1. **RECOMMENDATION SYSTEM**

**1.1 Introduction**

The explosive growth in the amount of available digital information and the number of visitors to the Internet have created a potential challenge of information overload which hinders timely access to items of interest on the Internet. Information retrieval systems, such as Google, DevilFinder and Altavista have partially solved this problem but prioritization and personalization (where a system maps available content to user’s interests and preferences) of information were absent. This has increased the demand for recommender systems more than ever before. Recommender systems are information filtering systems that deal with the problem of information overload by filtering vital information fragment out of large amount of dynamically generated information according to user’s preferences, interest, or observed behavior about item. Recommender system has the ability to predict whether a particular user would prefer an item or not based on the user’s profile.

Recommender systems are beneficial to both service providers and users. They reduce transaction costs of finding and selecting items in an online shopping environment. Recommendation systems have also proved to improve decision making process and quality. In e-commerce setting, recommender systems enhance revenues, for the fact that they are effective means of selling more products. In scientific libraries, recommender systems support users by allowing them to move beyond catalog searches. Therefore, the need to use efficient and accurate recommendation techniques within a system that will provide relevant and dependable recommendations for users cannot be over-emphasized.

Recommender Systems are simple algorithms which aim to provide the most relevant and accurate items to the user by filtering useful stuff from of a huge pool of information base. Recommendation engines discovers data patterns in the data set by learning consumer’s choices and produces the outcomes that co-relates to their needs and interests.

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



Fig 1. Recommendation phases

**1.2 Dataset collection for recommendation model**

Data collection is the process of gathering and measuring information on targeted variables in an established systematic fashion, which then enables one to answer relevant questions and evaluate outcomes. Data collection is a component of research in all fields of study including [physical](https://en.wikipedia.org/wiki/Physical_science) and [social sciences](https://en.wikipedia.org/wiki/Social_science), [humanities](https://en.wikipedia.org/wiki/Humanities), and [business](https://en.wikipedia.org/wiki/Business). While methods vary by discipline, the emphasis on ensuring accurate and honest collection remains the same. The goal for all data collection is to capture quality evidence that allows analysis to lead to the formulation of convincing and credible answers to the questions that have been posed.

Regardless of the field of study or preference for defining data (quantitative or qualitative), accurate data collection is essential to maintaining the integrity of research. Both the selection of appropriate data collection instruments (existing, modified, or newly developed) and clearly delineated instructions for their correct use reduce the likelihood of errors occurring.

Consequences from improperly collected data include:

* Inability to answer research questions accurately;
* Inability to repeat and validate the study.

A formal data collection process is necessary as it ensures that the data gathered are both defined and accurate and that subsequent decisions based on arguments embodied in the findings are valid. The process provides both a baseline from which to measure and in certain cases an indication of what to improve.

When building a model from a user's behavior, a distinction is often made between explicit and implicit forms of data collection.

Examples of explicit data collection include the following:

* Asking a user to rate an item on a sliding scale.
* Asking a user to search.
* Asking a user to rank a collection of items from favorite to least favorite.
* Presenting two items to a user and asking him/her to choose the better one of them.
* Asking a user to create a list of items that he/she likes.

Examples of implicit data collection include the following:

* Observing the items that a user views in an online store.
* Analyzing item/user viewing times.
* Keeping a record of the items that a user purchases online.
* Obtaining a list of items that a user has listened to or watched on his/her computer.
* Analyzing the user's social network and discovering similar likes and dislikes.

The recommender system compares the collected data to similar and dissimilar data collected from others and calculates a list of recommended items for the user.

In our project, we have implemented a data set of 100k ratings that was open source and available at movielens.org for educational purposes. The dataset contains movie names, movie ids, around 700 users and their ratings for the movies according to their taste.

**1.3 Data cleaning**

Data cleansing, also called data scrubbing, is the process of amending or removing data in a database that is incorrect, incomplete, improperly formatted, or duplicated. An organization in a data-intensive field like banking, insurance, retailing, telecommunications, or transportation might use a data cleansing tool to systematically examine data for flaws by using rules, algorithms, and look-up tables. Typically, a database scrubbing tool includes programs that are capable of correcting a number of specific type of mistakes, such as adding missing zip codes or finding duplicate records. Using a data clenasing tool can save a database administrator a significant amount of time and can be less costly than fixing errors manually.

Data cleansing is the process of altering data in a given storage resource to make sure that it is accurate and correct. There are many ways to pursue data cleansing in various software and data storage architectures; most of them center on the careful review of data sets and the protocols associated with any particular data storage technology.

Data cleansing is sometimes compared to data purging, where old or useless data will be deleted from a data set. Although data cleansing can involve deleting old, incomplete or duplicated data, data cleansing is different from data purging in that data purging usually focuses on clearing space for new data, whereas data cleansing focuses on maximizing the accuracy of data in a system. A data cleansing method may use parsing or other methods to get rid of syntax errors, typographical errors or fragments of records. Careful analysis of a data set can show how merging multiple sets led to duplication, in which case data cleansing may be used to fix the problem.

Many issues involving data cleansing are similar to problems that archivists, database admin staff and others face around processes like data maintenance, targeted data mining and the extract, transform, load (ETL) methodology, where old data is reloaded into a new data set. These issues often regard the syntax and specific use of command to effect related tasks in database and server technologies like SQL or Oracle. Database administration is a highly important role in many businesses and organizations that rely on large data sets and accurate records for commerce or any other initiative.

After cleansing, a data set should be consistent with other similar data sets in the system. The inconsistencies detected or removed may have been originally caused by user entry errors, by corruption in transmission or storage, or by different data dictionary definitions of similar entities in different stores. Data cleansing differs from data validation in that validation almost invariably means data is rejected from the system at entry and is performed at the time of entry, rather than on batches of data.

The actual process of data cleansing may involve removing typographical errors or validating and correcting values against a known list of entities. The validation may be strict (such as rejecting any address that does not have a valid postal code) or fuzzy (such as correcting records that partially match existing, known records). Some data cleansing solutions will clean data by cross checking with a validated data set. A common data cleansing practice is data enhancement, where data is made more complete by adding related information. For example, appending addresses with any phone numbers related to that address. Data cleansing may also involve activities like, harmonization of data, and standardization of data. For example, harmonization of short codes (st, rd, etc.) to actual words (street, road, etcetera). Standardization of data is a means of changing a reference data set to a new standard, ex, use of standard codes.

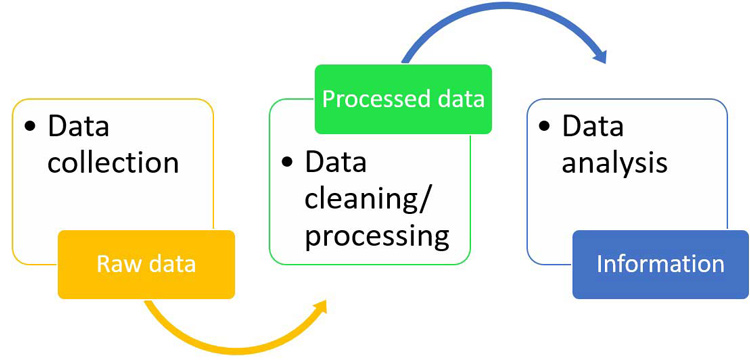


Fig 2. Data cleansing process

Challenges:

* Error correction and loss of information: The most challenging problem within data cleansing remains the correction of values to remove duplicates and invalid entries. In many cases, the available information on such anomalies is limited and insufficient to determine the necessary transformations or corrections, leaving the deletion of such entries as a primary solution. The deletion of data, though, leads to loss of information; this loss can be particularly costly if there is a large amount of deleted data.
* Maintenance of cleansed data: Data cleansing is an expensive and time-consuming process. So after having performed data cleansing and achieving a data collection free of errors, one would want to avoid the re-cleansing of data in its entirety after some values in data collection change. The process should only be repeated on values that have changed; this means that a cleansing lineage would need to be kept, which would require efficient data collection and management techniques.
* Data cleansing in virtually integrated environments: In virtually integrated sources like IBM’s DiscoveryLink, the cleansing of data has to be performed every time the data is accessed, which considerably increases the response time and lowers efficiency.
* Data-cleansing framework: In many cases, it will not be possible to derive a complete data-cleansing graph to guide the process in advance. This makes data cleansing an iterative process involving significant exploration and interaction, which may require a framework in the form of a collection of methods for error detection and elimination in addition to data auditing. This can be integrated with other data-processing stages like integration and maintenance.

In our project, we had a dataset provided by movielens.org which was not perfectly clean. There were errors in the movie name columns of the movie data set. We used PHP and SQL to correct the concerned dataset the code of which is included below. We had multiple movie names with the prefix ‘The’ put as a suffix in the movie name. For eg, the movie ‘The Usual Suspects’ was shown in the table as ‘Usual Suspects, The’. We had to implement a query by which we found out all such movie names and corrected them thereby cleaning the dataset.

**Data cleaning code:**



**1.4 Different methods for recommender model**

* **Collaborative Filtering:**

Collaborative filtering is a domain-independent prediction technique for content that cannot easily and adequately be described by metadata such as movies and music. Collaborative filtering technique works by building a database (user-item matrix) of preferences for items by users. It then matches users with relevant interest and preferences by calculating similarities between their profiles to make recommendations. Such users build a group called neighborhood. An user gets recommendations to those items that he has not rated before but that were already positively rated by users in his neighborhood.

Collaborative Filtering algorithm considers “User Behaviour” for recommending items. They exploit behaviour of other users and items in terms of transaction history, ratings, selection and purchase information. Other users behaviour and preferences over the items are used to recommend items to the new users. In this case, features of the items are not known.

Collaborative filtering has two senses, a narrow one and a more general one.

In the newer, narrower sense, collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person. For example, a collaborative filtering recommendation system for television tastes could make predictions about which television show a user should like given a partial list of that user's tastes (likes or dislikes). Note that these predictions are specific to the user, but use information gleaned from many users. This differs from the simpler approach of giving an average (non-specific) score for each item of interest, for example based on its number of votes.

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

The growth of the Internet has made it much more difficult to effectively extract useful information from all the available online information. The overwhelming amount of data necessitates mechanisms for efficient information filtering. Collaborative filtering is one of the techniques used for dealing with this problem.

The motivation for collaborative filtering comes from the idea that people often get the best recommendations from someone with tastes similar to themselves. Collaborative filtering encompasses techniques for matching people with similar interests and making recommendations on this basis.

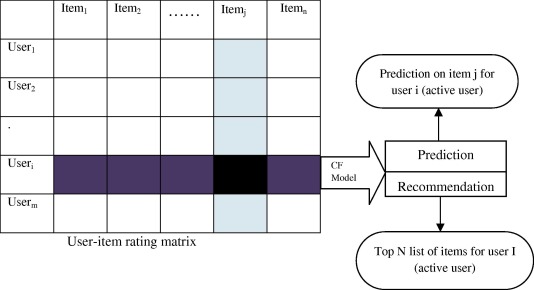


Fig 3. User- Item collaborative filtering

Collaborative filtering algorithms often require (1) users' active participation, (2) an easy way to represent users' interests, and (3) algorithms that are able to match people with similar interests.

Typically, the workflow of a collaborative filtering system is:

* A user expresses his or her preferences by rating items (e.g. books, movies or CDs) of the system. These ratings can be viewed as an approximate representation of the user's interest in the corresponding domain.
* The system matches this user's ratings against other users' and finds the people with most "similar" tastes.
* With similar users, the system recommends items that the similar users have rated highly but not yet being rated by this user (presumably the absence of rating is often considered as the unfamiliarity of an item).

A key problem of collaborative filtering is how to combine and weight the preferences of user neighbors. Sometimes, users can immediately rate the recommended items. As a result, the system gains an increasingly accurate representation of user preferences over time.

* **Content based Filtering:**

Content-based filtering, also referred to as cognitive filtering, recommends items based on a comparison between the content of the items and a user profile. The content of each item is represented as a set of descriptors or terms, typically the words that occur in a document. The user profile is represented with the same terms and built up by analyzing the content of items which have been seen by the user.

Several issues have to be considered when implementing a content-based filtering system. First, terms can either be assigned automatically or manually. When terms are assigned automatically a method has to be chosen that can extract these terms from items. Second, the terms have to be represented such that both the user profile and the items can be compared in a meaningful way. Third, a learning algorithm has to be chosen that is able to learn the user profile based on seen items and can make recommendations based on this user profile.

The information source that content-based filtering systems are mostly used with are text documents. A standard approach for term parsing selects single words from documents. The vector space model and latent semantic indexing are two methods that use these terms to represent documents as vectors in a multi dimensional space.

Relevance feedback, genetic algorithms, neural networks, and the Bayesian classifier are among the learning techniques for learning a user profile. The vector space model and latent semantic indexing can both be used by these learning methods to represent documents. Some of the learning methods also represent the user profile as one or more vectors in the same multi dimensional space which makes it easy to compare documents and profiles. Other learning methods such as the Bayesian classifier and neural networks do not use this space but represent the user profile in their own way.

Content based systems, recommends item based on a similarity comparison between the content of the items and a user’s profile. The feature of items are mapped with feature of users in order to obtain user – item similarity.

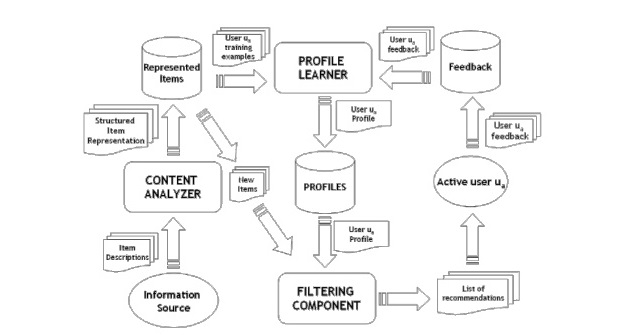


Fig 4. Content based filtering

Content-based technique is a domain-dependent algorithm and it emphasizes more on the analysis of the attributes of items in order to generate predictions. When documents such as web pages, publications and news are to be recommended, content-based filtering technique is the most successful. In content-based filtering technique, recommendation is made based on the user profiles using features extracted from the content of the items the user has evaluated in the past. Items that are mostly related to the positively rated items are recommended to the user. CBF uses different types of models to find similarity between documents in order to generate meaningful recommendations. It could use Vector Space Model such as Term Frequency Inverse Document Frequency (TF/IDF) or Probabilistic models such as Naïve Bayes Classifier, Decision Trees or Neural Networks to model the relationship between different documents within a corpus. These techniques make recommendations by learning the underlying model with either statistical analysis or machine learning techniques. Content-based filtering technique does not need the profile of other users since they do not influence recommendation. Also, if the user profile changes, CBF technique still has the potential to adjust its recommendations within a very short period of time. The major disadvantage of this technique is the need to have an in-depth knowledge and description of the features of the items in the profile.

Pros and Cons of content-based filtering techniques:

CB filtering techniques overcome the challenges of CF. They have the ability to recommend new items even if there are no ratings provided by users. So even if the database does not contain user preferences, recommendation accuracy is not affected. Also, if the user preferences change, it has the capacity to adjust its recommendations in a short span of time. They can manage situations where different users do not share the same items, but only identical items according to their intrinsic features. Users can get recommendations without sharing their profile, and this ensures privacy. CBF technique can also provide explanations on how recommendations are generated to users. However, the techniques suffer from various problems as discussed in the literature. Content based filtering techniques are dependent on items’ metadata. That is, they require rich description of items and very well organized user profile before recommendation can be made to users. This is called limited content analysis. So, the effectiveness of CBF depends on the availability of descriptive data. Content overspecialization is another serious problem of CBF technique. Users are restricted to getting recommendations similar to items already defined in their profiles.

So, we can say that collaborative and content based filtering techniques are distinct and it can be further made clear by the image below:

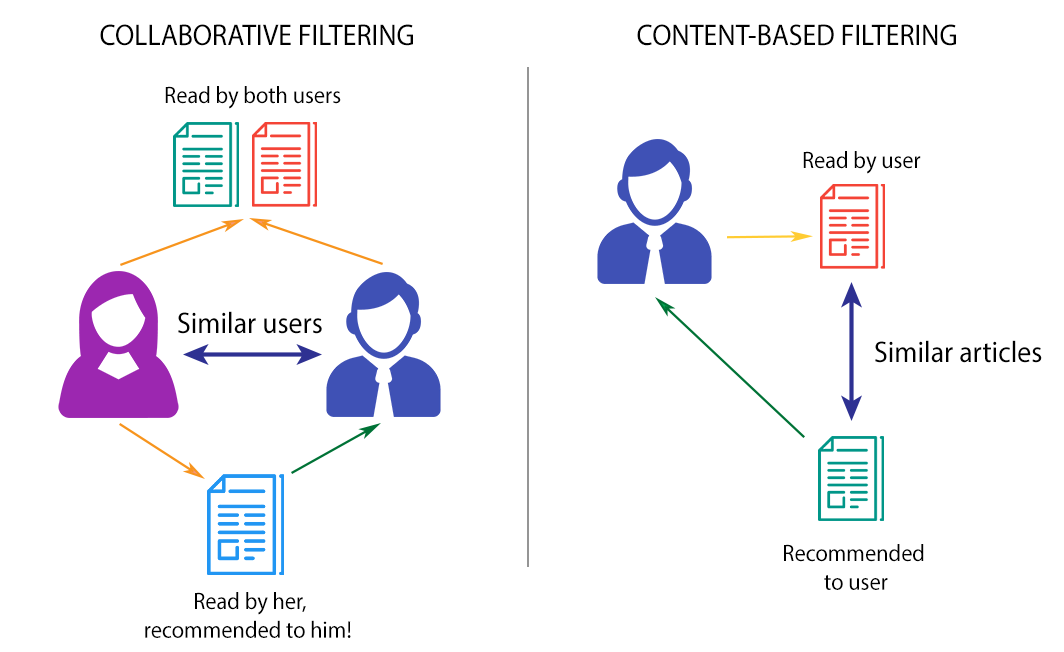


Fig 5. Differences between Collaborative and Content-based filtering

* **Hybrid Filtering:**

Hybrid filtering technique combines different recommendation techniques in order to gain better system optimization to avoid some limitations and problems of pure recommendation systems. The idea behind hybrid techniques is that a combination of algorithms will provide more accurate and effective recommendations than a single algorithm as the disadvantages of one algorithm can be overcome by another algorithm. Using multiple recommendation techniques can suppress the weaknesses of an individual technique in a combined model. The combination of approaches can be done in any of the following ways: separate implementation of algorithms and combining the result, utilizing some content-based filtering in collaborative approach, utilizing some collaborative filtering in content-based approach, creating a unified recommendation system that brings together both approaches.

A system that combines content-based filtering and collaborative filtering could take advantage from both the representation of the content as well as the similarities among users. Although there are several ways in which to combine the two techniques a distinction can be made between two basis approaches. A hybrid approach combines the two types of information while it is also possible to use the recommendations of the two filtering techniques independently.

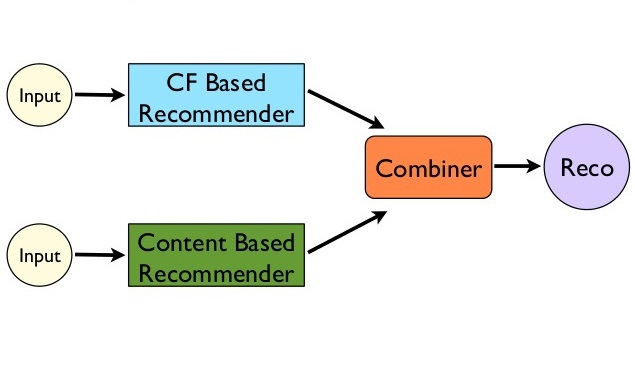


Fig 6. Hybrid filtering

**1.5 Scalability**

This is another problem associated with recommendation algorithms because computation normally grows linearly with the number of users and items. A recommendation technique that is efficient when the number of dataset is limited may be unable to generate satisfactory number of recommendations when the volume of dataset is increased. Thus, it is crucial to apply recommendation techniques which are capable of scaling up in a successful manner as the number of dataset in a database increases. Methods used for solving scalability problem and speeding up recommendation generation are based on Dimensionality reduction techniques, such as Singular Value Decomposition (SVD) method, which has the ability to produce reliable and efficient recommendations.

1. **FUNCTIONING OF THE WEBSITE**

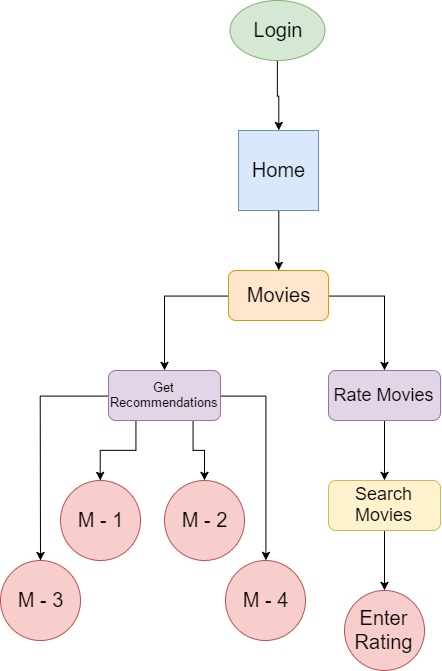
****

Fig 7. Website schemas

**2.1 Login Page**

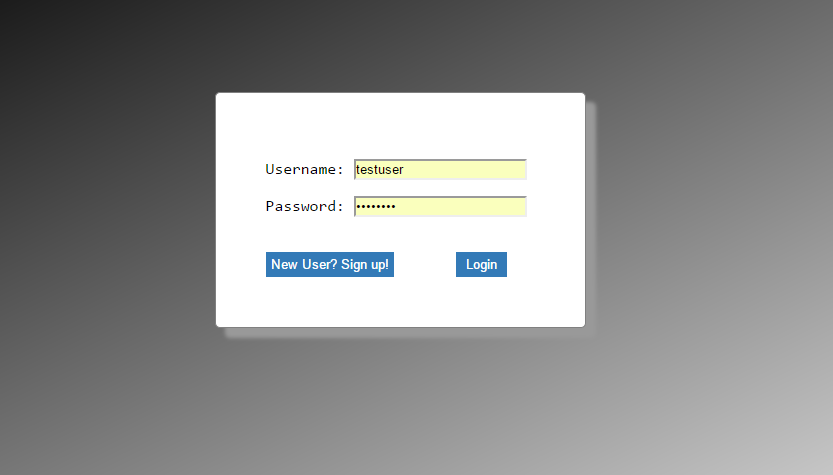


Fig 8. Login Page

1) The passwords are stored after passing them through MD5 hash function.

2) After the user enters the username and password, the entered password is passed through MD5 hash function.

3) The table “users” is queried for the username and the hashed password.

4) If a match is found, session variables for userid and username are set and the user is redirected to the home page.

5) If no match is found, an error is shown for the same.

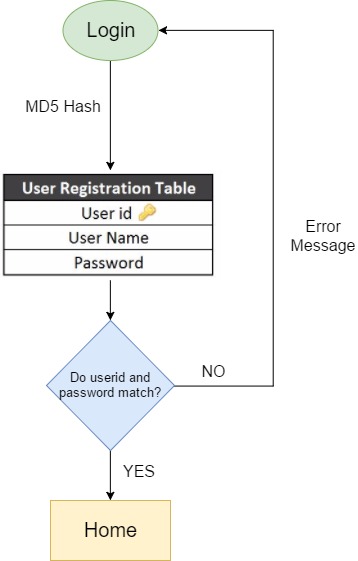
****

Fig 9. Login page flow diagram

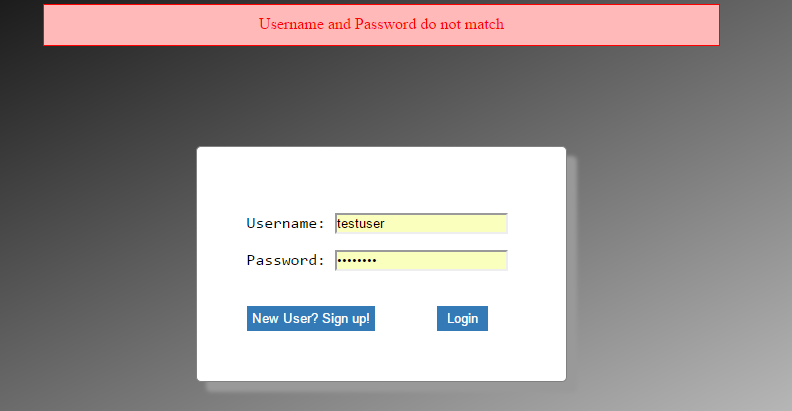


Fig 10. Username and Password mismatch error

**2.2 Registration Page**

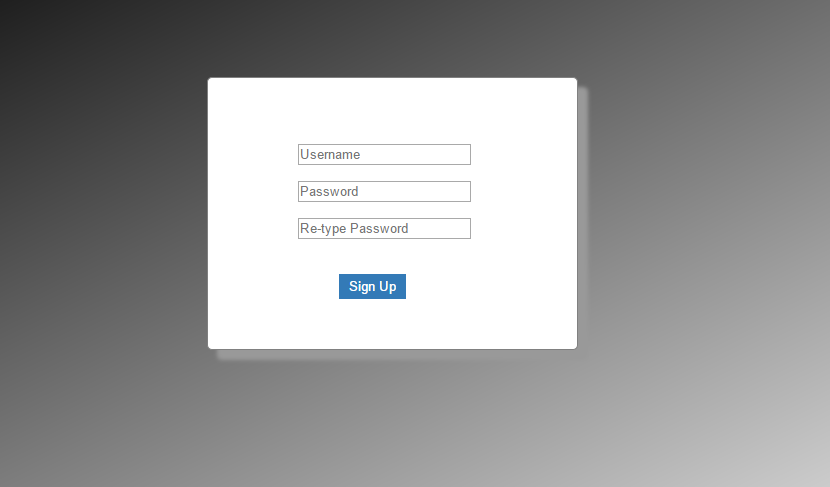


Fig 11. New user signup page

1) On the registeration page, the user enters his/her desired username and password. The password is to be typed twice to confirm and match.

2) When the user clicks the Sign Up button, the form is posted to the same page.

3) If the passwords don’t match, the session variable for message is set as “The two passwords do not match”. The password matching can be done using JavaScript dynamically too. That is a faster method than using php for the same.

4) If the length of the password is lesser than 8, the session variable for message is set as “The password length is less than 8”. We can also check for various parameters for the strength of the password like existence of at least one capital character, at least one special character and one number.

5) If the length of the username is lesser than 2, the session variable for message is set as “The Username is too small”.

6) If none of 3), 4) or 5) happen then a query is made to the “users” table to insert into it the username and MD5 hashed password. The userid is set as an Auto increment variable.

7) In the main body of the HTML page, we run another php snippet to check if the session variable for message is set or not. If it is set, then we show the error message accordingly.

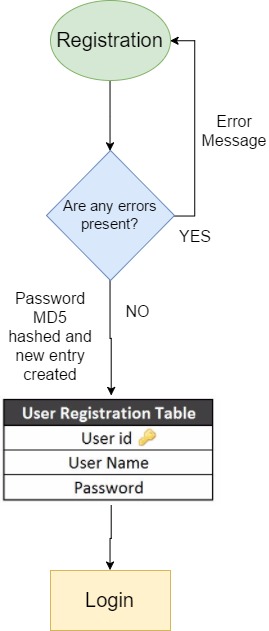


Fig 12. Registration page flow diagram

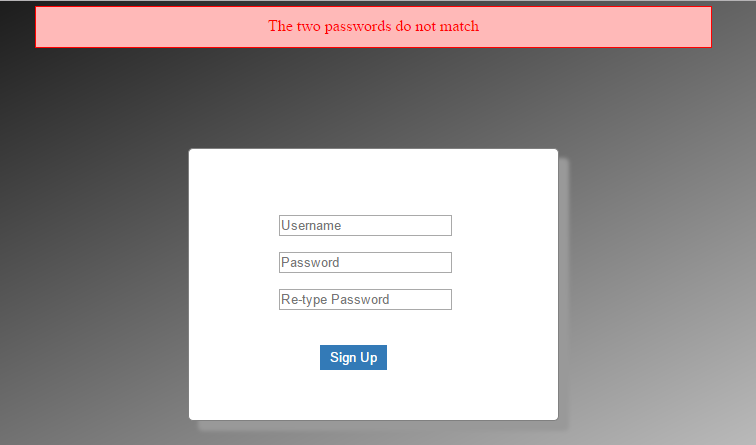


Fig 13. Password mismatch error

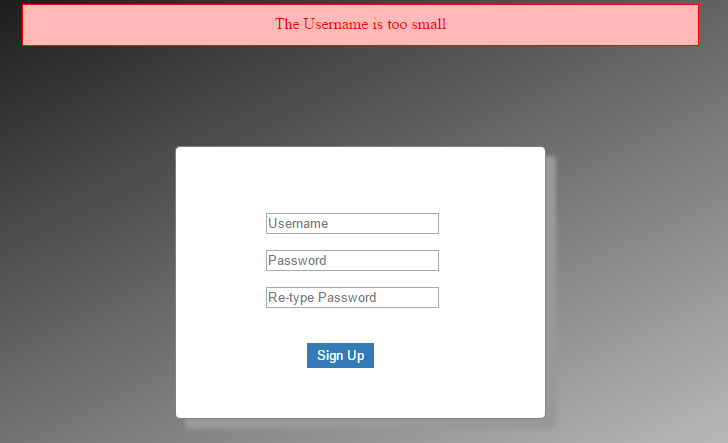


Fig 14. Short username error

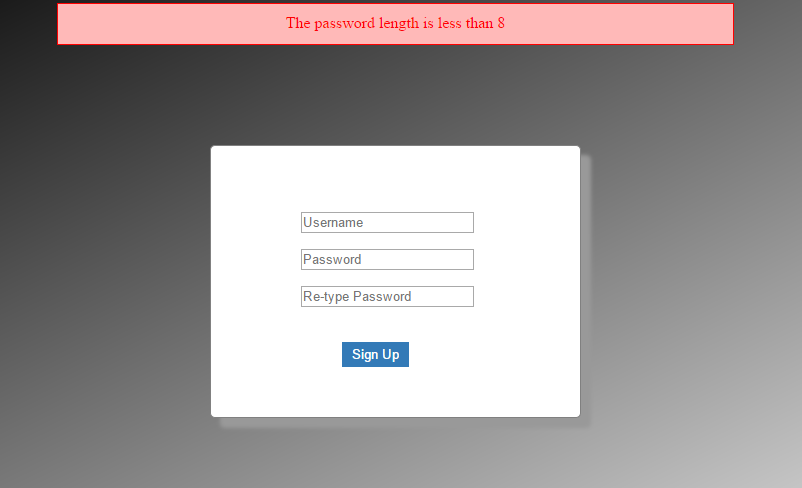


Fig 15. Insufficient password length error

**2.3 Movie Search Page**

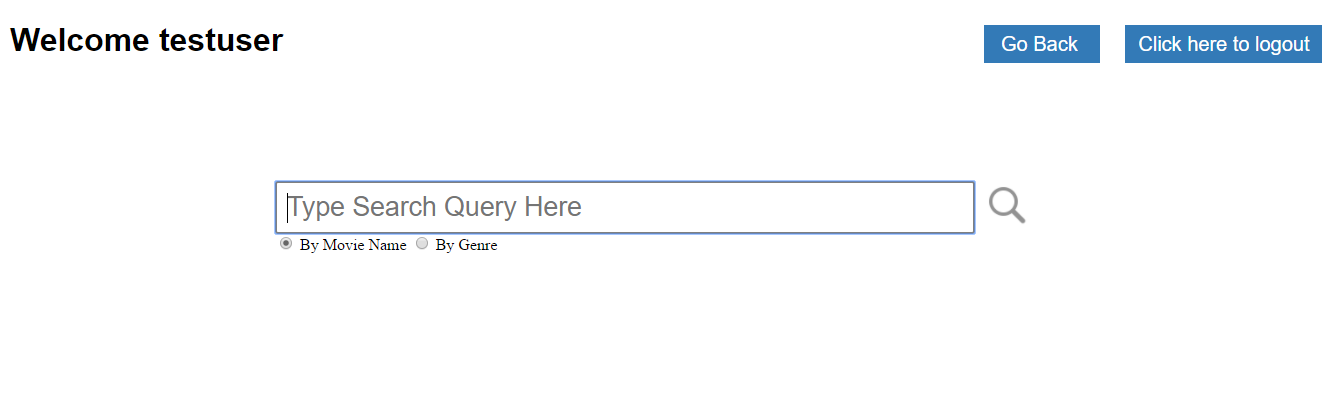


Fig 16. Movie Search Page

1) The user can type the name of the movie or part of it to search the movie database.

2) The search can also be done on the basis of the genre of the movie, for example: Comedy, Romance, etc.

3) The user enters his/her query into the search box and clicks on enter button or on the magnifying glass next to the seach box.

4) First of all, the length of the search query is checked and if it is too less then a JavaScript Alert pops up telling the user the same.

5) If the search query is greater than the required length and the user had selected search by name then a query is made to the table “moviedata” asking for all the movies whose name has the present search query as a substring. The search results are returned sorted in decreasing ordering of their average ratings and in case of a tie by the number of users who have rated.

6) If the search query is greater than the required length and the user had selected search by genre then a query is made to the table “moviedata” asking for all the movies whose genre is the same as that in the search query. The search results are returned sorted in decreasing ordering of their average ratings and in case of a tie by the number of users who have rated.

7) The filetered and sorted search results are printed via php as a HTML table.

8) If the user has already rated any of the returned movies, that movie name is shown in green otherwise it is shown in red.

9) The movie names are hyperlinks, clicking on which will lead the user to a page where he or she can add ratings for a movie.

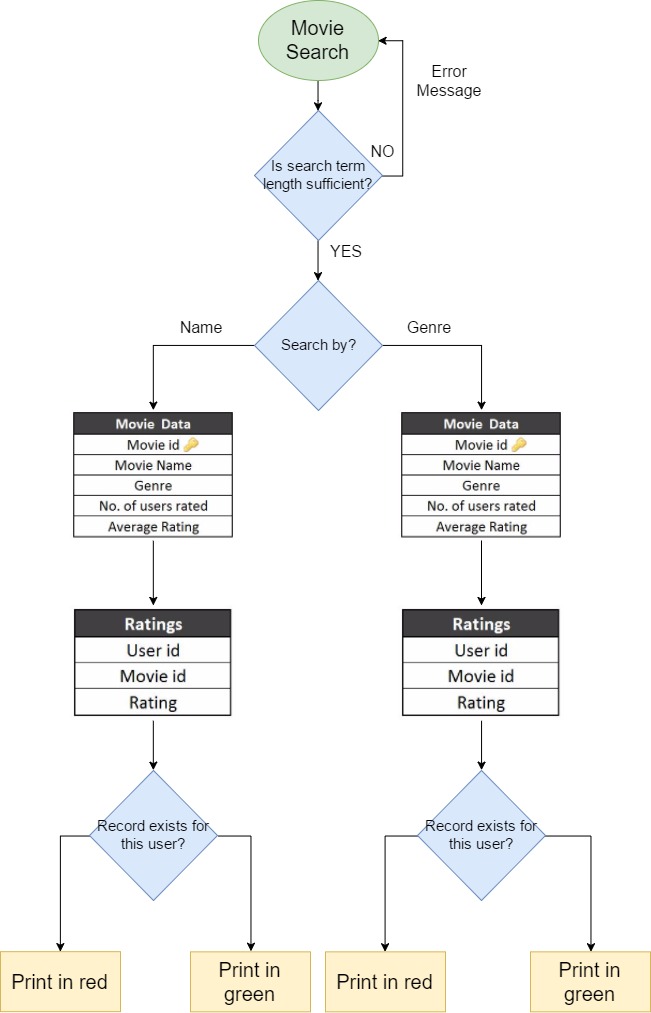


Fig 17. Search page flow diagram

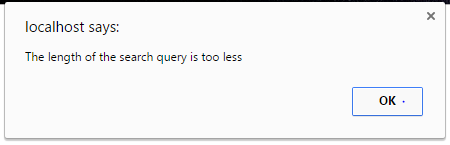


Fig 18. Error prompt when seearch query is small

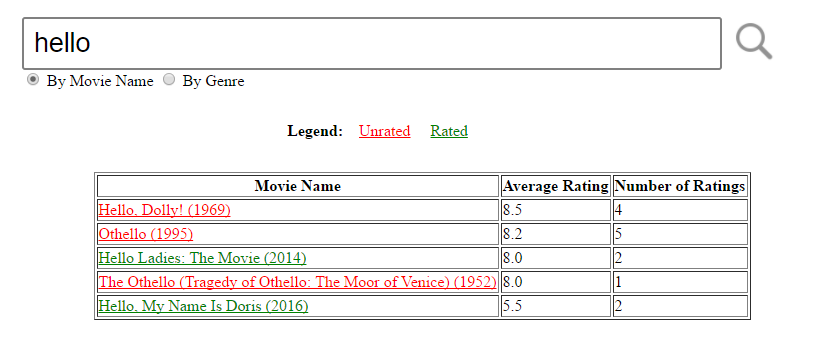


Fig 19. Search results in green and red

**2.4 Movie rating page**

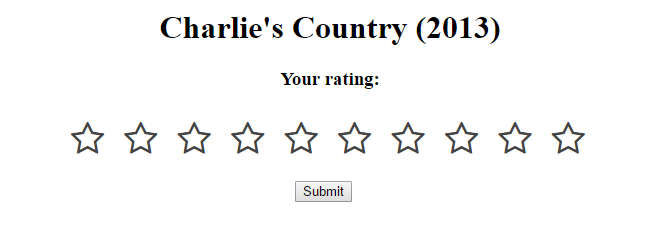


Fig 20. Movie rating page

1) When the user clicks on any movie from the search results, he or she is directed to a page where the user can modify (if already rated by the user) or enter his or her rating for that movie.

2) There are ten stars which change colour when hovered upon. The ratings work from the integers 1 to 10. Once the user clicks on a star and clicks on submit, his rating is entered. The stars can be thought of as radio buttons. If the user selects the xth star, it means the user wants to set the rating for this movie as x.

3) The user can change his or her rating multiple times by clicking on any star and only the final click will be counted for change.

4) The user rating is taken and a query is made to the “ratings” table to insert a new row.

5) The new row is inserted into the “ratings” table and then a query is made to the “moviedata” table to modify the average rating and the number of users rated.

6) Once the two values are modified in the “moviedata” table, the tab is killed and the user is now back to the search result page.

7) The user can now reload the search results and see that his or her latest rated movie is now shown in green colour if it was red before.



Fig 21. 10- star rating given for movie

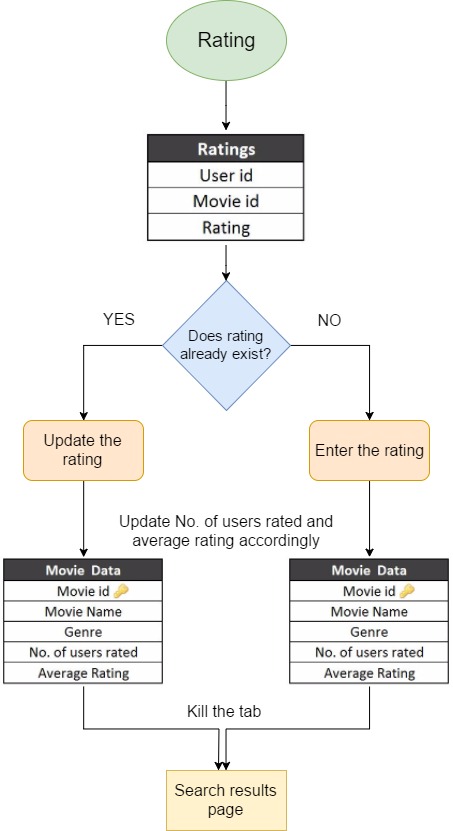


Fig 22. Rating page flow diagram

**2.5 Genre based Recommender system**

1) This method is based on the genres of the movies that the user has rated.

2) The first query is made to the table “ratings” to find which movies has the user rated.

3) Once the movieids are known, queries are made to the table “moviedata” to find what genres these movies are.

4) The genre attribute for the table “moviedata” actually stores one single string with consecutive genres separated by a ‘|’.

5) We make an associative array in php where the key is the genre name and the value is the count of the occurrence of the genre name in all the movies rated by the users. The associative array can be seen a function which maps keys to values.

6) We decide upon how many top values we want to display for each genre. For our implementation, we decided to display top 3 matches for each genre.

7) We iterate over each genre type that the user has rated a movie for.

8) Now a query is made to the table “moviedata” for the movieid, moviename, average rating and genre selected from 6) of all the movies with more than a threshold number of ratings.

9) For each movie that 7) returns, we make a sorted insert into the three average rating values we decided upon. We keep track of the names of the movies too.

10) We print the results in sorted order for all the genres the user has rated for.

11) One thing to note here is that instead of iterating over all the genres for which the user has rated a movie, we can decide on a threshold value x. We iterate over only those genres for which the user has at least an average rating of x. This makes sense because the user might have negatively rated movies of a particular genre.

12) This method is surprisingly fast and works well since we are not looking at all the movies but only of those genres that are relevant to the users.

Code:

$sql1 = "SELECT movieid FROM ratings WHERE userid='**$userid**' ";

$result1 = $conn->query($sql1);

$row1 = **mysqli\_fetch\_array**($result1);

$genres = **array**();

$movies = **array**();

$b = $row1["movieid"];

**array\_push**($movies,$row1["movieid"]);

$sql2 = "SELECT genre FROM moviedata WHERE movieid='**$b**' ";

$result2 = $conn->query($sql2);

$row2 = **mysqli\_fetch\_array**($result2);

$c = $row2["genre"];

$temp = "";

**for**($x = 0; $x < **strlen**($c); $x++){

**if**($c[$x]=='|'){

**if**(**array\_key\_exists**($temp, $genres)){

$genres[$temp]++;

}

**else**{

$genres[$temp]=1;

}

$temp = "";

}

**else**{

$temp = $temp.$c[$x];

}

}

**if**(**strlen**($temp)){

**if**(**array\_key\_exists**($temp, $genres)){

$genres[$temp]++;

}

**else**{

$genres[$temp]=1;

}

$temp = "";

}

**while**($row1 = $result1->fetch\_assoc()) {

$b = $row1["movieid"];

$sql2 = "SELECT genre FROM moviedata WHERE movieid='**$b**' ";

$result2 = $conn->query($sql2);

$row2 = **mysqli\_fetch\_array**($result2);

$c = $row2["genre"];

$temp = "";

**array\_push**($movies,$row1["movieid"]);

**for**($x = 0; $x < **strlen**($c); $x++){

**if**($c[$x]=='|'){

**if**(**array\_key\_exists**($temp, $genres)){

$genres[$temp]++;

}

**else**{

$genres[$temp]=1;

}

$temp = "";

}

**else**{

$temp = $temp.$c[$x];

}

}

**if**(**strlen**($temp)){

**if**(**array\_key\_exists**($temp, $genres)){

$genres[$temp]++;

}

**else**{

$genres[$temp]=1;

}

$temp = "";

}

}

**arsort**($genres);

**foreach** ($genres **as** $key => $value) {

**echo** "Genre: **$key**"."<br>";

$a = 0;

$aname = "";

$b = 0;

$bname = "";

$c = 0;

$cname = "";

$sql3 = "SELECT movieid, moviename, genre, avg\_rating FROM moviedata WHERE no\_of\_users\_rated > 10";

$result3 = $conn->query($sql3);

$row3 = **mysqli\_fetch\_array**($result3);

**if** (**strpos**($row3["genre"],$key) !== **false** && **in\_array**($row3["movieid"], $movies) === **false**){

**if**($row3["avg\_rating"] > $a){

$c = $b;

$cname = $bname;

$b = $a;

$bname = $aname;

$a = $row3["avg\_rating"];

$aname = $row3["moviename"];

}

**else** **if**($row3["avg\_rating"] > $b){

$c = $b;

$cname = $bname;

$b = $row3["avg\_rating"];

$bname = $row3["moviename"];

}

**else** **if**($row3["avg\_rating"] > $c){

$c = $row3["avg\_rating"];

$cname = $row3["moviename"];

}

}

**while**($row3 = $result3->fetch\_assoc()){

**if** (**strpos**($row3["genre"],$key) !== **false** &&**in\_array**($row3["movieid"], $movies) === **false**){

**if**($row3["avg\_rating"] > $a){

$c = $b;

$cname = $bname;

$b = $a;

$bname = $aname;

$a = $row3["avg\_rating"];

$aname = $row3["moviename"];

}

**else** **if**($row3["avg\_rating"] > $b){

$c = $b;

$cname = $bname;

$b = $row3["avg\_rating"];

$bname = $row3["moviename"];

}

**else** **if**($row3["avg\_rating"] > $c){

$c = $row3["avg\_rating"];

$cname = $row3["moviename"];

}

}

}

//echo $key."<br>";

//echo $value."<br>";

**echo** $aname." ".$a."<br>";

**echo** $bname." ".$b."<br>";

**echo** $cname." ".$c."<br>";

**echo** "<br>";

}

**2.6 Collaborative Filtering**

1) This method is based on user – user collaborative filtering. The basic idea behind this is that if two users have agreed on something in the past, then they will agree on things in the future too. For the current user, we find the most similar user. The most similar user is the user whose ratings are closest to our current user.

2) For this we define a similarity quotient. For our implementation, we have used the reciprocal of the Eucledian distance between the ratings vector of the two users for comparison.

3) For our current user, we store the movieids and the corresponding ratings in an two arrays. We also need a marked array to know which movies for our current user have been matched in any one iteration of the for loop.

4) We initialise max\_similarity and max\_similarity\_user as -1. After the full iteration, these will contain the maximum value of the similarity quotient and the userid of the user with which this similarity was achieved.

5) We loop over all the users and make a query to the table “ratings” to get the movieids and ratings for the current user.

6) We find the Eucledian distance between the rating arrays of the two users.

7) If a movie hasn’t been rated by a user, we take his or her rating for that movie to be 0.

8) After using the marked array to check for movies rated by the current user which didn’t find any match in this iteration, we update max\_similarity and max\_similarity\_user whenever needed.

9) In the end, we show the user those movies of the max\_similarity\_user which he or she hasn’t yet rated.

10) The time complexity of this method is O(n\*m\*t) where n is the number of users, m is the number of movies rated by our current user and t is the average number of movies rated by each user.

Code:

$start = **microtime**(**true**);

$sql1 = "SELECT movieid,rating FROM ratings WHERE userid='**$userid**' ";

$result1 = $conn->query($sql1);

$movies = **array**();

$ratings = **array**();

**while**($row1=**mysqli\_fetch\_assoc**($result1)){

**array\_push**($movies,$row1["movieid"]);

**array\_push**($ratings,$row1["rating"]);

}

$max\_similarity = -1;

$max\_similarity\_user = -1;

**for**($comp\_user = 1; $comp\_user<=671; $comp\_user++){

**if**($comp\_user!=$userid){

$sql2 = "SELECT movieid,rating FROM ratings WHERE userid='**$comp\_user**' ";

$result2 = $conn->query($sql2);

$marked = **array\_fill**(0,**count**($movies),0);

$score = 0;

**while**($row2=**mysqli\_fetch\_assoc**($result2)){

**if**(**in\_array**($row2["movieid"], $movies)){

$index = -1;

**for**($x = 0; $x < **count**($movies); $x++){

**if**($movies[$x]==$row2["movieid"]){

$index = $x;

**break**;

}

}

$score = $score + ($row2["rating"]-$ratings[$index])\*($row2["rating"]-$ratings[$index]);

//$score = $score + abs($row2["rating"]-$ratings[$index]);

$marked[$index] = 1;

}

**else**{

$score = $score + ($row2["rating"])\*($row2["rating"]);

//$score = $score + ($row2["rating"]);

}

}

**for**($y = 0; $y < **count**($movies); $y++){

**if**($marked[$y]==0){

$score = $score + ($ratings[$y])\*($ratings[$y]);

//$score = $score + ($ratings[$y]);

}

}

$score = 1/(**sqrt**($score) + 1);

**if**($score > $max\_similarity){

$max\_similarity = $score;

$max\_similarity\_user = $comp\_user;

}

}

}

**echo** $max\_similarity."<br>";

**echo** $max\_similarity\_user."<br>";

$sql3 = "SELECT movieid FROM ratings WHERE userid='**$max\_similarity\_user**' ";

$result3 = $conn->query($sql3);

$max\_similarity\_user\_array = **array**();

**while**($row3=**mysqli\_fetch\_assoc**($result3)){

**if**(**in\_array**($row3["movieid"], $movies) === **false**){

**array\_push**($max\_similarity\_user\_array, $row3["movieid"]);

}

}

**for**($x = 0; $x < **count**($max\_similarity\_user\_array); $x++){

$sql4="SELECT moviename,avg\_rating FROM moviedata WHERE movieid='**$max\_similarity\_user\_array**[**$x**]'";

$result4 = $conn->query($sql4);

**while**($row4=**mysqli\_fetch\_assoc**($result4)){

**echo** $row4["moviename"]." ".$row4["avg\_rating"]."<br>";

}

}

$time\_elapsed\_secs = **microtime**(**true**) - $start;

**echo** $time\_elapsed\_secs;

**2.7 Modified User- user Collaborative Filtering**

1) This method is an improvement over method 2 on the basis of implementation.

2) In method 2, we made two arrays for the current user: one to store movieids and the other to store the corresponding ratings.

3) In each iteration of the loop over all the users, we first found out which movies the user had rated and then searched for that movieid in the array of our current user.

4) If a match existed, we computed the distance between these two and squared it.

5) If a match didn’t exist, we let the rating of current user on this movie to be 0.

6) This led to O(m) extra computation in each iteration. Since we had to check if the movie existed in the movie array of the current user and decide accordingly.

7) Now what we can do is to use an associative array for the current user. The key is the movieid and the value is his or her rating on that movie.

8) In this way, we can access any index in essentially O(1) time on average. The time taken to search for the match in current user’s array is lessened and the time complexity of the algorithm improves by a factor of m.

9) In the end, we show the user those movies of the max\_similarity\_user which he or she hasn’t yet rated.

10) The time complexity of this method is O(n\*t) where n is the number of users and t is the average number of movies rated by each user.

Code:

$start = **microtime**(**true**);

//echo "3";

$sql1 = "SELECT movieid,rating FROM ratings WHERE userid='**$userid**' ";

$result1 = $conn->query($sql1);

$movies = **array**();

**while**($row1=**mysqli\_fetch\_assoc**($result1)){

$movies[$row1["movieid"]] = $row1["rating"];

}

$max\_similarity = -1;

$max\_similarity\_user = -1;

**for**($comp\_user = 1; $comp\_user<=671; $comp\_user++){

**if**($comp\_user!=$userid){

$sql2 = "SELECT movieid,rating FROM ratings WHERE userid='**$comp\_user**' ";

$result2 = $conn->query($sql2);

$marked = **array**();

$score = 0;

**while**($row2=**mysqli\_fetch\_assoc**($result2)){

**if**(**array\_key\_exists**($row2["movieid"],$movies)){

//$score = $score + ($row2["rating"]-$movies[$row2["movieid"]])\*($row2["rating"]-$movies[$row2["movieid"]]);

$score = $score + ($row2["rating"]-$movies[$row2["movieid"]]);

$marked[$row2["movieid"]] = 1;

}

**else**{

//$score = $score + ($row2["rating"])\*($row2["rating"]);

$score = $score + ($row2["rating"]);

}

}

/\*

foreach ($genres as $key => $value) {

echo "Genre: $key"."<br>";

\*/

**foreach** ($movies **as** $key => $value){

**if**(**array\_key\_exists**($key,$marked)==**false**){

//$score = $score + ($value)\*($value);

$score = $score + ($value);

}

}

//$score = 1/(sqrt($score) + 1);

$score = 1/($score + 1);

**if**($score > $max\_similarity){

$max\_similarity = $score;

$max\_similarity\_user = $comp\_user;

}

}

}

**echo** $max\_similarity."<br>";

**echo** $max\_similarity\_user."<br>";

$sql3 = "SELECT movieid FROM ratings WHERE userid='**$max\_similarity\_user**' ";

$result3 = $conn->query($sql3);

$max\_similarity\_user\_array = **array**();

**while**($row3=**mysqli\_fetch\_assoc**($result3)){

**if**(**array\_key\_exists**($row3["movieid"],$movies)==**false**){

**array\_push**($max\_similarity\_user\_array, $row3["movieid"]);

}

}

**for**($x = 0; $x < **count**($max\_similarity\_user\_array); $x++){

$sql4="SELECT moviename,avg\_rating FROM moviedata WHERE movieid='**$max\_similarity\_user\_array**[**$x**]'";

$result4 = $conn->query($sql4);

**while**($row4=**mysqli\_fetch\_assoc**($result4)){

**echo** $row4["moviename"]." ".$row4["avg\_rating"]."<br>";

}

}

$time\_elapsed\_secs = **microtime**(**true**) - $start;

**echo** $time\_elapsed\_secs;

**2.8 Modified Collaborative Filtering**

1) This method is an improvement on method 3.

2) We don’t iterated over all the users in our database but only on those who have rated at least one such movie which has been rated by out current user.

3) This decreases the number of iterations on the user dataset and thus the time complexity is now lesser than O(n\*t) where n is the number of users and t is the average number of movies rated by each user.

4) To keep track of which users have at least one movie in common with our current user, we maintain a boolean array called relevant\_array and before we calculate the similarity scores, we fill this array.

5) To fill relevant\_array optimally, we use all the movies rated by the current user and find out which other users have rated these. For each such user, we set relevant\_user as true.

6) We are kind of taking the union of the users who have rated at least one movie rated by our current user.

7) There can be further improvements to this algorithm if the number of users increases by a lot. We can use kNN (k Nearest Neighbours) instead of iterating over all the users.

Code:

$start = **microtime**(**true**);

//echo "3";

$sql1 = "SELECT movieid,rating FROM ratings WHERE userid='**$userid**' ";

$result1 = $conn->query($sql1);

$movies = **array**();

**while**($row1=**mysqli\_fetch\_assoc**($result1)){

$movies[$row1["movieid"]] = $row1["rating"];

}

$relevant\_users = **array**();

**for**($x = 0; $x <= 671; $x++){

$relevant\_users[$x]=0;

}

**foreach** ($movies **as** $key => $value) {

$sql4 = "SELECT userid FROM ratings WHERE movieid='**$key**' ";

$result4 = $conn->query($sql4);

**while**($row4=**mysqli\_fetch\_assoc**($result4)){

$relevant\_users[$row4["userid"]]=1;

}

}

$max\_similarity = -1;

$max\_similarity\_user = -1;

**for**($t = 1; $t<=671; $t++){

**if**($relevant\_users[$t]==1 && $t!=$userid){

$sql2 = "SELECT movieid,rating FROM ratings WHERE userid='**$t**' ";

$result2 = $conn->query($sql2);

$marked = **array**();

$score = 0;

**while**($row2=**mysqli\_fetch\_assoc**($result2)){

**if**(**array\_key\_exists**($row2["movieid"],$movies)){

//$score = $score + ($row2["rating"]-$movies[$row2["movieid"]])\*($row2["rating"]-$movies[$row2["movieid"]]);

$score = $score + ($row2["rating"]-$movies[$row2["movieid"]]);

$marked[$row2["movieid"]] = 1;

}

**else**{

//$score = $score + ($row2["rating"])\*($row2["rating"]);

$score = $score + ($row2["rating"]);

}

}

/\*

foreach ($genres as $key => $value) {

echo "Genre: $key"."<br>";

\*/

**foreach** ($movies **as** $key => $value){

**if**(**array\_key\_exists**($key,$marked)==**false**){

//$score = $score + ($value)\*($value);

$score = $score + ($value);

}

}

//$score = 1/(sqrt($score) + 1);

$score = 1/($score + 1);

**if**($score > $max\_similarity){

$max\_similarity = $score;

$max\_similarity\_user = $t;

}

}

}

**echo** $max\_similarity."<br>";

**echo** $max\_similarity\_user."<br>";

$sql3 = "SELECT movieid FROM ratings WHERE userid='**$max\_similarity\_user**' ";

$result3 = $conn->query($sql3);

$max\_similarity\_user\_array = **array**();

**while**($row3=**mysqli\_fetch\_assoc**($result3)){

**if**(**array\_key\_exists**($row3["movieid"],$movies)==**false**){

**array\_push**($max\_similarity\_user\_array, $row3["movieid"]);

}

}

**for**($x = 0; $x < **count**($max\_similarity\_user\_array); $x++){

$sql4="SELECT moviename,avg\_rating FROM moviedata WHERE movieid='**$max\_similarity\_user\_array**[**$x**]'";

$result4 = $conn->query($sql4);

**while**($row4=**mysqli\_fetch\_assoc**($result4)){

**echo** $row4["moviename"]." ".$row4["avg\_rating"]."<br>";

}

}

$time\_elapsed\_secs = **microtime**(**true**) - $start;

**echo** $time\_elapsed\_secs;

1. **REFERENCES**

* J.B. Schafer, J.A. Konstan, and J. Reidl, “E-Commerce Recommendation Applications,” Data Mining and Knowledge Discovery, Kluwer Academic, 2001, pp. 115-153.
* P. Resnick et al., “GroupLens: An Open Architecture for Collaborative Filtering of Netnews,” Proc. ACM 1994 Conf. Computer Supported Cooperative Work, ACM Press, 1994, pp. 175-186.
* J. Breese, D. Heckerman, and C. Kadie, “Empirical Analysis of Predictive Algorithms for Collaborative Filtering,” Proc. 14th Conf. Uncertainty in Artificial Intelligence, Morgan Kaufmann, 1998, pp. 43-52.
* B.M. Sarwarm et al., “Analysis of Recommendation Algorithms for E-Commerce,” ACM Conf. Electronic Commerce, ACM Press, 2000, pp.158-167.
* K. Goldberg et al., “Eigentaste: A Constant Time Collaborative Filtering Algorithm,” Information Retrieval J., vol. 4, no. 2, July 2001, pp. 133-151.
* P.S. Bradley, U.M. Fayyad, and C. Reina, “Scaling Clustering Algorithms to Large Databases,” Knowledge Discovery and Data Mining, Kluwer Academic, 1998, pp. 9-15.
* L. Ungar and D. Foster, “Clustering Methods for Collaborative Filtering,” Proc. Workshop on Recommendation Systems, AAAI Press, 1998.
* M. Balabanovic and Y. Shoham, “Content-Based Collaborative Recommendation,” Comm. ACM, Mar. 1997, pp. 66-72.
* G.D. Linden, J.A. Jacobi, and E.A. Benson, Collaborative Recommendations Using Item-to-Item Similarity Mappings, US Patent 6,266,649 (to Amazon.com), Patent and Trademark Office, Washington, D.C., 2001.
* B.M. Sarwar et al., “Item-Based Collaborative Filtering Recommendation Algorithms,” 10th Int’l World Wide Web Conference, ACM Press, 2001, pp. 285-295

1. **APPENDIX**

**4.1 logout.php**

<?php

**session\_start**();

**session\_destroy**();

**header**('location:login.php');

?>

**4.2 home.php**<?php

**session\_start**();

**if**(**isset**($\_SESSION['userid'])){

**echo** "Login successful! Welcome ";

**echo** $\_SESSION['username'];

**echo**'<a href="movies.php"> Movies </a>';

**echo**'<a href="logout.php"> Click here to logout </a>';

}

**else**{

**echo**'Please login to view this page';

**echo**'<a href="login.php"> Click here to login </a>';

}

?>

<title>**Home**</title>

**4.3 login.php**<?php

**session\_start**();

**if**(**isset**($\_SESSION['userid'])){

**header**('location:home.php');

}

$conn = **new** mysqli("localhost", "root", "recommender", "majorproject");

**if**(**isset**($\_POST['btn']))

{

//Get values passed from form in login.php file

$username = $\_POST['user'];

$password = $\_POST['pass'];

//To prevent MySql injection

$username = **mysqli\_real\_escape\_string**($conn, $username);

$password = **mysqli\_real\_escape\_string**($conn, $password);

$password = **md5**($password);

// Check connection

**if** ($conn->connect\_error) {

**die**("Connection failed: \n" . $conn->connect\_error);

}

//Query the database for user

$sql = "SELECT \* FROM users WHERE username = '**$username**' AND password = '**$password**' ";

$result = $conn->query($sql);

$row = **mysqli\_fetch\_array**($result);

**if**($row['username'] == $username && $row['password'] == $password){

$\_SESSION['userid'] = $row['userid'];

$\_SESSION['username'] = $username;

**header**("location:home.php");

}

**else**{

$\_SESSION['message']="Username and Password do not match";

**}**

**$conn->close();**

**}**

**?>**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>**

**login page**

**</title>**

**<link rel="stylesheet" type="text/css" href="login\_style.css">**

**</head>**

**<body>**

**<?php**

**if(isset($\_SESSION['message']))**

**{**

**echo "<div id='error\_msg'>".$\_SESSION['message']."</div>";**

**unset($\_SESSION['message']);**

**}**

**?>**

**<div id="frm">**

**<form action="login.php" method="POST">**

**<p>**

**<label>Username: </label>**

**<input type="text" id="user" name="user" />**

**</p>**

**<p>**

**<label>Password: </label>**

**<input type="password" id="pass" name="pass" />**

**</p>**

**<br>**

**<div>**

**<a href="register.php" class="button">New User? Sign up!</a>**

**<input type="submit" id="btn" name = "btn" value="Login" />**

**</div>**

**</form>**

**</div>**

**</body>**

**</html>**

**4.4 no\_of\_users\_rated.php**<?php

$conn = **new** mysqli("localhost", "root", "recommender", "majorproject");

//https://www.w3schools.com/sql/sql\_alter.asp

$sql = "SELECT \* FROM ratings";

$result = $conn->query($sql);

$row = **mysqli\_fetch\_array**($result);

$rating\_count = **array**();

$rating\_count = **array\_fill**(0,164980, 0);

$rating\_count[$row["movieid"]]++;

**while**($row = $result->fetch\_assoc())

{

$rating\_count[$row["movieid"]]++;

}

**for**($y = 1; $y<=100; $y++)

{

**echo** $rating\_count[$y]." ";

}

**for**($x = 1; $x<164890; $x++)

{

$sql1 = "UPDATE moviedata SET no\_of\_users\_rated = '**$rating\_count**[**$x**]' WHERE movieid = '**$x**' ";

//$sql1 = "INSERT INTO moviedata (no\_of\_users\_rated) VALUES ($rating\_count[$x]) WHERE movieid = '$x' ";

**if**($conn->query($sql1) == **TRUE**)

**echo** $x." ";

}

$conn->close();

?>

**4.5 avg\_rating.php**<?php

$conn = **new** mysqli("localhost", "root", "recommender", "majorproject");

$sql = "SELECT movieid, rating FROM ratings";

$result = $conn->query($sql);

$row = **mysqli\_fetch\_array**($result);

$rating\_sum = **array**();

$rating\_sum = **array\_fill**(0,164980, 0);

$rating\_sum[$row["movieid"]]+=$row["rating"];

**while**($row = $result->fetch\_assoc())

{

$rating\_sum[$row["movieid"]]+=$row["rating"];

}

**for**($x = 1; $x<164890; $x++)

{

$temp = $rating\_sum[$x];

$sql2 = "SELECT no\_of\_users\_rated FROM moviedata WHERE movieid = '**$x**'";

$result = $conn->query($sql2);

$row = **mysqli\_fetch\_array**($result);

**if**($row['no\_of\_users\_rated']!=0 && $conn->query($sql2)==**TRUE**)

{

$temp = 1.0\*$temp/$row['no\_of\_users\_rated'];

$sql3 = "UPDATE moviedata SET avg\_rating = '**$temp**' WHERE movieid = '**$x**' ";

//$sql1 = "INSERT INTO moviedata (no\_of\_users\_rated) VALUES ($rating\_count[$x]) WHERE movieid = '$x' ";

**if**($conn->query($sql3) == **TRUE**)

**echo** $x." ";

}

}

$conn->close();

?>

**4.6 enter\_rating.php**<?php

**session\_start**();

**if** (**isset**($\_POST['btn'])){

//$\_SESSION['enter\_rating\_id'] is the movie id

$rating = -1;

**if**($\_POST['star']=='star-10'){$rating = 10;}

**else** **if**($\_POST['star']=='star-9'){$rating = 9;}

**else** **if**($\_POST['star']=='star-8'){$rating = 8;}

**else** **if**($\_POST['star']=='star-7'){$rating = 7;}

**else** **if**($\_POST['star']=='star-6'){$rating = 6;}

**else** **if**($\_POST['star']=='star-5'){$rating = 5;}

**else** **if**($\_POST['star']=='star-4'){$rating = 4;}

**else** **if**($\_POST['star']=='star-3'){$rating = 3;}

**else** **if**($\_POST['star']=='star-2'){$rating = 2;}

**else** **if**($\_POST['star']=='star-1'){$rating = 1;}

**echo** $rating;

//echo $\_SESSION["userid"];

//echo $\_SESSION["enter\_rating\_id"];

$x = $\_SESSION["userid"];

$y = $\_SESSION["enter\_rating\_id"];

$conn = **new** mysqli("localhost","root","recommender","majorproject");

$sql2 = "INSERT INTO ratings (userid,movieid,rating) VALUES ('**$x**','**$y**','**$rating**')";

**if**($conn->query($sql2)===**TRUE**){

//echo "nice1";

}

$sql3 = " SELECT no\_of\_users\_rated, avg\_rating FROM moviedata WHERE movieid = '**$y**' ";

$result = $conn->query($sql3);

$row = **mysqli\_fetch\_array**($result);

$a = $row["no\_of\_users\_rated"];

$b = $row["avg\_rating"];

$b = ( 1.0\*($row["avg\_rating"])\*($row["no\_of\_users\_rated"]) + $rating)/($row["no\_of\_users\_rated"] + 1.0);

$a = $a + 1;

**$sql4 = "UPDATE moviedata SET no\_of\_users\_rated = '$a', avg\_rating = '$b' WHERE movieid = '$y' ";**

**if($conn->query($sql4)===TRUE){**

**//echo "nice2";**

**}**

**$conn->close();**

**echo "<script>window.close();</script>";**

**}**

**unset($\_SESSION['enter\_rating\_id']);**

**?>**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>Enter Rating</title>**

**<link rel="stylesheet" type="text/css" href="enter\_rating\_style.css">**

**<link rel="stylesheet" href="//netdna.bootstrapcdn.com/font-awesome/4.2.0/css/font-awesome.min.css">**

**</head>**

**<body>**

**<?php**

**$url = $\_SERVER['REQUEST\_URI'];**

**$split = -1;**

**for($x = 0; $x < strlen($url); $x++){**

**if($url[$x]=='='){**

**$split = $x;**

**break;**

**}**

**}**

**$query\_id = "";**

**for($x = $split+1; $x < strlen($url); $x++){**

**$query\_id = $query\_id.$url[$x];**

**}**

**$\_SESSION['enter\_rating\_id'] = $query\_id;**

**$conn = new mysqli("localhost","root","recommender","majorproject");**

**$sql = "SELECT moviename FROM moviedata WHERE movieid = $query\_id";**

**$result = $conn->query($sql);**

**$row = mysqli\_fetch\_array($result);**

**echo "<h1 align = center>";**

**echo $row["moviename"];**

**echo "</h1>";**

**echo "<h3 align = center>";**

**echo "Your rating: ";**

**echo "</h3>";**

**$conn->close();**

**?>**

**<div class="stars" style="margin:auto; width:940px;">**

**<form action="enter\_rating.php" method="POST">**

**<input class="star star-5" id="star-10" type="radio" value="star-10" name="star"/>**

**<label class="star star-5" for="star-10"></label>**

**<input class="star star-4" id="star-9" type="radio" value="star-9" name="star"/>**

**<label class="star star-4" for="star-9"></label>**

**<input class="star star-3" id="star-8" type="radio" value="star-8" name="star"/>**

**<label class="star star-3" for="star-8"></label>**

**<input class="star star-2" id="star-7" type="radio" value="star-7" name="star"/>**

**<label class="star star-2" for="star-7"></label>**

**<input class="star star-2" id="star-6" type="radio" value="star-6" name="star"/>**

**<label class="star star-2" for="star-6"></label>**

**<input class="star star-2" id="star-5" type="radio" value="star-5" name="star"/>**

**<label class="star star-2" for="star-5"></label>**

**<input class="star star-2" id="star-4" type="radio" value="star-4" name="star"/>**

**<label class="star star-2" for="star-4"></label>**

**<input class="star star-1" id="star-3" type="radio" value="star-3" name="star"/>**

**<label class="star star-1" for="star-3"></label>**

**<input class="star star-1" id="star-2" type="radio" value="star-2" name="star"/>**

**<label class="star star-1" for="star-2"></label>**

**<input class="star star-1" id="star-1" type="radio" value="star-1" name="star"/>**

**<label class="star star-1" for="star-1"></label>**

**<br><br><br><br>**

**<input style = "margin-left:640px;" type="submit" id="btn" name = "btn" value="Submit" />**

**</form>**

**</div>**

**</body>**

**</html>**

**4.7 movies.php**<?php

**session\_start**();

**if**(**empty**($\_SESSION['userid'])){

**header**('location:login.php');

}

?>

<html>

<head>

<link rel=**"stylesheet"** type=**"text/css"** href=**"movies\_style.css"**>

<title>

**Movies**

</title>

</head>

<body>

<header id=**"userheader"**>

<span class=**"box"** style=**" width: 320px; margin-right: 300px; margin-left: 10px;"**><?php **echo** '<h1 style="font-family: Arial;"> Welcome ' .$\_SESSION['username'].'</h1>' ?></span>

<span class = **"box1"**>

<span style=**"margin-right: 25px;"**><a href=**"recommendation.php"** id= **"rec"** style= **"color: #fff;**

**background: #337ab7;**

**padding: 8px 17px 8px 17px;**

**border: none;**

**text-decoration: none;**

**font-size: 15pt;**

**font-family: Arial;"**> **Get Recommendations** </a></span>

<span style=**"margin-right: 25px;"**><a href=**"ratemovies.php"** id= **"rate\_button"**> **Rate more movies** </a></span>

<span><a href=**"logout.php"** id= **"logout\_button"**> **Click here to logout** </a></span>

</span>

</header>

**<h1 style="text-align: center; font-family: Arial;">Movies you have already rated</h1>**

**<?php**

**$conn = new mysqli("localhost", "root", "recommender", "majorproject");**

**$user\_id\_query = $\_SESSION['userid'];**

**$sql = "SELECT \* FROM ratings WHERE userid = '$user\_id\_query'";**

**$result = $conn->query($sql);**

**$row = mysqli\_fetch\_array($result);**

**$movie\_array = array(); //movie names**

**$rating\_array = array(); //ratings for the movies**

**array\_push($movie\_array, $row["movieid"]);**

**array\_push($rating\_array, $row["rating"]);**

**while($row = $result->fetch\_assoc()) {**

**array\_push($movie\_array, $row["movieid"]);**

**array\_push($rating\_array, $row["rating"]);**

**}**

**//print\_r($movie\_array);**

**$arrlength = count($movie\_array);**

**for($x = 0; $x < $arrlength; $x++) {**

**$sql2 = "SELECT moviename FROM moviedata WHERE movieid = '$movie\_array[$x]'";**

**$result2 = $conn->query($sql2);**

**$row2 = mysqli\_fetch\_array($result2);**

**$movie\_array[$x] = $row2['moviename'];**

**}**

**echo '<table align="center" border="1"><tr><td style="text-align:center;"><b>Movie Name</b></td><td><b>Rating</b></td></tr>';**

**for($x = 0; $x < $arrlength; $x++) {**

**echo '<tr>';**

**echo '<td>' . $movie\_array[$x] . '</td>';**

**echo '<td>' . $rating\_array[$x] . '</td>';**

**echo '</tr>';**

**//echo $movie\_array[$x] . ' ' . $rating\_array[$x]. '<br>' ;**

**}**

**echo '</table>';**

**$conn->close();**

**?>**

**</body>**

**</html>**

**4.8 rate\_movies.php**<?php

**session\_start**();

**if**(**empty**($\_SESSION['userid'])){

**header**('location:login.php');

}

?>

<html>

<head>

<title>

**Rate Movies!**

</title>

<link rel=**"stylesheet"** type=**"text/css"** href=**"ratemovies\_style.css"**>

</head>

<body>

<header id=**"userheader"**>

<span class=**"box"** style=**" width: 320px; margin-right: 650px; margin-left: 10px;"**><?php **echo** '<h1 style="font-family: Arial;"> Welcome ' .$\_SESSION['username'].'</h1>' ?></span>

<span class = **"box1"**>

<span style=**"margin-right: 25px;"**><a href=**"movies.php"** id= **"rec"** style= **"color: #fff;**

**background: #337ab7;**

**padding: 8px 17px 8px 17px;**

**border: none;**

**text-decoration: none;**

**font-size: 15pt;**

**font-family: Arial;"**> **Go Back** </a></span>

<span><a href=**"logout.php"** id= **"logout\_button"**> **Click here to logout** </a></span>

</span>

</header>

<div id=**"search\_div"**>

<form name=**"form"** action=**"ratemovies.php"** method=**"post"**>

**<?php**

**if(isset($\_POST['btn']))**

**{**

**echo '<input id = search\_box name = search\_box type=text value="'.$\_POST['search\_box'].'">';**

**echo "</input>";**

**}**

**else**

**{**

**echo "<input id = 'search\_box' name = 'search\_box' placeholder ='Type Search Query Here' type='text' autofocus/>";**

**}**

**?>**

**<!--<input id = "search\_box" name = "search\_box" placeholder ="Type Search Query Here" type="text" autofocus/>-->**

**<input id = "search-icon" value="Submit" type="image" name="btn" src="search-icon.png"/>**

**<br>**

**<input type = "radio" name = "search\_type" value = "Movie" checked >**

**By Movie Name**

**<input type = "radio" name = "search\_type" value = "Genre" >**

**By Genre**

**</form>**

**</div>**

**<?php**

**$conn = new mysqli("localhost","root","recommender","majorproject");**

**if(isset($\_POST['btn']))**

**{**

**if(strlen($\_POST['search\_box']) <=2)**

**{**

**echo"<script>alert('The length of the search query is too less');</script>";**

**}**

**else if(isset($\_POST['search\_type']))**

**{**

**if($\_POST['search\_type']=='Movie')**

**{**

**//echo $\_POST['search\_box'];**

**echo '<span style = " margin-left: 40%; margin-right: 25%;"> <p style= "display : inline-block;"><b>Legend:</b>&nbsp;&nbsp;&nbsp;&nbsp;<p style = "color : red; display : inline-block;"><u>Unrated</u>&nbsp;&nbsp;&nbsp;&nbsp;</p> <p style = "color : green; display : inline-block;"><u>Rated</u></p> </p> </span>';**

**echo '<table align="center" border="1"><tr><td style="text-align:center;"><b>Movie Name</b></td><td><b>Average Rating</b></td>**

**<td><b>Number of Ratings</b></td></tr>';**

**$search\_term = $\_POST["search\_box"];**

**//upar waley mein mysqli\_real\_escape\_string nahi lena hai Mummy's ko search nahi karega**

**$sql = "SELECT movieid, moviename, no\_of\_users\_rated, avg\_rating FROM moviedata ORDER BY avg\_rating DESC, no\_of\_users\_rated DESC";**

**$result = $conn->query($sql);**

**$row = mysqli\_fetch\_array($result);**

**$flag = 0;**

**$x = $\_SESSION["userid"];**

**if (strpos(strtolower($row["moviename"]), strtolower($search\_term)) !== false) {**

**echo '<tr>';**

**$y = $row["movieid"];**

**$sql2 = "SELECT \* FROM ratings WHERE userid = '$x' AND movieid = '$y' ";**

**$result2 = $conn->query($sql2);**

**if (mysqli\_num\_rows($result2)==0)**

**echo "<td><a target='\_blank'style='color: red;' href='enter\_rating.php?id=";**

**else**

**echo "<td><a target='\_blank'style='color: green;' href='enter\_rating.php?id=";**

**echo $row["movieid"];**

**echo "'>";**

**echo $row["moviename"];**

**echo "</a></td>";**

**echo '<td>' . $row["avg\_rating"] . '</td>';**

**echo '<td>' . $row['no\_of\_users\_rated'] . '</td>';**

**echo '</tr>';**

**$flag = 1;**

**}**

**while($row = $result->fetch\_assoc()) {**

**if (strpos(strtolower($row["moviename"]), strtolower($search\_term)) !== false) {**

**echo '<tr>';**

**$y = $row["movieid"];**

**$sql3 = "SELECT \* FROM ratings WHERE userid = '$x' AND movieid = '$y' ";**

**$result3 = $conn->query($sql3);**

**if (mysqli\_num\_rows($result3)==0)**

**echo "<td><a target='\_blank'style='color: red;' href='enter\_rating.php?id=";**

**else**

**echo "<td><a target='\_blank'style='color: green;' href='enter\_rating.php?id=";**

**echo $row["movieid"];**

**echo "'>";**

**echo $row["moviename"];**

**echo "</a></td>";**

**echo '<td>' . $row["avg\_rating"] . '</td>';**

**echo '<td>' . $row['no\_of\_users\_rated'] . '</td>';**

**echo '</tr>';**

**$flag = 1;**

**}**

**}**

**echo '</table>';**

**if($flag==0){**

**echo"<script>alert('No movies found!');</script>";**

**}**

**}**

**else**

**{**

**echo '<table align="center" border="1"><tr><td style="text-align:center;"><b>Movie Name</b></td><td><b>Average Rating</b></td>**

**<td><b>Number of Ratings</b></td></tr>';**

**$search\_term = mysqli\_real\_escape\_string($conn, $\_POST['search\_box']);**

**$sql = "SELECT movieid, moviename, genre, no\_of\_users\_rated, avg\_rating FROM moviedata ORDER BY avg\_rating DESC, no\_of\_users\_rated DESC";**

**$result = $conn->query($sql);**

**$row = mysqli\_fetch\_array($result);**

**$flag = 0;**

**$x = $\_SESSION["userid"];**

**if (strpos(strtolower($row["genre"]), strtolower($search\_term)) !== false) {**

**echo '<tr>';**

**$y = $row["movieid"];**

**$sql2 = "SELECT \* FROM ratings WHERE userid = '$x' AND movieid = '$y' ";**

**$result2 = $conn->query($sql2);**

**if (mysqli\_num\_rows($result2)==0)**

**echo "<td><a target='\_blank'style='color: red;' href='enter\_rating.php?id=";**

**else**

**echo "<td><a target='\_blank'style='color: green;' href='enter\_rating.php?id=";**

**echo $row["movieid"];**

**echo "'>";**

**echo $row["moviename"];**

**echo "</a></td>";**

**echo '<td>' . $row["avg\_rating"] . '</td>';**

**echo '<td>' . $row['no\_of\_users\_rated'] . '</td>';**

**echo '</tr>';**

**$flag = 1;**

**}**

**while($row = $result->fetch\_assoc()) {**

**if (strpos(strtolower($row["genre"]), strtolower($search\_term)) !== false) {**

**echo '<tr>';**

**$y = $row["movieid"];**

**$sql2 = "SELECT \* FROM ratings WHERE userid = '$x' AND movieid = '$y' ";**

**$result2 = $conn->query($sql2);**

**if (mysqli\_num\_rows($result2)==0)**

**echo "<td><a target='\_blank'style='color: red;' href='enter\_rating.php?id=";**

**else**

**echo "<td><a target='\_blank'style='color: green;' href='enter\_rating.php?id=";**

**echo $row["movieid"];**

**echo "'>";**

**echo $row["moviename"];**

**echo "</a></td>";**

**echo '<td>' . $row["avg\_rating"] . '</td>';**

**echo '<td>' . $row['no\_of\_users\_rated'] . '</td>';**

**echo '</tr>';**

**$flag = 1;**

**}**

**}**

**if($flag==0){**

**echo"<script>alert('No movies found!');</script>";**

**}**

**echo '</table>';**

**}**

**}**

**}**

**$conn->close();**

**?>**

**</body>**

**</html>**

**4.9 recommendation.php**<?php

**session\_start**();

**if**(**empty**($\_SESSION['userid'])){

**header**('location:login.php');

}

?>

<html>

<head>

<title>

**Rate Movies!**

</title>

<link rel=**"stylesheet"** type=**"text/css"** href=**"recommendation\_style.css"**>

</head>

<body>

<header id=**"userheader"**>

<span class=**"box"** style=**" width: 320px; margin-right: 650px; margin-left: 10px;"**><?php **echo** '<h1 style="font-family: Arial;"> Welcome ' .$\_SESSION['username'].'</h1>' ?></span>

<span class = **"box1"**>

<span style=**"margin-right: 25px;"**><a href=**"movies.php"** id= **"rec"** style= **"color: #fff;**

**background: #337ab7;**

**padding: 8px 17px 8px 17px;**

**border: none;**

**text-decoration: none;**

**font-size: 15pt;**

**font-family: Arial;"**> **Go Back** </a></span>

<span><a href=**"logout.php"** id= **"logout\_button"**> **Click here to logout** </a></span>

</span>

</header>

<h2 style=**"text-align : center; font-family: Arial; margin-top: 50px;"**>**Click on any one of the recommendation methods**</h2>

<div style=**"margin-top: 80px;"**>

**<span class="box2">**

**<span style="margin-right: 100px; margin-left: 100px;"><a href="display\_recom.php?id=1" id= "method1" style= "color: #fff;**

**background: #337ab7;**

**padding: 8px 17px 8px 17px;**

**border: none;**

**text-decoration: none;**

**font-size: 15pt;**

**font-family: Arial;"> Method 1 </a></span>**

**<span style="margin-right: 100px; margin-left: 100px;"><a href="display\_recom.php?id=2" id= "method2" style= "color: #fff;**

**background: #337ab7;**

**padding: 8px 17px 8px 17px;**

**border: none;**

**text-decoration: none;**

**font-size: 15pt;**

**font-family: Arial;"> Method 2 </a></span>**

**<span style="margin-right: 100px; margin-left: 100px;"><a href="display\_recom.php?id=3" id= "method3" style= "color: #fff;**

**background: #337ab7;**

**padding: 8px 17px 8px 17px;**

**border: none;**

**text-decoration: none;**

**font-size: 15pt;**

**font-family: Arial;"> Method 3 </a></span>**

**<span style="margin-right: 100px; margin-left: 100px;"><a href="display\_recom.php?id=4" id= "method4" style= "color: #fff;**

**background: #337ab7;**

**padding: 8px 17px 8px 17px;**

**border: none;**

**text-decoration: none;**

**font-size: 15pt;**

**font-family: Arial;"> Method 4 </a></span>**

**</span>**

**</div>**

**</body>**

**</html>**

**4.10 register.php**

<?php

**session\_start**();

//connect to database

$conn = **new** mysqli("localhost","root","recommender","majorproject");

**if**(**isset**($\_POST['btn']))

{

$username=**mysqli\_real\_escape\_string**($conn, $\_POST['user']);

$password=**mysqli\_real\_escape\_string**($conn, $\_POST['pass']);

$password2=**mysqli\_real\_escape\_string**($conn, $\_POST['repass']);

**if**($password!=$password2){

$\_SESSION['message']="The two passwords do not match";

}

**else** **if**(**strlen**($password) < 8){

$\_SESSION['message']="The password length is less than 8";

}

**else** **if**(**strlen**($username) < 2){

$\_SESSION['message']="The Username is too small";

}

**else**

{

//Create User

$password = **md5**($password);

$sql="INSERT INTO users (userid,username,password) VALUES (NULL,'**$username**','**$password**')";

**if** ($conn->query($sql) === **TRUE**) {

$\_SESSION['message']="You are now logged in";

$\_SESSION['username']=$username;

**header**("location:login.php");

}

}

$conn->close();

**}**

**?>**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>**

**Register page**

**</title>**

**<link rel="stylesheet" type="text/css" href="register\_style.css">**

**</head>**

**<body>**

**<?php**

**if(isset($\_SESSION['message']))**

**{**

**echo "<div id='error\_msg'>".$\_SESSION['message']."</div>";**

**unset($\_SESSION['message']);**

**}**

**?>**

**<div id="frm">**

**<form action="register.php" method="POST">**

**<p>**

**<input placeholder = 'Username' type='text' id='user' name="user" />**

**</p>**

**<p>**

**<input placeholder = "Password" type="password" id="pass" name="pass" />**

**</p>**

**<p>**

**<input placeholder = "Re-type Password" type="password" id="repass" name="repass" />**

**</p>**

**<br>**

**<div>**

**<input type="submit" id="btn" name= 'btn' value="Sign Up" />**

**</div>**

**</form>**

**</div>**

**</body>**

**</html>**