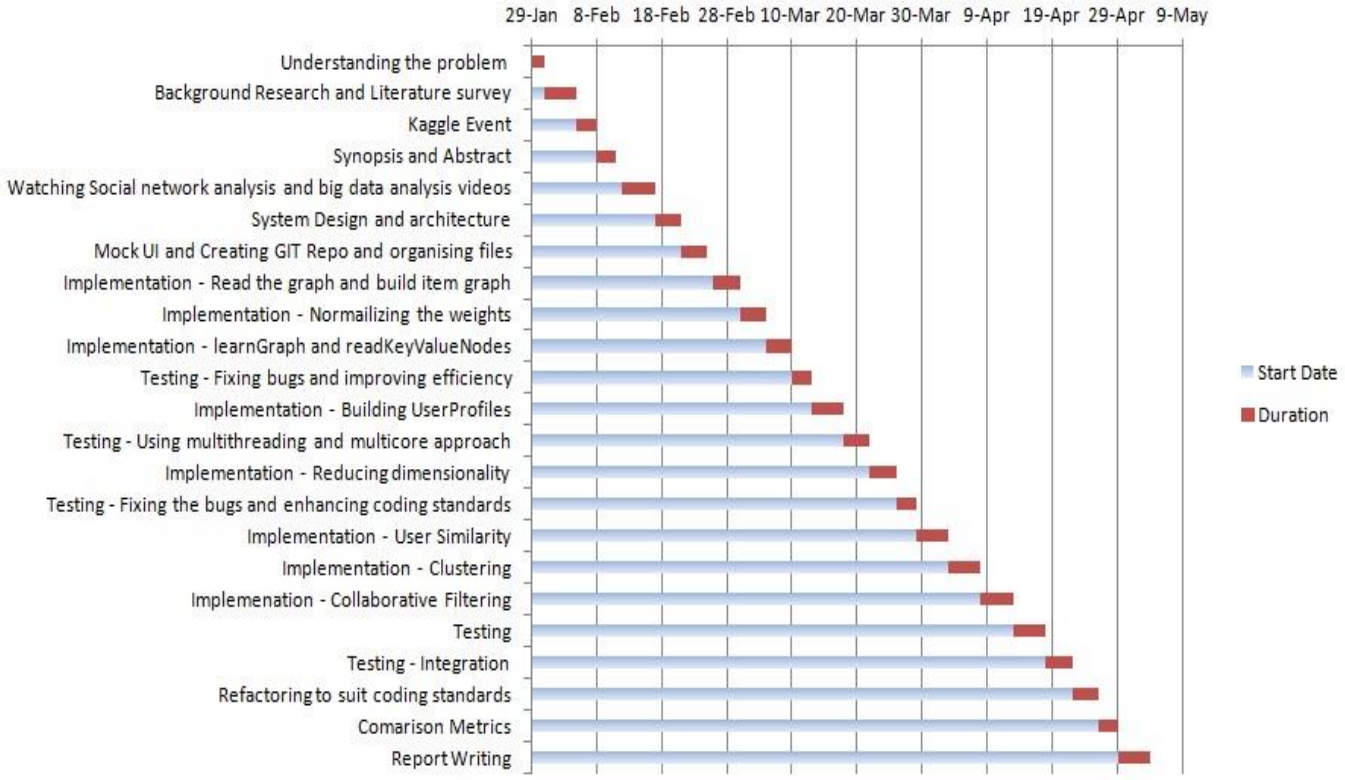
5.2.2 The source code can be found at https://github.com/vijeshm/recommendation-algorithm-fyp.

5.2.3 The system is accessible and available to everyone without any discrimination among the users.

5.2.4 The present version is not platform independent. It requires a linux based operating system and the exact configuration is use case dependent.

# Chapter 6

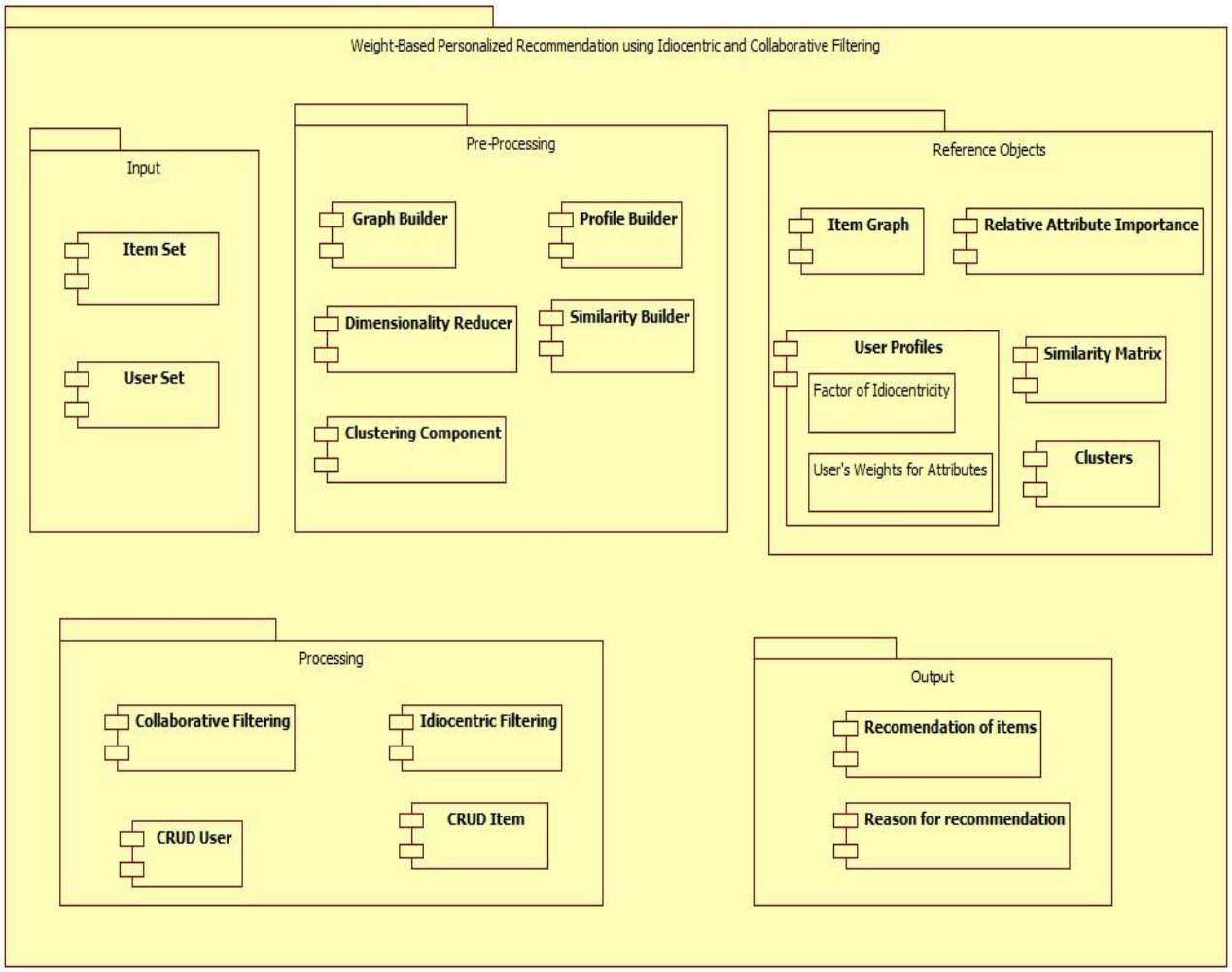
# Gantt Chart


29-Jan
8-Feb
18-Feb
28-Feb
10-Mar
20-Mar
30-Mar
9-Apr
19-Apr
29-Apr
9-May
Understanding the problem
Background Research and Literature survey
Kaggle Event
Synopsis and Abstract
Watching Social network analysis and big data analysis videos
System Design and architecture
Mock UI and Creating GIT Repo and organising files
Implementation - Read the graph and build item graph
Implementation - Normailizing the weights
Implementation - learnGraph and readKeyValueNodes
Testing - Fixing bugs and improving efficiency
Implementation - Building UserProfiles
Testing - Using multithreading and multicore approach
Implementation - Reducing dimensionality
Testing - Fixing the bugs and enhancing coding standards
Implementation - User Similarity
Implementation - Clustering
Implemenation - Collaborative Filtering
Testing
Testing - Integration
Refactoring to suit coding standards
Comarison Metrics
Report Writing
Start Date
Duration

# Chapter 7

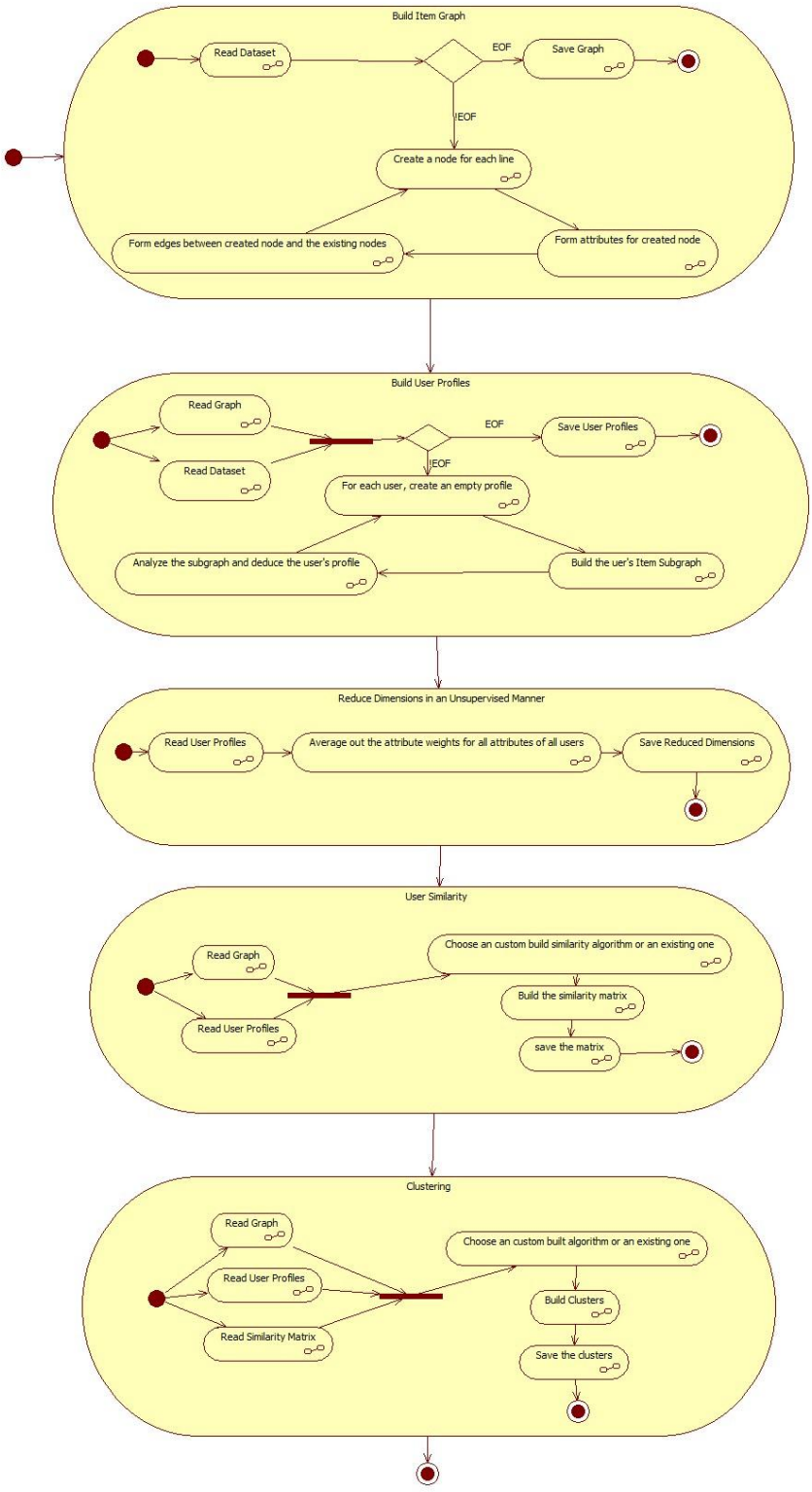
# System Design

## 7.1 Component Diagram:




The above component diagram 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:



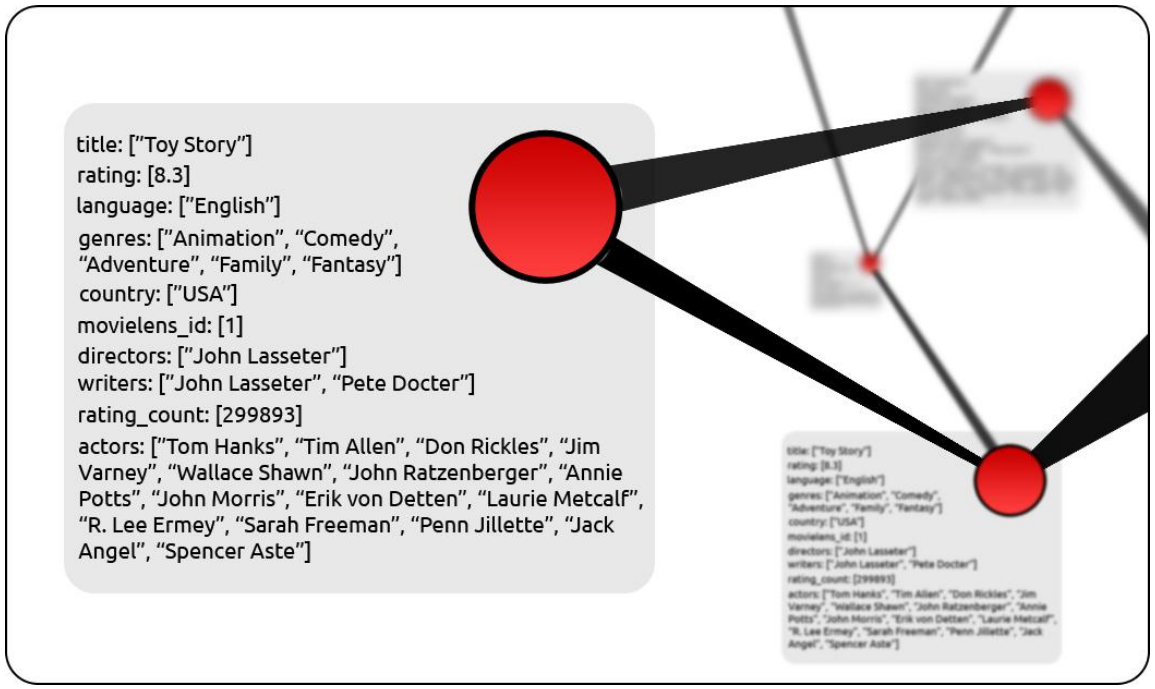
# Chapter 8

# 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. The following figure illustrates the concept explained above.




## 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:

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

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

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

## 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:

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

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 $u_p$ and $u_q$, 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 $u_p$ and $u_q$ 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.

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

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

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

## 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 $u_p$ in each cluster *c*, we increment the score of each item *i* consumed by $u_p$, by the similarity between *u* and $u_p$. Note that the clusters we obtained are overlapping. That is, a single user can belong to multiple clusters. If the given user $u_1$ is very similar to another user $u_2$, then $u_2$ will appear in majority of the clusters where $u_1$ is present. This implies that the score for the items consumed by $u_2$ is naturally boosted, since $u_2$ is presented in most of $u_1$'s clusters and the similarity between $u_1$ and $u_2$ 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:

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

- 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:

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

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.

```
keyValueNodes = { "attribute-1":{
                                    "value-1" : [ item-1, item-2, ...],
                                    "value-2" : [ item-2, item-3, ...],
                                    ...
                                    },
                  "attribute-2":    ...
                  ...
                   }
```

## 9.8 Categorization of non-categorical Data

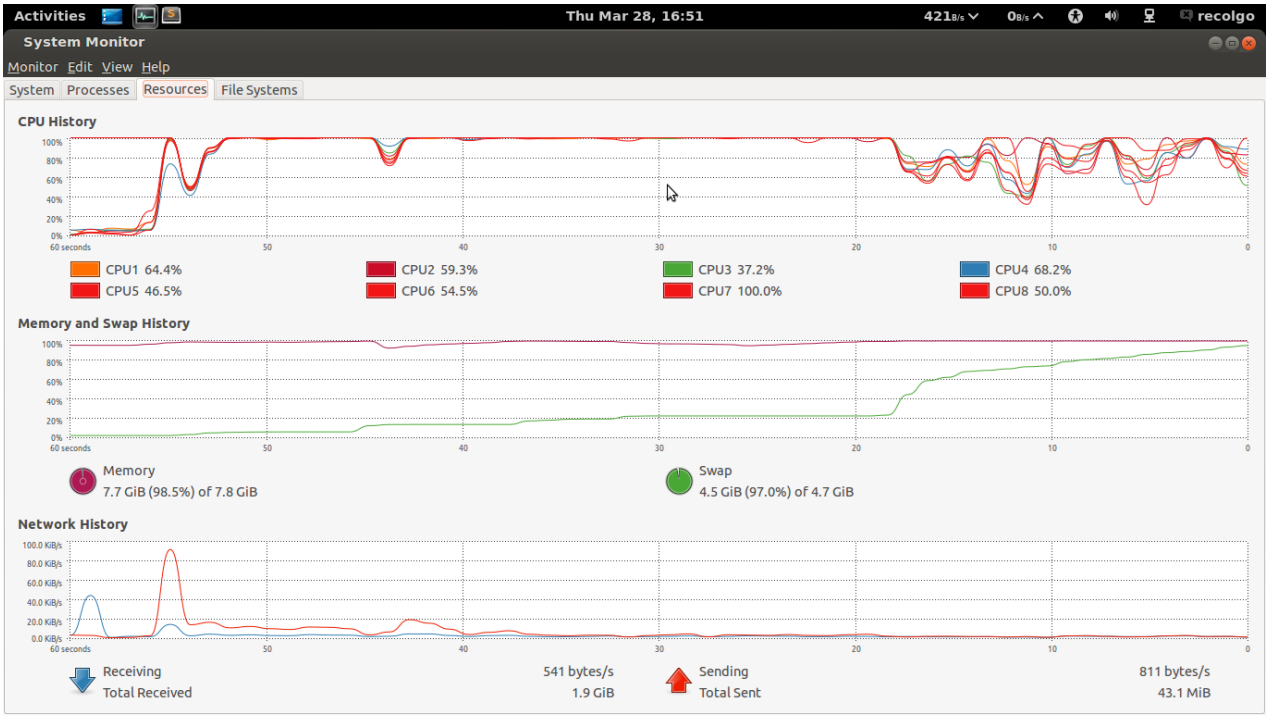
Attributes that are of integers, floating point and time data types fall under numerical or non-categorical data. But our algorithm is designed to work for categorical data. Hence there is a need to convert this data into categories. We referred *"Data Clustering Theory, Algorithms and Applications by 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. The following figure 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).

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:

```
{
        "a2":  {
                v1 : [item1, iem2],
                v3 : [item3, item4]
}
```

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
```

- For Non-Categorical 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] ← D[attribute] ∪ values taken by attribute for i
    end for
end for
clusters ← empty associative array
for all attribute in D do
    cutpoints ← []
    sort D[attribute]
    differences ← list of differences between consecutive values of D[attribute]
    avrg ← 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 ← cluster that the value of attribute belongs to
        add i to the KeyValueNodes[attribute][clust]
    end for
end for
```

## 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**

## 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

**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**

**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**

## 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 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:** $userSimilarityMatrix$

Initialize $userSimilarityMatrix$ to an empty 2d matrix of size $numberOfUsers$ x $numberOfUsers$
**for all** $attribute$ in $KeyValueNodes$ **do**
  **for all** $value$ in $KeyValueNodes[attribute]$ **do**
    Initialize $ItemPairs$ to list of combination of Items taking 2 at a time.
    **for all** $item1$ and $item2$ in $ItemPairs$ **do**
      $userList1 \leftarrow$ list of users who have consumed item1
      $userList2 \leftarrow$ list of users who have consumed item2
      $usersIntersectionSet \leftarrow userList1 \cap userList2$
      $usersUnionSet \leftarrow userList1 \cup userList2$
      $userPairsIntersectionSet \leftarrow$ list of combination of $usersIntersectionSet$ taking 2 at a time.
      $userPairsUnionSet \leftarrow$ list of combination of $usersUnionSet$ taking 2 at a time.
      **for all** $user1$ and $user2$ in $userPairsIntersectionSet$ **do**
        $score \leftarrow userProfiles[user1]["weights"][attribute] + userProfiles[user2]["weights"][attribute]$
        $userSimilarityMatrix[user1][user2]['numerator'] += score$
      **end for**
      **for all** $user1$ and $user2$ in $userPairsUnionSetSet$ **do**
        $score \leftarrow userProfiles[user1]["weights"][attribute] + userProfiles[user2]["weights"][attribute]$
        $userSimilarityMatrix[user1][user2]['denominator'] += score$
      **end for**
    **end for**
  **end for**
**end for**

## 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:** $recommendedItemsIdeo$

Initialize $recommendedItemsIdeo$ to an empty list
Initialize $scoreIdeo$ to an associative array
**for all** $item$ in $itemList$ **do**
  **for all** $attribute$ in $itemList[item]$ **do**
    **for all** $value$ in $itemList[item][attribute]$ **do**
      $weight \leftarrow \left(\frac{1}{userWeights[attribute]*userWeights[value]}\right)^2$
      append weight to $scoreIdeo[item]['weight']$ list
      $avgRating \leftarrow$ average of ratings given by the user for all the items he has consumed with $attribute$, $value$ combination
      append $avgRating$ to $scoreIdeo[item]['rating']$ list
    **end for**
  **end for**
**end for**
Initialize $predictedRatingIdeo$ to an associative array
**for all** $item$ in $itemList$ **do**
  $predictedRatingIdeo[item] \leftarrow \frac{scoreIdeo[item]['weight'] \bullet scoreIdeo[item]['rating']}{\sum scoreIdeo[item]['weight']}$
**end for**
$recommendedItemsIdeo \leftarrow$ sort the items according to $predictedRatingIdeo$ in descending order

## 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 $userClusters$

## 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:** $recommendedItemsColl$

Initialize $recommendedItemsColl$ to an empty list
Initialize $scoreColl$ to an associative array
**for all** $cluster$ in $clusters$ **do**
  **if** user $\in$ $cluster$ **then**
    **for all** $otherUser$ in $cluster$ **do**
      **for all** $item$ in $itemList$ **do**
        **if** $otherUser$ has consumed the $item$ **then**
          $rating \leftarrow$ rating given the $otherUser$ for that item.
          append $rating$ to $scoreColl[item]['rating']$ list
          $similarity \leftarrow userSimilarityMatrix[user][otherUser]$
          append $similarity$ to $scoreColl[item]['similarity']$ list
        **end if**
      **end for**
    **end for**
  **end if**
**end for**
Initialize $predictedRatingColl$ to an associative array
**for all** $item$ in $itemList$ **do**
  $predictedRatingColl[item] \leftarrow \frac{scoreColl[item]['rating'] \bullet score[item]['similarity']}{\sum score[item]['similarity']}$
**end for**
$recommendedItemsColl \leftarrow$ sort the items according to $predictedRatingColl$ in descending order

# Chapter 11

# Project Highlights

Some of the major highlights of the project are:

## 11.1 In-homogeneity in Item Properties:

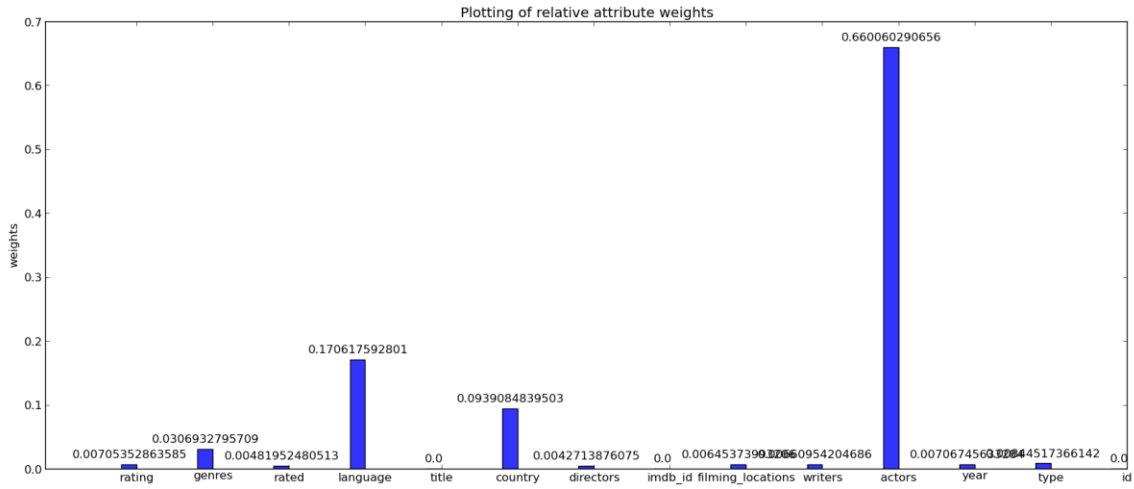
An item is associated with a set of properties. Given a set of items, there is no constraint that all the items should possess the same set of properties. Generally, recommendation systems assume homogeneity in properties and are not applicable to inhomogeneous item datasets.

Most of the data that is available in the real world is inhomogeneous and it takes a lot of effort in cleaning the data. The approach was developed keeping in mind the in-homogeneity of available real world data. The contrast between the earlier approaches and our approach is analogous to the contrast between SQL and NoSQL ideologies. In fact, our approach can be applied to two item entities which are completely different in real world but overlap in certain characteristics.

## 11.2 Unsupervised Dimensionality Reduction:

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.

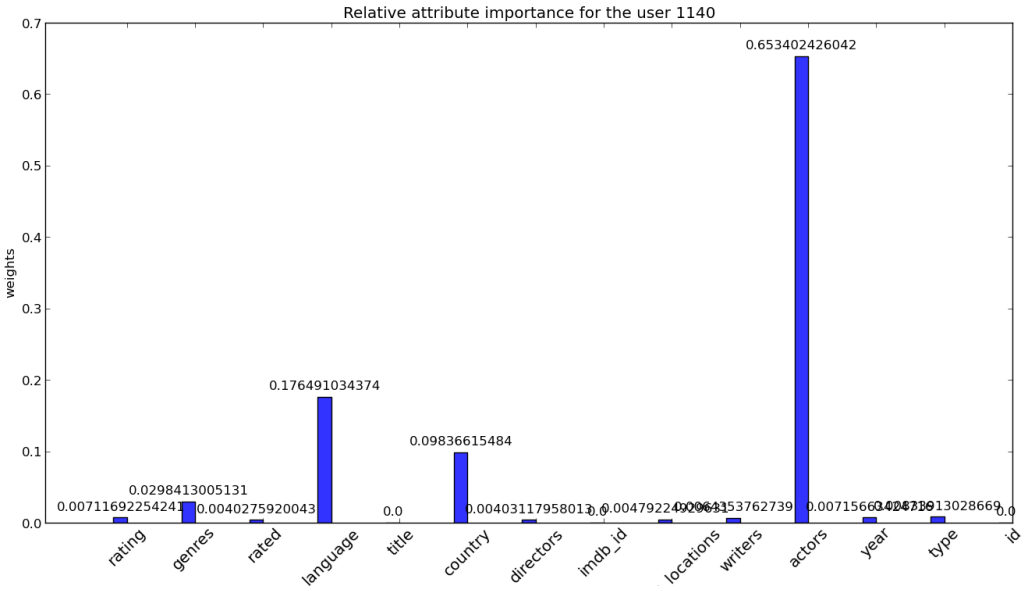
Consider an item with 100 properties. The importance of each of the properties is initially unknown. For the data-miner working with this item, it proves to be a hard task to manually select those features which *might* be of importance. There are less number of techniques which perform unsupervised dimensionality reduction of the dataset. Based on the pattern in which the users have consumed the items, we quantify and normalize the importance of each attribute in the item dataset. This might be of great use to a data-miner in order to gain insights about the data and the usage pattern.




Apart from unsupervised dimensionality reduction of the attribute, we also quantify the importance of each of the values that the attribute can take.

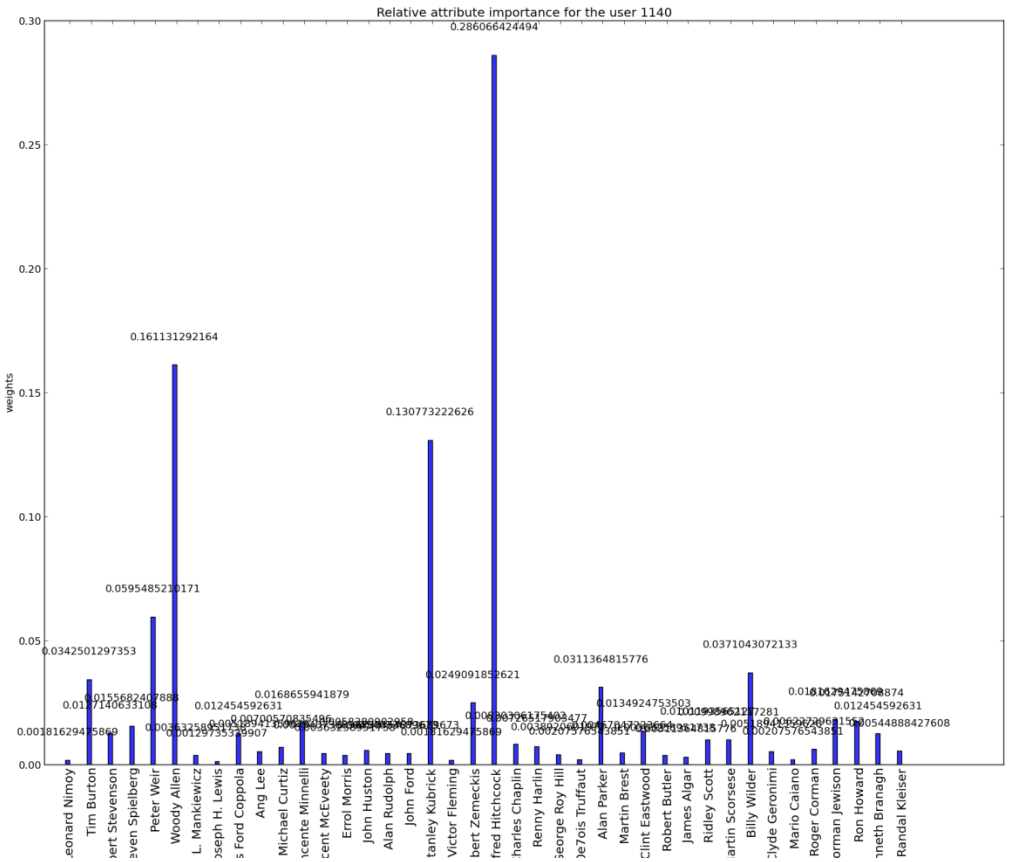
## 11.3 User Profiling - deducing the weights of attributes just by using ratings:

We perform user profiling as an integral step in the process of recommendation. From the dataset, the only information that we have is the rating that the users give to items. Observing the pattern in which the users have consumed the items, we compute the relative importance that the user gives to the various attributes. This relative importance is further used in unsupervised dimensionality on the item dataset.




## 11.4 User Profiling - deducing the weights of values for each attributes just by using ratings

Apart from deducing the attribute importance, we also deduce the relative importance of all the values that a particular attribute can take.
An interesting point to note here is that we are deducing all this information *only* based on the ratings that a particular user has given to set of items.




## 11.5 Rating prediction algorithm

Apart from generating recommendations, our approach also shells out a rating prediction algorithm as a by-product. In the internal workings of our approach, we predict the ratings given by the user to various items, in an idiocentric and a collaborative fashion. The target researcher/developer can halt the algorithm at this stage, represent the ratings and extract potentially useful information from it.

## 11.6 Extendibility of the approach to items with no ratings

As mentioned earlier, the data available from the internet is highly unstructured and unreliable. All the data required might not be available. The approach that we have developed is designed to be robust in handling incomplete data. Consider a scenario

wherein we have partial information about the items and we have no information about the user's ratings. In such a case, by building the user dataset such that all the consumed items have the same rating, the algorithm predicts the probability with which the user will consume an item.

## 11.7 Possibility of modifying this algorithm to the Link Prediction problem if the user set and the item set are the same

Link Prediction refers to inferring the likelihood of new interactions among the members of a social network, given a snapshot. We propose a solution to this problem by reducing the link prediction problem to a recommendation problem, and then solving the recommendation problem using the developed approach. Assuming that the snapshot of the social network is available, the item dataset and the user dataset can be built in the following way. The item dataset can be reverse engineered by analyzing the links in the snapshot of the social network. The members of the social network form the users in the user set. The items of consumption for each user are the same as the user's neighbors in the snapshot of the network.

If any extra information is available about the members, it can be included in the item dataset. If the edges in the snapshot of the network are weighted by the strength of the bond between its ends, then the weights can be mapped on to the ratings in the user dataset. Again, this option highlights the robustness of the algorithm.

## 11.8 Idiosyncrasy factor for each user

Idiosyncrasy factor is a real number in the range [0-1] which quantifies how idiocentric a user is. During the bootstrapping phase of our approach, we maintain a neutral opinion about the user. When we have a system that can take feedback about the recommendations that we provide, we tweak the Idiosyncrasy factor based on the choice that the user makes. The feedback obtained influences the future recommendations and the process iterates until convergence. An idiocentric factor of 1 represents complete self-oriented behavior and a factor of 0 implies complete group-oriented behavior.

# Chapter 12

# Integration

The integration involved in our project was majorly for two sections:

## 12.1 Algorithmic Integration:

The approach we followed during the course of development was linear and we developed the modules in a linear fashion. To divide the development part equally among the team, we used the interfaces approach. The modules were developed assuming the dependent module is present and has a specific input and output parameters. This was a major integration task, as the dependency modules were also written in parallel. Git versioning system was a big help in code management and version control.

The next part of algorithmic integration involved combining the idiocentric and collaborative recommendations. The results obtained for any user has two parts, results as per his idiocentric behavior and as per his collaborative behavior. The weight for any attribute in idiocentric is, say *alpha*. Then the weight for the attribute in collaborative will be *(1-alpha)*. The combined recommendation is obtained by the weighted combination of idiocentric and collaborative predicted ratings, with weights *alpha* and *1-alpha.*

## 12.2 Front-end Back-end Integration:

All through the course of the project, the back-end development was maintained using a single file approach. While integrating with the UI, we had to refactor the code into multiple files as per the functions, such that they can be used in the handler methods of the UI elements. There are two different flows in this part:

12.2.1 Communication from UI to backend: The UI is coded using Qt framework. On click of a button, the event handlers are called in Qt. A process can be spawned using the QProcess object in the event handler. We extract certain parameters from the UI and spawn a python process in the form of supporting scripts to run the back-end file. To make this flow work, we had to write many supporting files which help in execution of the modules of the code.

12.2.2 Communication from backend to UI: After the code and supporting scripts are executed in the backend, the desired results are generated. These results have to be brought back to the front-end where they can be displayed. We had to write many supporting scripts to showcase these results on the UI.

Also many other supporting scripts were written for plotting the graphs, populating the dropdown menus etc.

As discussed above, the main portion of integration involved changing the linear approach to a modular approach and writing supporting scripts to integrate frontend and backend.

# Chapter 13

# Testing

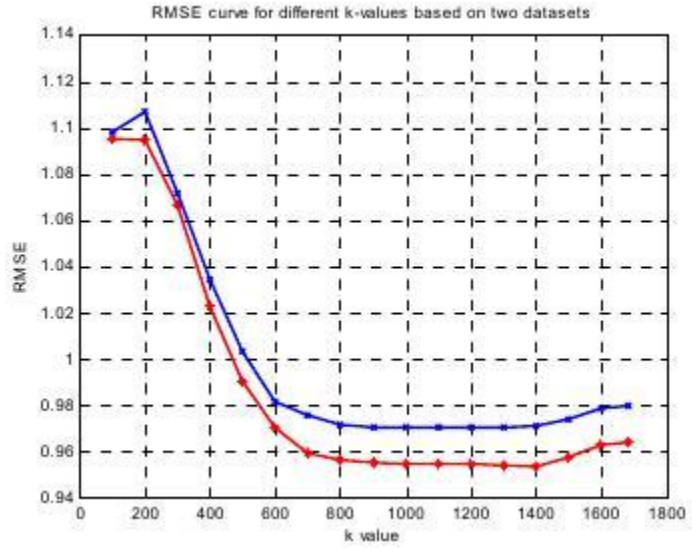
Zhouxiao Bao and Haiying Xia, in their paper titled "Movie Rating Estimation and Recommendation", build a movie rating prediction system based on the training sets provided by MovieLens. The paper was written as a part of their final project in Machine Learning course, Autumn 2012 at Stanford University. They present various approaches that signify the Root Mean Square error on changing the parameters for various predictor algorithms. We use the results obtained by their works as a benchmark on comparison to our algorithm.

Root Mean Square Error is a common metric that is used to evaluate the accuracy of prediction algorithms. It is defined as the square root of average of error squares, with the error being taken as the deviation of the predicted rating from the actual rating. The mathematical definition of RMSE is as follows:

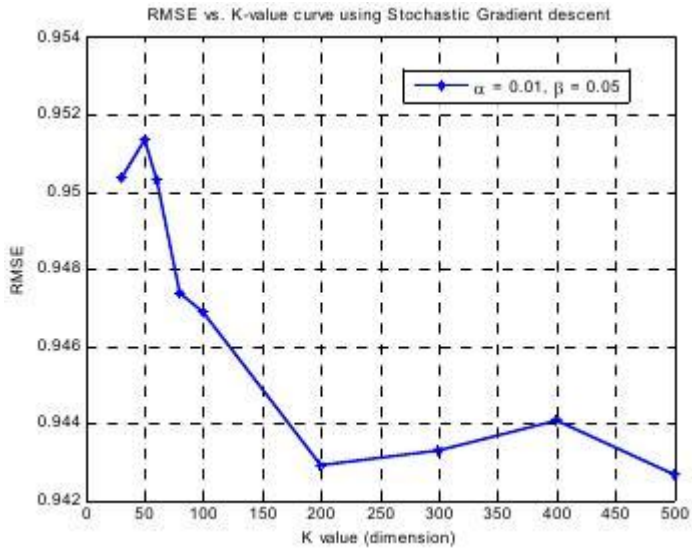
$$\mathrm{RMSE} = \sqrt{\frac{1}{|S_{test}|}\sum_{(u,i)\in S_{test}} (R_{ui} - T_{ui})^2}$$

Bao and Xia, in their paper, have chosen MovieLens 100k Dataset1 as the training set. This dataset contains 100,000 ratings from 943 on 1682 movies, in which each user has rated at least 20 movies. Among MovieLens 100k Data Set, they choose ua.base/test and ub.base/test for result comparison between different training sets and testing sets. The whole set u data is split into a training set and a test set with exactly 10 ratings per user in the test set. The sets ua.test and ub.test are disjoint.

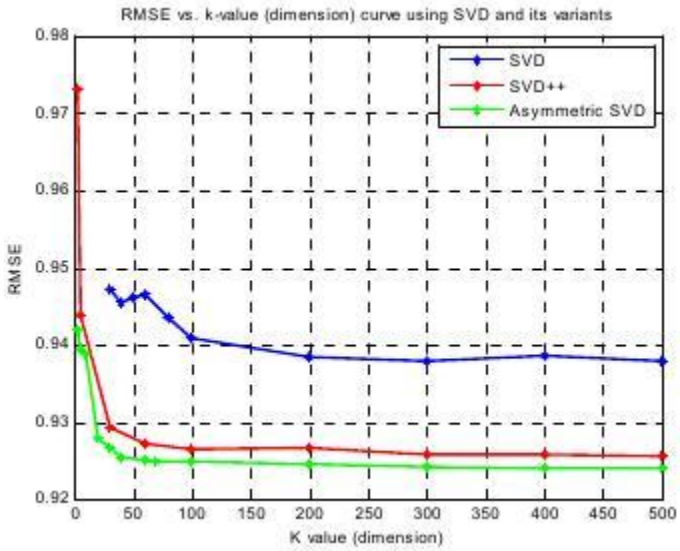
On application of Baseline Predictors technique, the RMSE error on the ua and ub datasets were found to be **0.96648** and **0.97737** respectively. On application of KNN (K-Nearest Neighbor) technique to predict movie ratings, the following plot was obtained. From the plot, it can be seen that the least RMSE that could be achieved with KNN was around **0.95**.



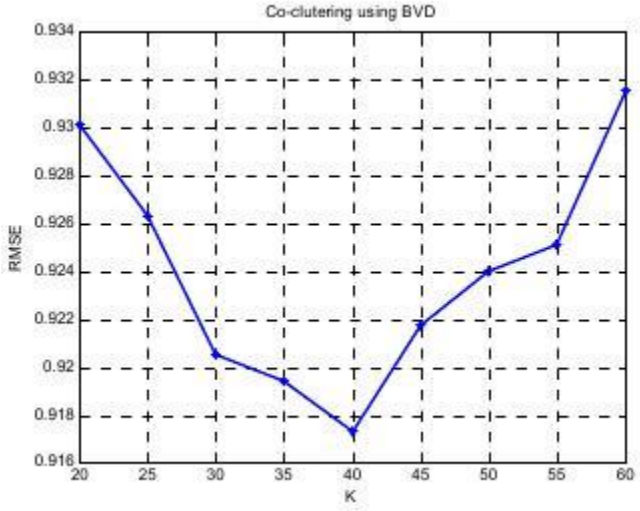

On application of stochastic gradient method for rating prediction with the dimension as a parameter, the following plot was obtained. The least RMSE that could be achieved was around **0.943**.




On application of SVD and its variants with the dimension as a parameter, the following plot was obtained. The least RMSE for SVD, SVD++ and Asymmetric SVD are around **0.939**, **0.926** and **0.924** respectively.




The Global Neighborhood model produced an RMSE of **0.931926**. The best results were obtained when co-clustering using BVD was applied on the datasets. The plot below shows the variation of the results. The minimum RMSE obtained was around **0.917.**




Note that six out of the eight techniques were parameterized and the optimum value of the parameter would be found out by iterative execution of the algorithm by varying the parameter on x-axis.

We tested the accuracy of our algorithm on the aggregated MovieLens_1m dataset. The dataset consists of 1,000,000 ratings from 6040 users on 3883 movies, in which each user has rated atleast 20 movies. To obtain metadata about the movies themselves, we used the APIs from http://imdbapi.org and http://omdbapi.com/. We split the user dataset in the ratio 80:20 and used 80% of the user data for training. The rating predictions were made on the remaining 20% of the dataset.

To test the accuracy of our approach, we choose RMSE as the metric. On applying the algorithm, we obtained an RMSE of **0.903**. The key points to be noted here are as follows. We are applying our algorithm on a significantly larger dataset with more number of users, ratings and movies. We are using metadata about the movies in order to determine the final rating. The algorithm is non-parameterized. i.e. there is no necessity to run the algorithm iteratively to find out an optimal point that gives the best accuracy. The algorithm has a wide range of by-products that serve as a solution to various other problems.

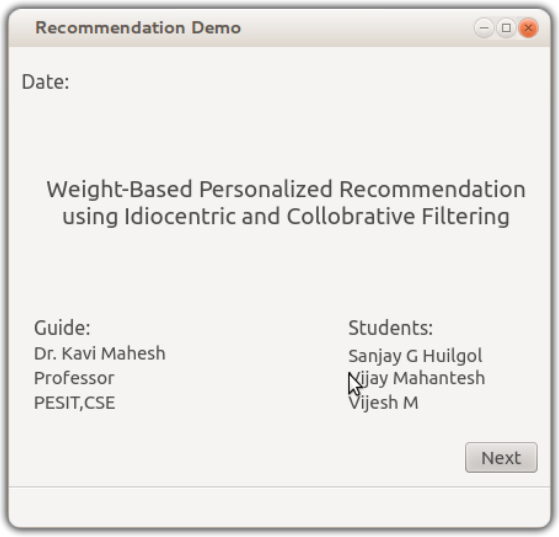
In order to give out recommendations, we predict the rating that the user gives to an item that hasn't been previously consumed. On sorting the movies in the decreasing order of their ratings, the recommendation list is obtained. The rating prediction tests that were carried out prove that our recommendation is indeed legitimate.

# Chapter 14

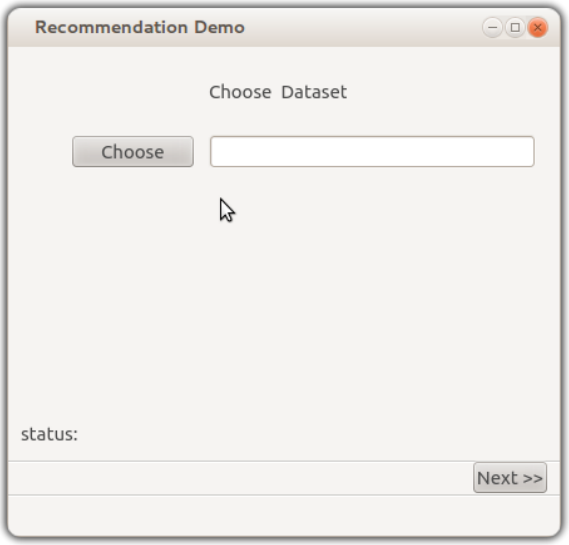
# Screenshots

The following screenshots show the UI at various stages during the execution of the system:

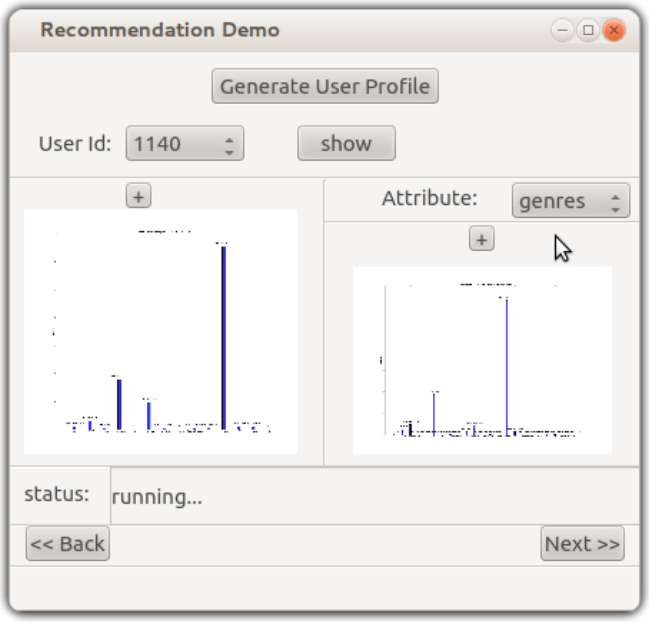
14.1 The Welcome Screen:



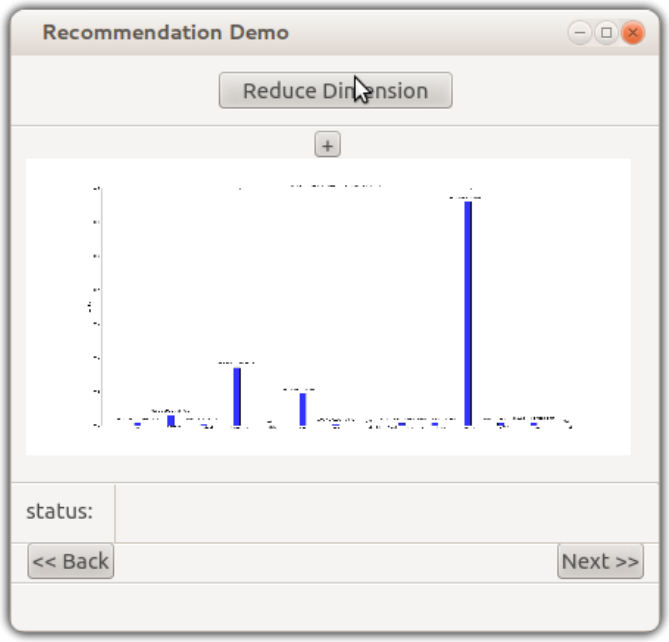

14.2 The second screen which prompts to choose for a dataset:



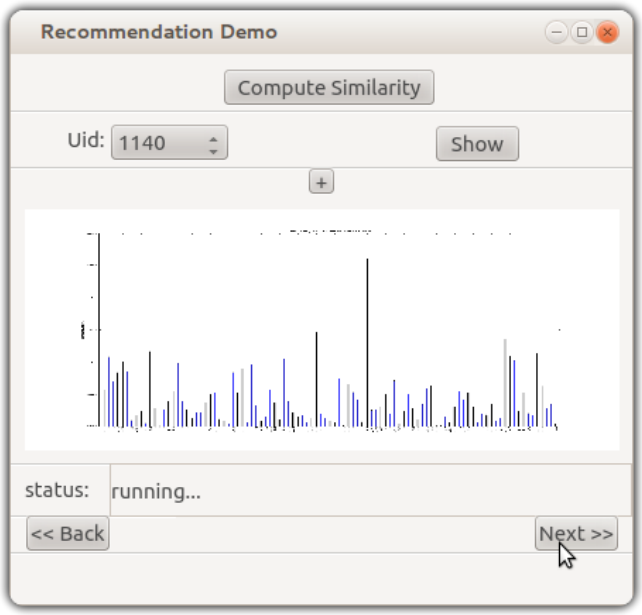
14.3 On clicking "Generate User Profiles" and selecting a user, the relative importance of attribute and the values will be generated.



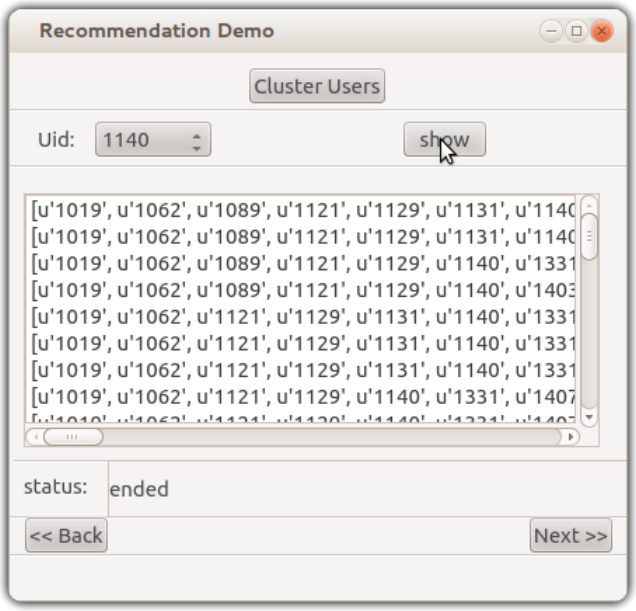

14.4 On clicking "Reduce Dimensions", unsupervised dimensionality reduction will be carried out.



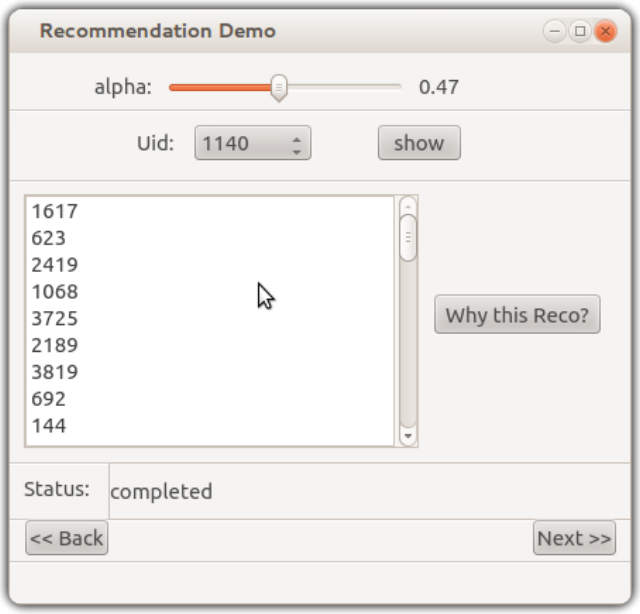
14.5 On clicking "Compute Similarity", the algorithm runs the code for generating the similarity matrix. On choosing a user, we show his similarities with all other users.



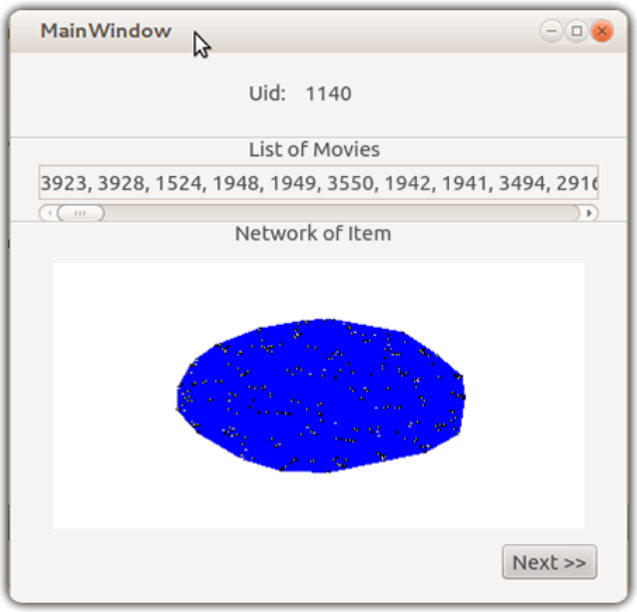

14.6 On clicking "Cluster Users" and choosing a user, the list of clusters that the user belongs to will be shown.



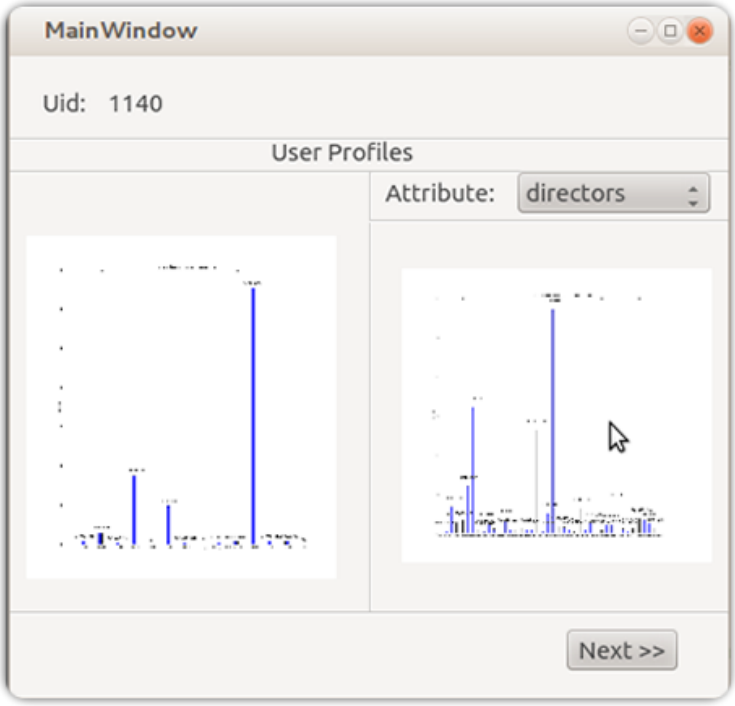
14.7 On setting the value of alpha and querying for recommendations for a particular user, the sorted list of item IDs will be shown. We also give an insight into how the list was recommended:



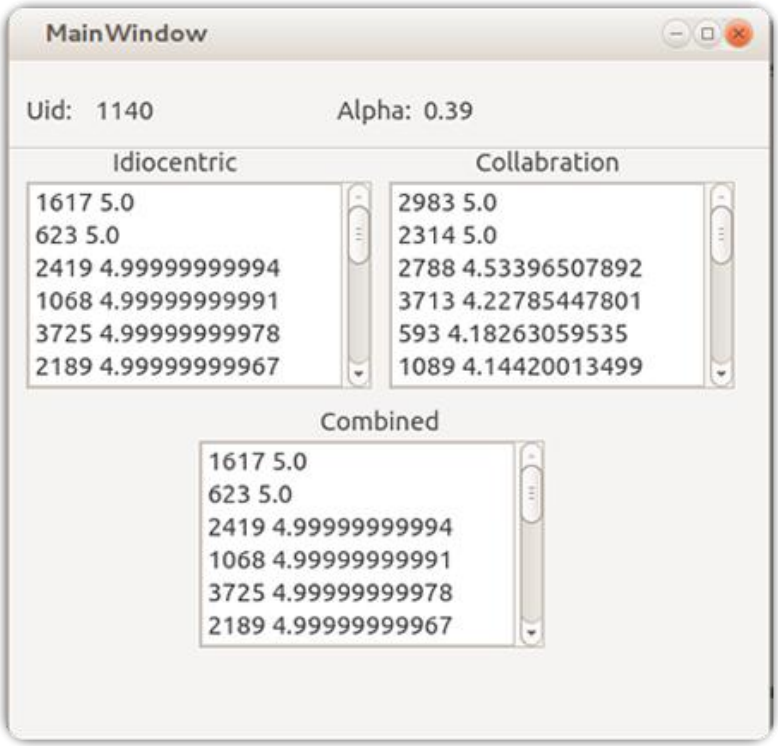

14.8 Item Graph for the selected user:



## 14.9 User Profile of the selected user:




## 14.10 The component-wise split of recommendation



# Chapter 15

# Conclusion

In this project, we demonstrated a novel approach for recommending items to users and performed experiments on the aggregated MovieLens dataset to test our algorithm. The target users for our application are researchers and firms that use some form of recommendation in their products.

There has been a lot of significant work done in the field of recommendation techniques. Recommendation systems have gained popularity since the inception of the papers on collaborative filtering since the 1990s. The techniques that have been developed are versatile in nature. Various forms of recommendation systems have been developed based on the application specificity, supervised / unsupervised learning, collaborative filtering, content-based filtering, hybrid systems etc. In this project, we designed a weight-based, personalized and robust recommendation technique that performs unsupervised dimensionality reduction of attributes and values on the dataset and incubates a rating prediction algorithm and a link prediction algorithm as by-products.

We use graph as the core reference structure. On an abstract level, we form a graph of items and the relations between items are manifested as properties of the corresponding edges. The actual implementation of the algorithm involves a reverse indexed data structure for efficiency purposes. Based on the users' history of item consumption, we profile each user and deduce the weight that the user gives to the attributes and the values that the attribute can take. We use the profiles to statistically determine the importance of each attribute and value in the dataset, in turn performing dimensionality reduction. To find out the similarity between users, we modified the simple Jaccard similarity to consider the properties of users and items. Based on a similarity threshold, we form overlapping clusters of users. To provide recommendation for a particular user, we follow two distinct approaches: idiocentric and collaborative filtering. These recommendations are then combined to form hybrid recommendation.

The algorithm has been implemented using Python as the primary language. During the course of development, we have used a large number of python libraries. The major ones include networkx, pickle, json and numpy. The networkx module is used to store and manipulate graph

objects in memory. The pickle and json libraries were used to store and retrieve the reference structures on the disk. We have used the Git versioning system to maintain various versions of our code and denote landmarks in the development. The UI mockups were created using Pencil, a mockup creator tool for Ubuntu. The final UI was created using Qt framework. During the course of the development, we used Gephi to visualize and analyze the intermediate graphs.

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 for our project can be found at https://github.com/vijeshm/recommendation-algorithm-fyp. We encourage researchers and developers to contribute to the code and the idea behind the algorithm.

There were numerous hurdles that we had to face during the execution of the project. We addressed the several issues concerning exporting huge in-memory objects on to a disk and came up several custom solutions to solve the problem. In order to categorize the non-categorical data, we built a trivial, yet powerful clustering technique. We had issues with memory errors on a 32-bit OS, which was realized after a couple of days of execution. We had to consider a lot of tradeoffs when choosing the implementation language and decided upon python as the programming language. As far as the databases are concerned, we chose against using a database and decided to use in-memory objects. Consequently, we had to optimize the storage of in memory objects. In order to reduce the memory consumption, we exploited the graph structure and used reverse indexing to solve the issue. Finally, we had to choose linear execution of the algorithm as opposed to parallel execution, even though the algorithm is inherently parallelizable. We used three approaches for parallelizing the execution, but due to the overheads involved, we had to choose linear execution against all of the three approaches.

The approach has been designed with prime importance being given to robustness. This is because the real-world data is incomplete and unreliable. Other highlights of the project include user profiling by deducing the relative importance of attributes and values alike, unsupervised dimensionality reduction on the item dataset, a rating prediction approach, reducibility to link

prediction algorithm and the extendibility of the approach to the cases where user rating data is unavailable.

Our results show that the importance of each dimension of the dataset can be weighted in an unsupervised manner, in accordance with the pattern that the users follow to consume the items. We deduce the relative importance that each user gives to each of the attributes and the values that it can take. The recommendations that we provide to a user is personalized based on the corresponding relative importance of the corresponding user.

In order to test our approach, we used the aggregated movielens_1m dataset as a benchmark. The movielens_1m dataset consists of 1 million ratings from 6040 users on 3883 movies. A survey paper on moviens_100k dataset by Bao and Xia, from autumn 2012 Machine Learning Course at Stanford University, compares four parameterized approaches for rating prediction. The movielens_100k dataset consists of 100,000 ratings from 1000 users on 1700 movies. The RMS error for each of the four approaches varies as the parameter for the corresponding algorithm varies. Out of the four minima RMS errors, the least RMS error in rating prediction was found to be 0.917. On application of our non-parameterized algorithm on the aggregated movielens_1m dataset, we were able to achieve an RMS error of 0.903.

The future enhancements for the project include code parallelization, manifesting the algorithm as a link prediction algorithm and a rating prediction algorithm, development of more space-efficient and time-efficient techniques, application of the standard machine learning techniques for a better comparison and the benchmarking of the results.

# Chapter 16

# Future Enhancements

Since our project is partially research based, there is lot of scope for future enhancement. The developed approach is open-ended in the perspective of the number of combinations of algorithms that can be plugged in at various points in the algorithm. The future enhancements for the project include:

1. Code Parallelization:

   The approach and methods that we use is inherently parallel. In the course of the project, we attempted to parallelize the code. We had come up with three different solutions, but the overheads caused by interprocess and interthread communication slowed down the method further. In the near future, we hope to identify more components that can be parallelized and build modules that are optimized to utilize multiple cores to efficiently generate the reference structures.

2. Manifesting the algorithm as a link prediction algorithm and a rating prediction algorithm:

   The power of our approach lies in its robustness. In general, any problem that can be reduced to a problem of recommendation can be solved using our approach. Consider the rating prediction problem for example. If we modify our algorithm to stop at the penultimate step, the probable rating by a user for the item can be predicted. Initially, if we have no data about the user ratings, but only possess knowledge about which items are consumed, our algorithm predicts the probability with which the item is consumed. In the Social Network Analysis domain, link prediction is a well-studied problem. Given a snapshot of a social network, a mapping can be constructed between the network and the input datasets to our algorithm. On execution, our algorithm predicts the probability of linkage between members of the social network.

3. Development of a more space-efficient and time-efficient algorithm:
   For huge datasets, the current implementation of the algorithm consumes a lot of in-memory space and time. Although a lot of optimizations have been carried in increase space efficiency, there is a lot of work that needs to be done to address time complexity.

4. Application of the standard Machine Learning techniques on the datasets:
   In order to rigidly benchmark our algorithm, there is a need to implement various Machine Learning and Data Mining techniques on the datasets and compare the results.

5. Testing against other datasets:
   In the course of our project, we have been testing the performance and accuracy of our algorithm on the aggregated movielens dataset. In order to verify the accuracy of our algorithm, there is a need to test it against other datasets for its robustness. Various heterogeneous item and user datasets can be combined into a single one and various metrics could be applied on the results to test the limits of robustness of our algorithm. This opens up a lot of scope for research in the domain.

6. Application to a web-based system:
   The developed approach can be applied to a web-based product that uses some form of recommendation, or any feature that can be reduced to the problem of recommendation. In order to apply our approach to a web-based system, it must be able to handle real-time data. During the design of the approach, some basic steps have been taken in this direction.

# Chapter 18

# References

[1] Fabian Abel, Qi Gao, Geert-Jan Houben, and Ke Tao. *Analyzing temporal dynamics in twitter profiles for personalized recommendations in the social web.* In ACM WebSci'11, pages 1-8, 2011.

[2] Toine Bogers. *Movie recommendation using random walks over the contextual graph*. In CARS'10: *Proceedings of the 2nd workshop on context-aware recommender systems, 2010.*

[3] Frederico Durao and Peter Dolog. *Personalized tag-based recommendation in social web systems.* In CEUR Workshop Proceedings, volume 485, pages 40-49, 2009.

[4] Ziyu Guan, Jiajun Bu, Qiaozhu Mei, Chun Chen, and Can Wang. *Personalized tag recommendation using graph-based ranking on multi-type interrelated objects.* In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval.* ACM, 2009.

[5] W. Hill, Stead L., Rosenstein M., and Furnas G. *Recommending and evaluating choices in a virtual community of use*. In Proceedings of CHI, '95, 1995.

[6] Konstan J., Miller B., Maltz. D., Herlocker J., Gordon L., and Reidl J. Grouplens: *Applying collaborative filtering to usenet news*. Communications of the ACM, 40(3), pages 77-87, 1997.

[7] Resnick P., Lacovou N., Suchak M., Bergstrom P., and Reidl J. Grouplens: *An open architecture for collaborative filtering of netnews.* In proceedings of CSCW'94, Chapel Hill, NC, 1994.

[8] Breese J S, Heckerman D, , and Kadie C. *Empirical analysis of predictive algorithms for collaborative filtering*. In *Proceedings of 14th Conference on Uncertainty in Artificial Intelligence*, pages 43-52, 1998.

[9] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. *Item-based collaborative filtering recommendation algorithms*. In *10th international conference on World Wide Web*, pages 285-295, 2001.

[10] Andriy Shepitsen, Jonathan Gemmell, Bamshad Mobasher, and Robin Burke. *Personalized recommendation in social tagging systems using hierarchical clustering.* In ACM conference on Recommender systems, pages 259-266, 2008.

[11] Shardanand U. and Maes P. Social information filtering: *Algorithms for automating 'word of mouth'*. In Proceedings of CHI '95. Denver CO., 1995.

[12] http://en.wikipedia.org/wiki/Graph_database

[13] http://www.jython.org/

[14] http://en.wikipedia.org/wiki/Jython

[15] http://www.hypergraphdb.org/index

[16] http://objectivity.com/

[17] http://pandas.pydata.org/

[18] http://blog.perrygeo.net/2008/04/19/a-quick-cython-introduction/

[19]http://books.google.co.in/books?id=HMfJHBW8x_EC&pg=PA37&lpg=PA37&dq=clustering+single+dimensional+numerical+data&source=bl&ots=Vhj07kk1vs&sig=l0XNbHjFEj74OBC_SO2TN12JeY&hl=en&sa=X&ei=RrdRUavPK87LrQe2n4GICg&ved=0CF0Q6AEwBQ#v=onepage&q=clustering%20single%20dimensional%20numerical%20data&f=false

[20] http://docs.python.org/2/library/multiprocessing.html

[21] http://www.tutorialspoint.com/python/python_multithreading.htm

[22] http://www.devshed.com/c/a/Python/Basic-Threading-in-Python/3/

[23] http://stackoverflow.com/questions/11968689/python-multithreading-wait-till-all-threads-finished

[24]http://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set

[25]http://www.google.co.in/url?sa=t&rct=j&q=&esrc=s&source=web&cd=4&ved=0CFQQFjAD&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.107.9895%26rep%3Drep1%26type%3Dpdf&ei=uYFYUfOXJcX7rAfUvYDYBw&usg=AFQjCNFGfk0V7cG6e0DvAi9WTdkU8-zkUw&sig2=c5oT8YD9GLLrLdrxIKepEQ&bvm=bv.44442042,d.bmk&cad=rja