The Faculty of Computers and Information

Competitive programming Recommender Engine/Platform for Improving Problem Solving Competencies

A graduation project dissertation by:

Adel Omer Ewis

Ahmed Nasser Hussin

Marwan Mohamed Kamel

Mohamed Ali

Mostafa Mohamed Thabet

Submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Computers & Information, at the Computer Science Dept., the Faculty of Computers & Information, Helwan University

Supervised by:

Dr. Amr S.Ghoneim

July 2016

“For the things we have to learn before we can do them, we learn by doing them.”

— **Aristotle, the Nicomachean Ethics**

Abstract

Complex situations and problems leads to think different than usual and use the giving information to solve these problems. These skills human gains with time passes and practice how to solve problems.

Practicing by solving problems in math, chemistry or mostly in any field of science has been one of best ways to learn, doing is mostly better than just reading new thing to do, and the same thing goes for computer programming.

Practical exercises is an integral part of learning programming , which is a core basic skill required in computer science and it is best learned by doing and luckily there is many programming problems that are available on the web that require problem solver to write an algorithm to solve a particular problem under specific time and space constrains but with this diversity beginner faces a problem in not knowing where to start they and on often situation they face a hard problem for them and get frustrated and decide not to practice at all , the same thing goes for experienced programmers who are looking for a challenge and end up facing easy problem for them . They need recommendation for problems based on their skill.

Recommender systems apply knowledge discovery techniques for making personalized recommendation for information and it has wide spread on the web from recommending products in E-commerce websites like Amazon recommending music and videos in Spotify and YouTube, and in search engines to rank search results by using the links on millions of web sites to decide which pages were most relevant and recommendation systems has achieved a lot of success in those areas.

In this document we present a recommendation system for recommending problems to student to solve based on their skills and their solved problems history using collaborative filtering techniques.

Keywords

Problem Solving, Recommendation System, Programming, collaborative filtering, Competitive programming, ACM programming contest

Index

A

C

L

R

LIST OF TABLES

LIST OF FIGURES

Chapter 1: Introduction

1.1 Overview

The Computer science students from all over the world in their four years of study they do they best to learn programming to create something useful either for programmer or the ordinary people in final product like desktop application or on the web, or solve some problems using programming languages.

Every software has an idea and problems in the implementation process to solve these problems it will take much time if the problem is new to the programmer or problem is much complex. So, there is an organization called Association for Computing Machinery (ACM) release yearly competition for the college students of computer science or related fields to compete to each other with number of problems with some constraints like time and amount of resources allowed to use.

This competition encourages students to learn programming and problems solving skills able to solve any problem faster in short amount of time.

In the beginning of this competition the students enter the competition with self-learning and depend on their own but by the time the colleges hired people to train the competitors who will enter the competition to teach them some techniques and ways to solve problems that has special way to treat.

This competition is holding locally, regionally and internationally (ICPC), in the local competition the first three places will lead to the regionally competition and so on in the regionally competitions to reach to the international one.

There are many students who wants to enter the competition and win but they don't have any help and coaches to train them and give them some knowledge and experience to solve this type of problems. some people who interested in this competition and want to help these students that they don't have any coach to train they released websites that have a lot of problems to solve with online judge to see if the problem solved correct or wrong answer.

This is good idea but by the time the problem appears that if new competitor he stuck and doesn't know how to begin, the problems don't categorize by level or by the problem type like (String, math, dynamic programming, geometry, etc.).

The most website used by the competitors is (Codeforces) this website has the problem set and the problems are leveled with problem tags but it sorts the problems according to how many persons solved this problem.

So, the new competitor using the website without any coach to train might open any problem without guidance and try to solve the problem and then get frustrated that he can't to solve this problem time by time he will leave this thing.

So, why computer science or other related fields students train their self in problems solving skills. The reasons are many some of them the problems solving skills improve data structure techniques and algorithms how to use them to solve any problem and improve the performance of the software.

As a result, the international software companies hire those people who has problem solving skills by creating a contest on any online judge – mostly CodeForces and HackerRank – to test the competitors' skills and choose the best competitors to train or work with them. Orange company starts to do this competition to hire best of the best and continue his work with best software engineers.

The idea is if we have recommendation system recommends the problems based on some features and attributes that recommend to every person the problems that can fit and able to solve or slightly harder to enhance his level and gain experience in problem solving and with practice over and over on solving he will be able to enter the ACM competitions.

1.2 Research Motivation

Companies starts to interest in recommendation system for 15 years ago because the ability to recommends things like people's taste based on some features to know the user's taste. And automated collaborative filtering systems have proven to be accurate enough for entertainment domains [27,29,30,31]

Recommendation system tested and working till now in movies, music application and commercial websites. In the movies recommendation engine recommends to every user in the system the movies that they may like based on the personal profile history or based on the similarity between the whole users on the system. Example on that is Netflix now the bestselling in the website came from the recommendation system and the accuracy of recommendation system reached to 71%, also Amazon the most selling items came from the recommendation system and increased the revenue to the company.

So, the recommendation engines played now main role in the business companies as it is increase the revenue of the company double or triple unlike the system without recommendation engine.

Recommendation systems have shown to be successful in many domains where information overload exists. This success has motivated research on how to deploy recommender systems in educational scenarios to facilitate access to the wide spectrum of information. Tackling open issues in their

deployment is gaining importance as lifelong learning becomes a necessity of the current knowledge-based society.

We want to use educational recommendation system to enhance contestant problem solving skills by recommends to him problems he can solve and by practice the problem solving skill will level up to give him a chance to register in the problem solving competition ACM ICPC and ranking among his competitors.

The recommendation system has two techniques and every one of them used depend on the situation and the data available:

- **Collaborative filtering**: methods are based on collecting and analyzing a large amount of information on users' behaviors, activities or preferences and predicting what users will like based on their similarity to other users. A key advantage of the collaborative filtering approach is that it does not rely on machine analyzable content and therefore it is capable of accurately recommending complex items such as movies without requiring an "understanding" of the item itself.
- **Content-based filtering**: methods are based on a description of the item and a profile of the user's preference. In a content-based recommender system, keywords are used to describe the items and a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past (or is examining in the present). In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended.

The problem solver knowledge about the topic that the problem addresses and how to use that knowledge to solve it (like a problem require knowledge about binary search tree prior solving) and this not the topic we try to solve as most of the problems has tutorial with it and anyone can read and search about the topic included in this problem or read others solutions.

1.3 Research Question

Recommendation systems have shown to be successful in many domains but few of this are made in the field of education and rarely of this is tried in creating some sort of a guidance for student in problem solving , in this field theirs only few web sites like (a2oj.com) that offer some sort of guidance for problem solver by giving them only a ladder of static pre defied set of problems but the problem with this is it is fixed and need to be changed manually by someone whit expertise in that field and these problems are nowhere near to a personalized list it is just sorted by its level but students have different mindset they all does not solve like one another they need some kind of personalized list of problems for each of them.

So that takes us to our main questions:

How well collaborative filtering approach will can achieve in producing some personalized problems for users, and how effective is this in enhancing their performance?

We believe that by solving more problems students will get better in time same as practice in mathematics.

How to prevent the problem solver from sticking to long in specific level without getting any better?

This require two things:

- The problem solver knowledge about the topic that the problem addresses and how to use that knowledge to solve it (like a problem require knowledge about binary search tree prior solving) and this not the topic we try to solve as most of the problems has tutorial with it and anyone can read and search about the topic included in this problem or read others solutions.
- he problem The problem solver knowledge about the topic that the problem addresses and how to use that knowledge to solve it (like a problem require knowledge about binary search tree prior solving) and this not the topic we try to solve as most of the problems has tutorial with it and anyone can read and search about the topic included in this problem or read others solutions. Solver knowledge about the topic that the problem addresses and how to use that knowledge to solve it (like a problem require knowledge about binary search tree prior solving) and this not the topic we try to solve as most of the problems has tutorial with it and anyone can read and search about the topic included in this problem or read others solutions.
- Having some kind of ladder that takes him from level of problems that he can solve and has been used to it to a higher and harder level of problems and so on.

1.4 Original Contributions

The main contribution of this thesis to test the effectiveness of collaborative filtering techniques in enhancing student performance in problem solving by recommending problems suitable for them and if it does how well does it enhance their performance and how well the quality of each problem solved by each user affect their performance.

This is the first research that addresses using recommendation system in that area (competitive programming and problem solving).

Chapter 2: A Literature Review

2.1 Recommendation System – Overview

Although the roots of recommender systems can be traced back to the extensive work in the cognitive science, approximation theory, information retrieval, forecasting theories, and also have links to management science, recommender systems emerged as an independent research area in the mid- 1990's when researchers started focusing on recommendation problems that explicitly rely on the ratings structure. In its most common formulation, the recommendation problem is reduced to the problem of estimating ratings for the items that have not been seen by a user. Intuitively, this estimation is usually based on the ratings given by this user to other items and on some other information that will be formally described below. Once we can estimate ratings for the yet unrated items, we can recommend to the user the item(s) with the highest estimated rating(s).

RS make use of different sources of information for providing users with predictions and recommendations of items. They try to balance factors like accuracy, novelty, disparity and stability in the recommendations. Collaborative Filtering (CF) methods play an important role in the recommendation, although they are often used along with other filtering techniques like content-based, knowledge-based or social ones.

CF is based on the way in which humans have made decisions throughout history: besides on our own experiences, we also base our decisions on the experiences and knowledge that reach each of us from a relatively large group of acquaintances.

In recommender systems the utility of an item is usually represented by a rating, which indicates how a particular user liked a particular item, e.g., John Doe gave the movie "Harry Potter" the rating of 7 (out of 10).

However, as indicated earlier, in general utility can be an arbitrary function, including a profit function. Depending on the application, utility u can either be specified by the user, as is often done for the user-defined ratings, or is computed by the application, as can be the case for a profit-based utility function.

The evolution of RS has shown the importance of hybrid techniques of RS, which merge different techniques in order to get the advantages of each of them.

Recommender systems are usually classified into the following categories, based on how recommendations are made:

Content-based recommendations: the user is recommended items similar to the ones the user preferred in the past;

Collaborative recommendations: the user is recommended items that people with similar tastes and preferences liked in the past;

Hybrid approaches: these methods combine collaborative and content-based methods.

2.2 Hybrid systems

Recommendation has many significant advantages over traditional content-based filtering, primarily because it does not depend on error-prone machine analysis of content. The advantages include the ability to filter any type of content, e.g. text, art work, music, mutual funds; the ability to filter based on complex and hard to represent concepts, such as taste and quality; and the ability to make serendipitous recommendations. It is important to note that recommendation technologies do not necessarily compete with content-based filtering. In most cases, they can be integrated to provide a powerful hybrid filtering solution. recommendation systems have been successful in research, with projects such as GroupLens [29,30], Ringo [31], Video Recommender [27], and MovieLens [32] gaining large

followings on the Internet. Commercially, some of the highest profile web sites like Amazon.com, CDNow.com, MovieFinder.com, and Launch.com have made successful use of recommendation technology.

2.3 SOURCES OF ERROR

2.3.1 Model/Process Errors

Model or process errors occur when the recommendation system uses a process to compute recommendations that does not match the user's requirements. For example, suppose a person A solved a problem of level A div2 (the easiest level) and problem E div1(the hardest level) and a person B solved the first problem person solved this does not necessarily mean that person B is capable of solving problem of level E. The recommendation system he is using however does not have a computational model that is capable of recognizing the two distinct interests represented in his rating profile. As a result, the recommendation system may match person A with person B, resulting in a continuous stream of recommendations for problems that could only be solved by someone who solved the harder problem.

2.3.2 Data Errors

Data errors result from inadequacies of the data used in the computation of recommendations. Data errors usually fall into three classes: not enough data, poor or bad data, and high variance data. Missing and sparse data are two inherent factors of recommendation computation. If the data were complete, there would be no need for recommendation systems to predict the missing data points. Items and users that have newly entered the recommendation system are particularly prone to error.

As for a movies recommendation example When a new movie is first released, very few people have rated the movie, so the recommendation must base predictions for that movie on a small number of ratings. Because there are only a small number of people who have rated the movie, the recommendation system may have to base recommendations on ratings

from people who do not share the user's interests very closely. Likewise, when new users begin using the system, they are unwilling to spend excessive amounts of time entering ratings before seeing some results, forcing the recommendation system to produce recommendations based on a small and incomplete profile of the user's interests. The result is that new users may be matched with other people who share their interests on a small subset of items, but in actuality don't share much more in common.

Even in cases where considerable amounts of data are available about the users and the items, some of the data may contain errors.

For example, suppose Ahmed found a problem of harder level than he usually solves and got it right and the recommendation system takes this problem into account for Ahmed because he solved harder level than he usually solved once does not mean that he reached that level, he may not be aware of the offending rating.

High variance data is not necessarily considered bad data, but can be the cause of recommendation errors. For example, of all the people selected who have rating profiles similar to Ahmad's, half rated the problem number 365 high and half rated it low. As a movie that polarizes interests, the proper prediction is probably not the average rating (indicating ambivalence), although this is probably what will be predicted by the recommendation system.

2.4 Related Work

[1] The main aim of this work is provide students with recommendations, in the complex process of deciding how many and which courses enroll on, taking into consideration academic performance and other similar characteristics.

The research has focused in testing, using real data, the application of techniques and tools in data mining to support students when enrolling on

a new term. To achieve this target, they had analyzed real data from other students who had taken the same courses in the past, using techniques such as decision trees to discover trends, patterns and rules that will be used as support for decision making in the itinerary of a certain university career.

It was found that using data mining techniques, enabled them to develop a model representing the behavior of students in their way through different academic courses. This facilitates a proper vision of the behavior and performance of the group of students at certain university career and, at the same time, allows feeding the system to offer recommendations for students to increase their effectiveness and relevance at decision taking in relation with the courses to be enrolling on.

The paper presented four experiments by using the same technique with four different configurations, in order to show the advantages and disadvantages of the technique used as well as to detect classifiers that would perform better in real settings. By applying the learning algorithm to all the instances, the accuracy was 77.3% Even though the obtained results had been good and satisfying, but we see that it is important to test the system with data from other universities or other schools in order to know consistency and convergence.

[2] Different collaborative filtering techniques have been proposed to overcome the problem of the processing time and the data latency. The obtained results from different recommendation systems noticed that collaborative filtering techniques provide the systems enough ability to support users with recommendations. Consequently, the recommendation systems are able to predict user behavior patterns without any knowledge of the user in advance, and to test the accuracy by the comparing the obtained predictions and the reality.

If clustering is performed by Genetic algorithm, Nearest-Neighbor algorithm, or the algorithm developed based on any of these two, the gaps

among data affect the accuracy of the prediction a lot. This also means that missing data would lower the accuracy of the prediction.

The situation is the same for EM approaches. This is because Expectation Maximization (EM) approaches perform better when the probability space is more complete. The accuracy of the prediction performed by hierarchical approaches may also be affected since the recommended items would be too general due to the lack of detailed categorizations.

These statements indicate that the reduction of the missing or insufficient data is not simple, and that some approximations are required to be performed in order to provide better predictions of user preferences. An approach to decrease online computational time is to allow recommendation systems perform the clustering offline (Chee, S. H. S., Han, J., & Wang, K. (2001). RecTree: An Efficient Collaborative Filtering Method. Lecture Notes in Computer Science).
Several algorithms and techniques proposed perform computations in a constant time (Lemire D. (2003). Scale and Translation Invariant Collaborative Filtering Systems. Journal of Information Retrieval). These algorithms and techniques provide the possibility of real time updates on profiles in the recommendation systems.

[3] Recommendation algorithms provide an effective form of targeted marketing by creating a personalized shopping experience for each customer. For large retailers like Amazon.com, a good recommendation algorithm is scalable over very large customer bases and product catalogs, requires only subsecond processing time to generate online recommendations, is able to react immediately to changes in a user's data, and makes compelling recommendations for all users regardless of the number of purchases and ratings. Unlike other algorithms, item-to-item collaborative filtering is able to meet this challenge.

• Traditional collaborative filtering does little or no offline computation, and its online computation scales with the number of customers and catalog items. The algorithm is impractical on large data sets, unless it uses dimensionality reduction, sampling, or partitioning — all of which reduce recommendation quality.

• Cluster models can perform much of the computation offline, but recommendation quality is relatively poor. To improve it, it's possible to increase the number of segments, but this makes the online user–segment classification expensive.

• Search-based (content-based) models build keyword, category, and author indexes offline, but fail to provide recommendations with interesting, targeted titles. They also scale poorly for customers with numerous purchases and ratings.

Rather than matching the user to similar customers, item-to-item collaborative filtering matches each of the user's purchased and rated items to similar items, then combines those similar items into a recommendation list.

[4] This research presents a new way to measure similarity between users, which is usable in collaborative filtering processes carried out in recommender systems. The proposed metric is formulated via a simple linear combination of values and weights. Values are calculated for each pair of users between which the similarity is obtained, whilst weights are only calculated once, making use of a prior stage in which a genetic algorithm extracts weightings from the recommender system which depend on the specific nature of the data from each recommender system. The results obtained present significant improvements in prediction quality, recommendation quality and performance.

This paper presented a genetic algorithm method for obtaining optimal similarity functions. The similarity functions obtained provide better quality and quicker results than the ones provided by the traditional metrics. Improvements can be seen in the system's accuracy, in the coverage and in the precision & recall recommendation quality measures. The proposed use of GAs applied to the RS is a novel approach and has the main advantage that it can be used in all CF-based RS, without the need to use hybrid models which often cannot be applied, as in many cases no reliable demographic information or content-based filtering information is available. The GA-metric runs 42% as fast as the correlation, which is a great advantage in RS, which are highly dependent on equipment overloads when many different recommendation requests are made simultaneously (user to user) or on costly off-line running processes (item to item).

[5] The recommender systems of e-learning allow the possibility of weighting the importance of the recommendations that each user generates, depending on their level of knowledge. In order to include the knowledge level of the users in the collaborative filtering step, it is necessary to design new metrics which, being based on the current ones, incorporate the additional information with regards to the scores obtained by each user.

The validation of the new metrics requires a modification of the traditional equations used in order to measure the total error of the system when these metrics are used; the mean absolute error, mean squared error, or whatever other measurement of the total accuracy of the system must also reflect the knowledge level of the users. The new metric proposed in the paper has obtained better results than the traditional equivalent when both have been subjected to the processing of the total accuracy of the system using a MAE measure adapted to include the knowledge of the users. The remaining indicators studied (percentage of correct predictions and percentage of particularly erroneous predictions) has also produced better results using the proposed metric. Although the experiments have

not been carried out with an e-learning database, both the equations designed and the methodology used could be used in the same way in different recommender systems of e-learning. The solidity of the database used, the large number of experiments carried out and the quality of the results obtained permit us to face developments in the distinct recommender systems in the sense of differentiating the users by some characteristic, such as by knowledge in the sphere of collaborative e-learning. A well-defined field of research exists in collaborative filtering when the nature of the recommender systems allows the incorporation of weighting in the importance of each one of the users, and the collaborative systems of e-learning are named to lead the developments in this new field of investigation.

[6] Collaborative filtering recommender systems often use nearest neighbor methods to identify candidate items. In this paper they present an inverted neighborhood model, k-Furthest Neighbors, to identify less ordinary neighborhoods for the purpose of creating more diverse recommendations.

The approach is evaluated two-fold, once in a traditional information retrieval evaluation setting where the model is trained and validated on a split train/test set, and once through an online user study (N=132) to identify users' perceived quality of the recommender. A standard k-nearest neighbor recommender is used as a baseline in both evaluation settings. This evaluation shows that even though the proposed furthest neighbor model is outperformed in the traditional evaluation setting, the perceived usefulness of the algorithm shows no significant difference in the results of the user study.

This work presented an extensive analysis of the recommendation quality of both a nearest neighbor and a furthest neighbor algorithm. They evaluated the quality of these algorithms through traditional information retrieval ways as well as through a user study in order to gain insight on the

perceived usefulness of the algorithms. In terms of precision and recall, the nearest neighbor algorithm outperforms the furthest neighbor approach. The lists of recommended items are however almost completely disjoint due to the nature of the furthest neighbor approach, which recommends more nonobvious and diverse items. By analyzing the feedback obtained from a user study with more than 130 participants have shown that a higher predictive accuracy of a recommender system, in this scenario, does not increase the perceived usefulness of the recommender system, from the users' perspective.

[7] In this paper, we chose to focus on the most widely used dataset and relevance measure in order to test the main methods for collaborative filtering and their main options. According to first results on default approaches, it seems that using Bayes model for default predictions is relevant since it has reasonable error rate for very low execution time.

Besides, another important advantage of such a technique is that it is easily updatable since it is incremental, whereas the other approaches need to relearn their entire model in order to take into account new data. For all experiments, rounding the predicted ratings led to an improvement ranging from 2 to 6 percent of the MAE. Besides, rounding ratings is natural in practice, since real users generally prefer rating scales based on natural numbers than on real numbers. Computing predictions using weighted sum of deviations from the mean also led to better results than using simple weighted sum for both user- and item-based approaches. As far as we know, using such prediction scheme for item-based approaches is new, and that is what led us to the best results.

The lowest error rates were reached using Pearson similarity for user-based approaches and probabilistic similarity for item-based approaches. Using Bayes default approach in order to predict ratings inside a given cluster leads to better results than when the mean item rating of the cluster

members is used. Considering the prediction of the nearest cluster is better than computing a weighted sum of the predictions of each cluster.

Finally, K-means led to better results than more sophisticated algorithms like LAC or SSC. We think it is not relevant, in the field of collaborative filtering, to assume that the ratings of users of a same cluster follow a normal distribution. In particular, we are faced with the problem that users generally use differently the rating scale: for example, one user may rate 5 a movie he likes and 3 a movie he dislikes whereas another user with the same tastes will rate 4 a movie he likes and 1 a movie he dislikes. More generally, item-based approaches seem the bests in our experiments. But these results need to be taken with precaution.

[8,9] In general, data mining techniques are defined as extracting or mining knowledge from data. These techniques are used for the exploration and analysis of large quantities of data in order to discover meaningful patterns and rules.

They can be used to lead decision making and to predict the effect of decisions. Significantly, many researchers have used data mining techniques to improve the performance of recommender systems. Consequently, it is meaningful to classify the research papers according to data mining techniques.
data mining techniques could be classified into the following eight categories: association rule, clustering, decision tree, k-nearest neighbor, link analysis, neural network, regression, and other heuristic methods.

(1) Association rule: Association rule mining refers to the discovery of all association rules that are above user-specified minimum support and minimum confidence levels. Given a set of transactions in which each transaction contains a set of items, an association rule applies the form X) Y, where X and Y are two sets of items [10].

(2) Clustering: The clustering method identifies a finite set of categories or clusters to describe data. Among the clustering methods, the most popular are K-means and self-organizing map (SOM). K-means takes the input parameter, K, and partitions a set of n objects into K clusters [11]. SOM is a method for an unsupervised learning, based on an artificial neurons clustering technique [12].

(3) Decision tree: Most popular classification method is decision tree induction. The top node in a tree is called as a root node. A decision tree is a tree that each internal node (non-leaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each terminal node (leaf node) denotes a class prediction.

(4) k-Nearest neighbor: The k-NN (k-nearest neighbor) model, a typical traditional CF-based recommender system, makes recommendations according to the following three phases.

(a) Recommender systems construct a user profile using the user's preference ratings, which are obtained either directly from explicit ratings of items or indirectly from purchase or usage information.

(b) Recommender systems apply statistical or machine learning techniques to discover k users, known as neighbors or recommenders, who in the past have shown similar behaviors. A neighborhood is formed based on the degree of similarity between a mark user and other users.

(c) Once a neighborhood is formed for a target user, recommender systems make a top-n item set that the target user is most likely to purchase by analyzing the items in which neighbors have exhibited interest [14].

(5) Neural network: A neural network is a parallel distributed information processing system that is able to learn and self-organize. This system consists of a large number of uncomplicated processing entities which are interconnected to form a network that conducts complex computational tasks A neural network builds a class of very pliable model that can be used for a diversity of different applications, such as prediction, non-linear regression, or classification [15].

(6) Regression: Regression analysis is a powerful process for analyzing associative relationships between dependent variables and one or more independent variables. It has been used for curve fitting, prediction, and testing systematic hypotheses about relationships between variables.

This work [17] explained the two main types of recommendation systems and simplified the mechanism of their work first one is the content based CBF recommendation systems typically
(1) build an item profile by extracting a set of features from each item in the item set,
(2) build a content-based user profile from a set of features of the items which each user purchased.
(3) calculate the similarity between the user profiles and the item profiles using similarity function and.
(4) recommend top n items with high similarity scores. Specifically, they recommend items based on the similarity between items.

In the early stage of CBF recommendation systems, they were used to recommend documents such as net news, web pages and books. Both user profiles and item profiles are represented by vectors of associated weights given to a set of keywords.

A similarity score is computed based on both profiles using a heuristic function, such as the Pearson correlation coefficient, cosine similarity, or distance-based similarity.

[18] On the other hand, CF-based recommendation systems typically:
(1) construct a user profile from rating information of each user on items,
(2) select neighbors of a target user using similarity function based on the rating information.

(3) predict the ratings of the target user on target items as an average, weighted sum or adjusted weighted sum of ratings given to them by neighbors, and,

(4) recommend top-rated n items. These methods of rating prediction which recommend items based on the similarity between users are called memory-based CF.

Another method of rating prediction is model-based CF in which a model such as a probabilistic model or a machine learning model is built from a large collection of ratings in order to predict ratings of target items.

[20] Tapestry [21] is one of the earliest implementation of collaborative filtering-based recommender system. This system relied on the explicit opinions of people from a close-knit community, such as an office workgroup. However, recommender system for large communities cannot depend on each person knowing the other. Later, several ratings-based automated recommender systems were developed. The GroupLens research system [23, 22] provides a pseudonymous collaborative filtering solution for usenet news and movie.

Ringo [24] and video recommender [25] are email and web-based systems that generate recommendations on music and movies, respectively. A special issue of communications of the ACM [26] presents a number of different recommender systems.

Other technologies have also been applied to recommender systems, including Bayesian networks, clustering, and horting. Bayesian networks create a model based on a training set with a decision tree at each node and edges representing user information. The resulting model can be built off-line over a matter of hours or days.

The resulting model is very neighbor methods. Bayesian networks may prove practical for environments in which knowledge of user preferences changes slowly with respect to the time needed to build the model but are not suitable for environments in which user preference models must be updated rapidly or frequently.

Chapter 3: Methodologies

3.1 Data Representation for the Recommendation Engine

Data model is a core part for the recommendation engine it holds the user-problem preference matrix which the recommendation engine uses to recommend problems to users.

The submission data we get form CodeForces API contains all user submission in a JSON format for each user in a separate file, of course we need another representation of all users and their submission for the recommendation algorithm, so we tested two ways for data Representation:

1- We tried to make one big matrix that contains all the available users(including the ones who never solved a problem but registered on the site) and all the available problems that is at the time of writing is 24974 by 3514 that is a matrix of size more than 87 million element of the matrix and made a very big file of size more than 200 mb , that was very inefficient as it contains empty elements (intersection of user row and problem column) and made a big file that will only get bigger after users make more submission and thus will take longer time to read and much more unnecessarily space
2- Using Mahout **DataModel** , Apache Mahout is a project of the Apache Software Foundation to produce free implementations of distributed or otherwise scalable machine learning algorithms focused primarily in the areas of collaborative filtering, clustering and classification.
   we used DataModel of this library which resulted in much more efficient data representation in file and in memory and faster data to

memory loading (about 3X times faster than the previous matrix) and less than a half the size of the previous matrix .
All the JSON files of user's submission are parsed using JAVA Jackson library which is considered to be the fastest parsing library for JSON files.

What is DataModel?

A **DataModel** is the interface to information about user preferences.
it stores and provides access to all the preference, user, and item data needed in the
computation and it puts efficiency and scalability in mind in it.

Representing preference data in dataModel:

The input to a recommender engine is preference data—who likes what, and how
much. That means the input to Recommenders is simply a set of user ID, item
ID, and preference value tuples—a large set, of course. Sometimes, even preference
values are omitted in our case (using binary data) and preferences are stored in preference object which is the most basic abstraction, representing a single user ID, item ID, and preference value. One object represents one user's preference for one item.

So why not just use a regular java object or Collection<Preference> or Preference [] instead of a preference object? That is because of the Objet overhead Collections and arrays turn out to be quite inefficient for representing large numbers of Preference objects and objects in Java has significant overhead. A single Preference contains 20 bytes of useful data: an 8-byte user ID (Java long), an 8-byte item ID (long), and a 4-byte

preference value (float). The object's existence entails a startling amount of overhead: 28 bytes! This includes an 8-byte reference to the object, and, due to Object overhead and other alignment issues, another 20 bytes of space within the representation of the object itself. Hence a Preference object already consumes 140 percent more memory than it needs to, just due to overhead.

PreferenceArray and implementations:

As it was mentioned in the previous section creating an array of preference java objects will cause a major overhead so instead we will use PreferenceArray which is an interface
whose implementations represent a collection of preferences with an array-like
API. For example, GenericUserPreferenceArray represents all preferences associated with one user. Internally, it maintains a single user ID, an array of item IDs, and an array of preference values. The marginal memory required per preference in this representation is only 12 bytes (one 8-byte item ID and a 4-byte preference value in
an array). Compare this to the approximately 48 bytes required for a full Preference object. The four-fold memory savings alone justifies this special implementation, but it also provides a small performance win, because far fewer objects must be allocated and examined by the garbage collector. The reduced memory requirement that PreferenceArray and its implementations bring is well worth its complexity. Cutting memory requirements by 75 percent isn't just saving a couple of megabytes—it's saving tens of gigabytes of memory at reasonable scale. The below figures compare between the less efficient and the java object representation and DataModel representation

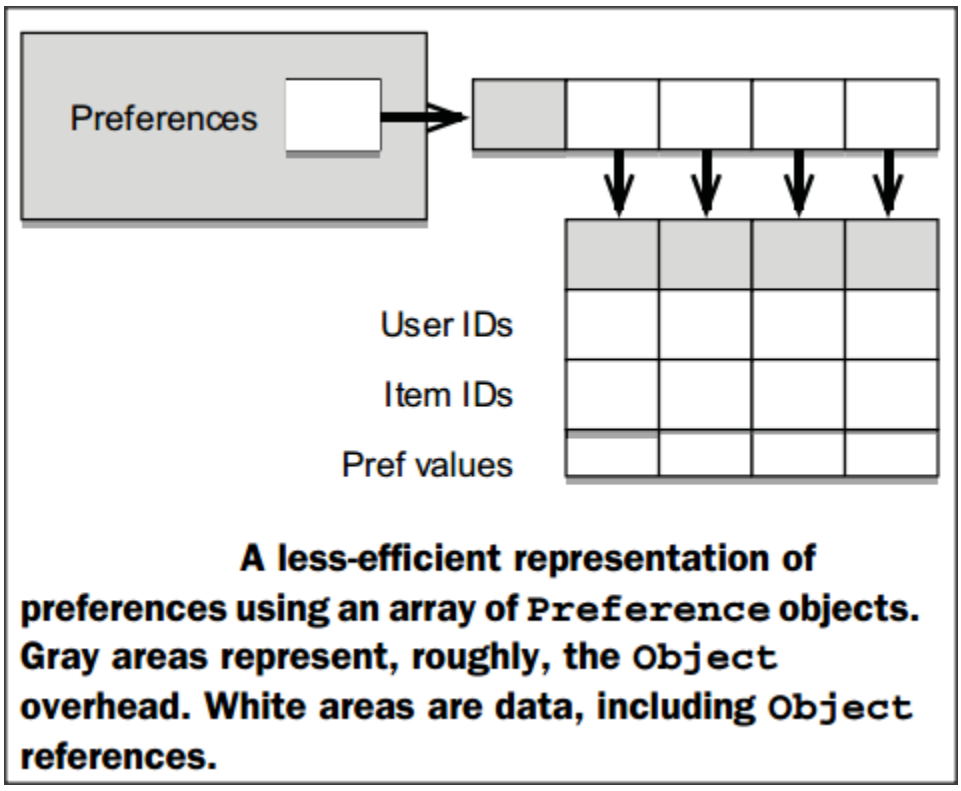

Figure 1: DataModel of Java Object

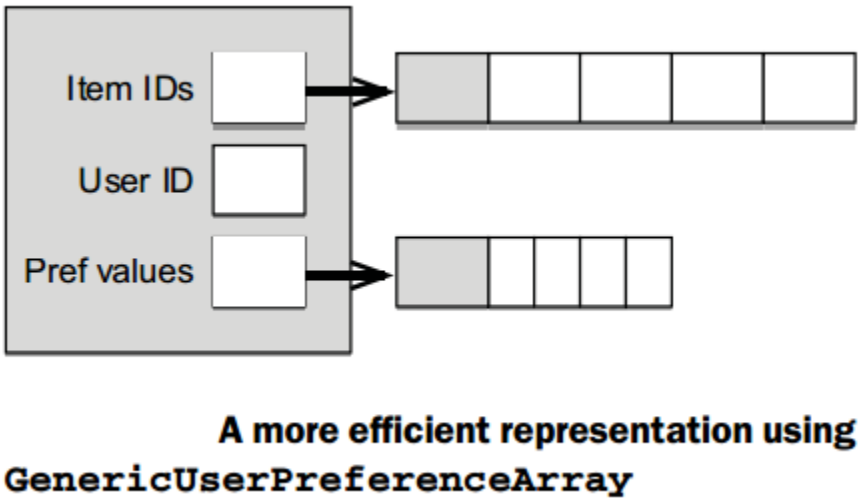

Figure 2: DataModle of Mahout object

A GenericDataModel has its own implementation for map and set collection called FastMap, FastByIDMap, and FastIDSet it behaves the same but they're specialized explicitly and only for what recommenders need. They reduce memory footprint rather than significantly increasing performance and the storage difference between them is significant: FastIDSet requires about 14 bytes per member on average, compared to 84 bytes for HashSet. FastByIDMap consumes about 28 bytes per entry, compared to again about 84 bytes per entry for HashMap. It goes to show that when one can make stronger assumptions about usage, significant improvements are possible—here, largely in memory requirements. Given the volume of data involved in recommender systems, these custom implementations more than justify themselves.

To wrap things up A GenericDataModel simply accepts preferences as inputs, in
the form of a FastByIDMap that maps user IDs to PreferenceArrays with data for
those users (like shown in the second figure)

DataModel Input

We do not use GenericDataModel direcutly instead we use **FileDataModel** which reads data from a file that we previously parsed from the user submission and stores the resulting preferencemdata in memory, in a GenericDataModel.

Another important feature of the FileDataModel is its support for data update files. These are just more data files that are read after the main data file, and that overwrite any previously read data. New preferences are added and existing ones are updated, all of this is updated without the need to read the whole data files again it will just update in-memory dataModel with the new data all of this thanks to DataModel implementation of Refreshable interface and by using the method refresh this simply requests that the component reload, recompute, and refresh its

own state based on the latest input data available, after asking its dependencies to do likewise.

3.2 Recommendation System work

The recommendation engine in Collaborative filtering based that involves rating on items are two types item-item based and user-item based. The two types follow the same steps to generate recommended items to users but with slightly different with them.

The item-item based shows the relation between the items with each other but the user-item based shows the relation between the users and items.

To get the similarity between two users we have techniques to calculate the similarity and each technique work depend on the dataset we have to work perfectly.

3.2.1 Similarity Techniques

3.2.1.1 Euclidean distance score

One very simple way to calculate a similarity score is to use a Euclidean distance
score, which takes the items that people have ranked in common and uses them as axes for a chart.

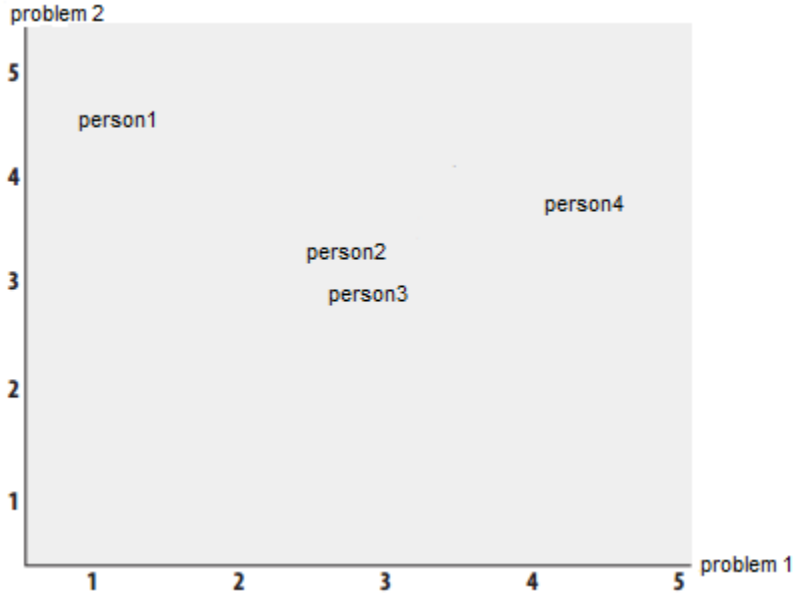

Figure 3: Euclidean distance score

To calculate the distance between person1 and person2 in the chart, take the difference in each axis, square them and add them together, then take the square root of the sum.

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$

Equation 1

(work with real number dataset)

3.2.1.2 Pearson correlation score

A slightly more sophisticated way to determine the similarity between people's interests is to use a Pearson correlation coefficient. The correlation coefficient is a measure of how well two sets of data fit on a straight line. The formula for this is more complicated than the Euclidean distance score, but it tends to give better results in situations where the data isn't well normalized.

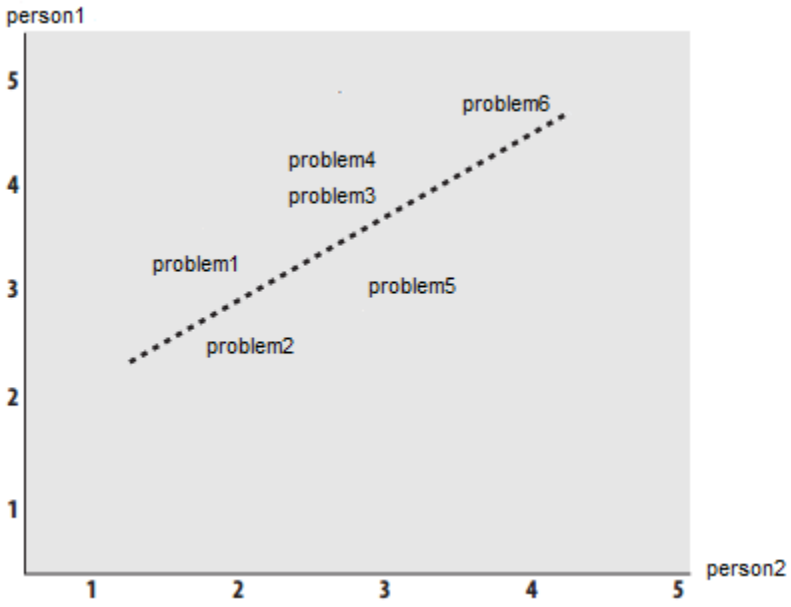

Figure 4: Pearson correlation score

the Pearson correlation score first finds the items rated by both persons. It then calculates the sums and the sum of the squares of the ratings for the two persons, and calculates the sum of the products of their ratings. Finally, it uses these results to calculate the Pearson correlation coefficient.

$$r = \frac{\sum XY - \frac{(\sum X)(\sum Y)}{n}}{\sqrt{\left(\sum X^2 - \frac{(\sum X)^2}{n}\right)\left(\sum Y^2 - \frac{(\sum Y)^2}{n}\right)}}$$

Equation 2

(work with real number dataset)

3.2.1.3 The Jaccard similarity coefficient

Jaccard coefficient is a statistic used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets.

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

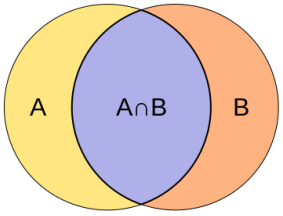

Figure 5

Equation 3

Jaccard coefficient is using to calculate similarity of binary datasets between two persons and uses the same concept of the intersection divided by the size of the union but on the binary data.

M_{11} : represents the total number of attributes where A and B both have a value of 1.

M_{01} : represents the total number of attributes where the attribute of A is 0 and the attribute of B is 1.

M_{10} : represents the total number of attributes where the attribute of A is 1 and the attribute of B is 0.

M_{00} : represents the total number of attributes where A and B both have a value of 0.

The Jaccard similarity coefficient, J:

		A	
		0	1
B	0	M_{00}	M_{10}
	1	M_{01}	M_{11}

Figure 6: Jaccard similarity

$$J = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

Equation 4

(work with Boolean dataset)

3.2.2 Ranking the users' similarity

The previous techniques calculate the similarity between two users only. So, we calculate the similarity between a user and the others to get an ordered list of people with similar taste to the specified user.

3.2.3 Recommending items

Finding a good critic to read is great, but what I really want is an item recommendation right now. I could just look at the person who has tastes most similar to mine and look for an item he likes that I haven't seen yet, but that would be too permissive. Such an approach could accidentally turn up reviewers who haven't reviewed some of the items that I might like. It could also return a reviewer who strangely liked an item that got bad reviews from all the other critics returned by top Matches.
To solve these issues, you need to score the items by producing a weighted score that ranks the critics. Take the votes of all the other critics and multiply how similar they are to me by the score they gave each item.

Table 1:Rank the Items

Critic	Similarity	item 1	S.item 1	item 2	S.item 2	item 3	S.item 3
person 1	0.99	3.0	2.97	2.5	2.48	3.0	2.97
person 2	0.38	3.0	1.14	3.0	1.14	1.5	0.57
person 3	0.89	4.5	4.02			3.0	2.68
person 4	0.92	3.0	2.77	3.0	2.77	2.0	1.85
person 5	0.66	3.0	1.99	3.0	1.99		
Total			12.89		8.38		8.07
Sim. Sum			3.84		2.95		3.18
Total/Sim. Sum			3.35		2.83		2.53

This table shows correlation scores for each critic and the ratings they gave the three items that I haven't rated. The columns beginning with S. give the similarity multiplied by the rating, so a person who is similar to me will contribute more to the overall score than a person who is different from me. The Total row shows the sum of all these numbers.

We could just use the totals to calculate the rankings, but then an item reviewed by more people would have a big advantage. To correct for this, you need to divide by the sum of all the similarities for critics that reviewed that item (the Sim. Sum row in the table).

These steps general and concept for recommending things to any user. All the previous similarity techniques are tested and had chosen the best based on the dataset type and the resulted generated from the recommendation engine.

3.3 Challenges of CF Recommendation System

3.3.1 the dataset and algorithms used problems

we are working on the codeforces dataset including the problems data and users' data with their submissions. The problems in the codeforces and the user practice on it don't have rating system that every user can rate the problem difficulty or like, dislike the problem. So, we invented rating system for the problems in order to use in recommendation engine.

The rating system we created is to calculate the number of submissions on a specific problem until the accepted answer. So, the problem's rating will be (one over the count of submissions number until the accepted answer) otherwise the problem's rating will be zero.

$$\text{Problem rating} = \frac{1}{\text{Count submission number until accepted answer}}$$

Equation 5

we applied the rating system to all users and the problems they solved and build a massive matrix consists of the users and their problems rating to send this matrix to the recommendation engine to generate the recommended problems suited each user to solve or to challenge their selves.

Table 2:the user-problem matrix real number

	Problem 1	Problem 2	Problem 3	Problem N
User 1	.32	.25	.47	.21
User 2	1	1	0	0
User 3	0	0	.25	.6
User N	.2	0	0	0

When applying the pearson correlation coefficient technique on this matrix to generate the recommended problems to users. The results from the recommendation engine weren't accurate as we want and according to our experiences in problem solving and take long time about 20 second to read the matrix and generate the recommended problems. So, this method doesn't reach to our goal and can't depend on it.

We applying the Euclidean distance score technique on this matrix and the recommendation result as the same as the pearson correlation coefficient technique but the last one results better than Euclidean technique.

3.3.2 the dataset and algorithms used Solutions

The previous problems rating system had an issue when the algorithm applied on the matrix the similarity values were very small that's why the recommended problems to users weren't accurate. So, we thought in another rating system for rating the problems and if the user solves a problem the rating will be 1 otherwise 0.

Table 3:the user-problem matrix Boolean number

	Problem 1	Problem 2	Problem 3	Problem N
User 1	0	0	1	0
User 2	1	1	0	1
User 3	0	0	1	0
User N	1	0	1	0

Using the Boolean values in the matrix data enforce us to use the Jaccard similarity coefficient because it's the only algorithms work only with Boolean data. The generated problems form this technique was very good and better result than the previous techniques we have tried and tested with different users to make sure the recommendation engine work very well, but the algorithm took about 16 seconds to build the data matrix in memory and generate the recommended problems. So, doing optimization to make the algorithms executing faster and better must do to fast serve for user.

3.3.3 Optimization on dataset and algorithm

Using the previous technique for building the matrix data consuming large amount of time and the matrix data full with unimportant data that slows the execution and read the data to build the matrix data. If we are neglecting the unimportant data in files, we can build the matrix data faster using just important data. So, we are building the matrix data in new way that Guarantees low data entry to the matrix and fast execution.

Table 4:User-item relation

User1	Problem1
User1	Problem2
User1	Problem3
User2	Problem3
User2	Problem5
User3	Problem1
User N	Problem N

The new matrix builds in a new concept different than the previous matrices, the new matrix shows the relation between the users and their solved problems only and neglects the unsolved or wrong answer problems.

We customized the Jaccard algorithm to work with the new matrix data. The result the same as the previous Jaccard algorithm with old matrix data but better executing time it runs at 5 seconds for reading the data from file to the matrix and generate the recommended problems to user.

We can optimize the executing time of reading the data in a matrix by reading the data in the matrix once and put it in the application context. So, the recommendation engine won't take long time to recommend and the matrix data will be read if and only if the user update his data.

3.3.4 Cold Start

In the recommendation system collaborative type often require a large amount of existing data on a user in order to make accurate recommendations. So, if the user new in the codeforces and didn't solve any problem, the recommendation engine won't generate any recommendation. The solution for that issue is to put a fixed list of problems for beginner users to solve to Guarantees that has at least one problem solved to make the recommendation engine to generate problems to solve.

3.4 Collaborative filtering

The goal of a collaborative filtering algorithm is to suggest new problems or to predict the utility of a certain problem for a particular user based on the user's previous submissions and solutions for the problems. In a typical CF scenario, there is a list of *m* users U = {*u*1*, u*2*, . . ., um*} and a list of *n* problems I = {*i*1*, i*2*, . . ., in*}. Each user *Ui* has a list of problems *Iui*, which the user has solved. The condition of the problem (solved or not solved) can be explicitly given by the user as a zero if not solved before or one if solved.

Note that *I ui* ⊆I and it is possible for *Iui* to be a *null-set*. There exists a distinguished user *Ua* ∈U called the *active user* for whom the task of a collaborative filtering algorithm is to find a problem likeliness that can be of two forms.

Prediction is a numerical value, *Pa, j*, expressing the predicted likeliness of item *i j* ∈ *Iua* for the active user *Ua*.

Recommendation is a list of *N* items, *Ir* ⊂I, that the active user will like the most. Note that the recommended list must be on problems not already solved by the active user, i.e., *I r* ∩ *Iua* = . This interface of CF algorithms is also known as *Top-N recommendation*.

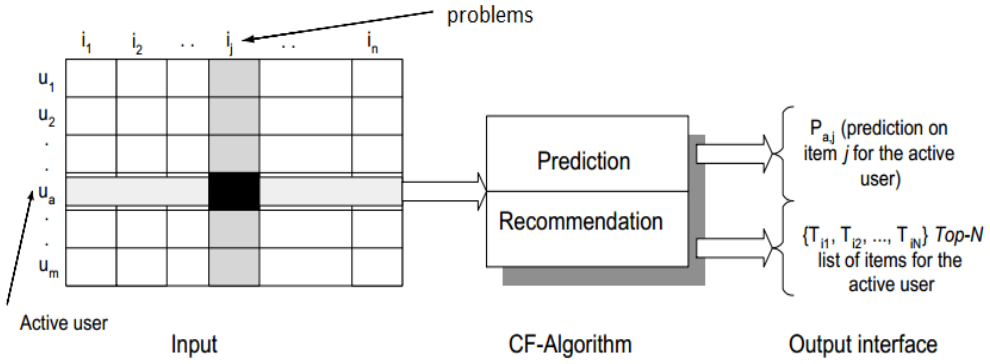

Figure 7: The Collaborative Filtering Process.

Figure 1 shows the schematic diagram of the collaborative filtering process. CF algorithms represent the entire *m* × *n* user-item data as a ratings matrix, A. Each entry *a i, j* in A represent the condition score (zero or one) of the *i*th user on the *j*th problem. Each individual score is 1 if the user has solved the problem before and it can as well be 0 indicating that the user has not yet solved that problem.

3.4.1 Item based Collaborative filtering

the item-based approach looks into the set of items the target user has rated and computes how similar they are to the target item *i* and then selects *k* most similar items {*i*1*, i*2*,. . ., ik*}. At the same time their

corresponding similarities {*si*1, *si*2,. . . , *sik* } are also computed. Once the most similar items are found, the prediction is then computed by taking a weighted average of the target user's ratings on these similar items. We describe these two aspects namely, the similarity computation and the prediction generation in details here.

3.4.1.1 Similarity calculations

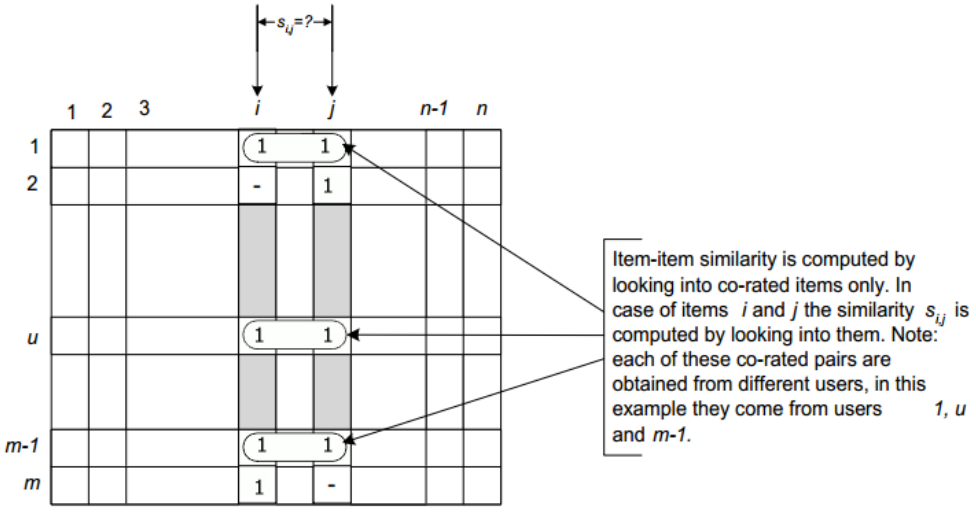

Figure 8: Isolation of the co-rated items and similarity computation

One critical step in the item-based collaborative filtering algorithm is to compute the similarity between items and then to select the most similar items. The basic idea in similarity computation between two items *i* and *j* is to first isolate the users who have rated both of these items and then to apply a similarity computation technique to determine the similarity *si, j* . Figure 2 illustrates this process, here the matrix rows represent users and the columns represent problems. There are a number of different ways to compute the similarity between items. Here we present Four such

methods. These are cosine-based similarity, Jaccard similarity, Euclidean similarity and Manhattan distance.

3.4.1.1.1 Cosine Similarity

In this case, two items are thought of as two vectors in the *m* dimensional user-space. The similarity between them is measured by computing the cosine of the angle between these two vectors. Formally, in the *m* × *n* ratings matrix in Figure 2, similarity between items *i* and *j*, denoted by *sim(i, j)* is given by

$$sim(i, j) = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\|_2 * \|\vec{j}\|_2}$$

Equation 6

now we have two vectors i and j, I = [1,0,1,1,1] j = [1,1,1,1,0]

First step we have to multiply those vectors

Table 5:dot product of two vectors i and j

I	1	0	1	1	1
J	1	1	1	1	0
I . j	1	0	1	1	0

Then should take the summation of ***i*** dot product ***j*** which equals to 3.

Second step: we have to get the summation of the multiplication of each vector itself

$$||i|| = \sum_{x=1}^{n} i_x^{\,2} \;,\; ||j|| = \sum_{x=1}^{n} j_x^{\,2}$$ *Equation 7*

||i|| = 1*1 +0*0+ 1*1 + 1*1 +1*1 = 4

||j|| = 1*1 +1*1+ 1*1 + 1*1 +0*0 = 4

Third step: take the square root for ||i|| and ||j|| and multiply each other

$$|i|_2 * |j|_2 = \sqrt[2]{||I||} * \sqrt[2]{||j||} = \sqrt[3]{4} * \sqrt[3]{4} = 4$$

The final step: last thing that we have to do is dividing I.j (I dot product J) by , so the similarity will be cos(i,j) = 3⁄4 = .75 which means that item i is similar to item j with 75%.

3.4.1.1.2 Jaccard or Tanimoto's definitions of similarity and distance

The **Jaccard index**, also known as the **Jaccard similarity coefficient** (originally coined *coefficient de communauté* by Paul Jaccard), is a statistic used for comparing the similarity and diversity of sample sets. The Jaccard coefficient measures similarity between finite sample sets.

Let's assume that we have two vectors of solved or not solved problem according to *N* users.

Table 6:shows users and their condition (solved/not solved) for specific two problems

Users	Problem **X**	Problem **Y**
U1	1	1
U2	1	0
U3	0	1

U4	1	1
U5	0	0
U6	0	1
U7	1	1

This table represents the solving/not solving records for 7 different users for a specific two problems X and Y which means that U1 solved X and Y but U3 solved only Y. as we previously stated the Jaccard similarity follows this equation.

$$Jaccard\ similarity(X,Y) = \frac{\sum_i (X_i \wedge Y_i)}{\sum_i (X_i \vee Y_i)}$$

Equation 8

First step we have to calculate the summation of anding and oring of the two vectors X and Y.

Table 7:calculating the anding and Oring of the X and Y

Users	Problem **X**	Problem **Y**	$(X_i \wedge Y_i)$	$(X_i \vee Y_i)$
U1	1	1	1	1
U2	1	0	0	1
U3	0	1	0	1
U4	1	1	1	1
U5	0	0	0	0

U6	0	1	0	1
U7	1	1	1	1
-	-	-	3	6

($X_i \wedge Y_i$)column represents the binary AND between each single user condition (1 or 0) in both vectors .

($X_i \vee Y_i$)column represents the binary OR between each single user condition (1 or 0) in both vectors.

Then we have to calculate the summation of each of the two columns and divide the first one by the second one which means that sim(X,Y) = 3/6 = .5

So, from the previous calculation we can simply say that problem X is similar to problem Y with 50 % and vice versa (that problem Y is similar to problem X with 50 %).

3.4.1.1.3 Manhattan distance similarity (city block distance)

Manhattan distance is a metric in which the distance between two points is the sum of the absolute differences of their Cartesian coordinates. In simple way of saying it is the absolute sum of difference between the x-coordinates and y-coordinates. Suppose we have two point A and B if we want to find the Manhattan distance between them, just we have to sum up the absolute x-axis and y – axis variation means we have to find how these two points A and B are varying in X-axis and Y- axis.

$$Manhattan\ distance(X,Y) = \sum_{i=1}^{n} | X[i] - Y[i] |$$

Equation 9

This is the formula of calculating the distance between two coordinates. To use Manhattan distance in calculating the similarity between two different items we have to use the normalized equation.

$$Manhattan\ sim(X,Y) = \sum_{i=1}^{n} 1 - \frac{|\ X[i] - Y[i]\ |}{n}$$

Equation 10

According to table 3.8 we have 2 vectors X and Y represent the user's solving condition for x and y.

X = (1,1,0,1,0,0,1) Y = (1,0,1,1,0,1,1)

n = the size of *X* or *Y*

$$Manhattan\ sim(X,Y\ ) = \sum_{i=1}^{n} 1 - \frac{1+1+1+1+0+1+1}{7} = .142$$

According to the manhattan similarity problem X is similar to problem Y by 0.142 the vice versa is also true.

3.4.1.1.4 Log Likelihood similarity

Log-likelihood–based similarity is similar to the Tanimoto coefficient–based similarity, though it's more difficult to understand intuitively. It's another metric that doesn't take account of individual preference values. The math involved in computing this similarity metric is beyond the scope of this book to explain. Like the Tanimoto coefficient, it's based on the number of problems in common between two users, but its value is more an expression of how *unlikely* it is for two users to have so much overlap, given the total number of items out there and the number of problems each user has solved.

Chapter 4: System Analysis

4.1 Introduction

In this section we will discuss our web application which could be used by users to follow up with their recommended problems, and we will go through main five points to clarify what have been done.

4.2 Technology Stack

As we know, Open source technology is defined as the production and development philosophy of allowing end users and developers to not only see the source code of software, but modify it as well.

So, we focused on using open source technologies to can achieve best results and find a huge community which can assist in case of some troubles.

4.2.1 Java Development Kit

The Java Development Kit (JDK) is an implementation of either one of the Java Platform, Standard Edition, Java Platform, Enterprise Edition or Java Platform, Micro Edition platforms released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris, Linux, Mac OS X or Windows.

4.2.2 Apache Maven

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

4.2.2.1 Used Maven Dependencies

Table 8:Maven Dependencies

#	Group ID	Artifactid	Version
4.2.2.1.1	org.omnifaces	omnifaces	2.3
4.2.2.1.2	org.primefaces	primefaces	5.3.14
4.2.2.1.3	org.eclipse.persistence	eclipselink	2.6.2
4.2.2.1.4	org.eclipse.persistence	org.eclipse.persistence.jpa.modelgen.processor	2.6.2
4.2.2.1.5	Javax	javaee-web-api	7.0
4.2.2.1.6	org.json	json	20160212
4.2.2.1.7	org.jsoup	jsoup	1.7.2
4.2.2.1.8	org.apache.mahout	mahout-mr	0.12.2
4.2.2.1.9	commons-io	commons-io	2.5
4.2.2.1.10	com.fasterxml.jackson.core	jackson-data bind	2.7.5
4.2.2.1.11	com.fasterxml.jackson.core	jackson-annotations	2.7.5
4.2.2.1.12	com.fasterxml.jackson.core	jackson-core	2.7.5

4.2.2.1.1 - Omnifaces Library

OmniFaces is an open source utility library for the JavaServer Faces 2 framework. It was developed using the JSF API, and its aim is to make JSF life easier by providing a set of artifacts meant to improve the functionality of the JSF framework.

4.2.2.1.2 Primefaces Library

The initial development of Primefaces was started in late 2008. Predecessor of Primefaces is the YUI4JSF library, a set of JSF components based on YUI Javascript library. YUI4JSF got cancelled in favor of Primefaces in early 2009.

Since its release, Primefaces has been strongly supported by Oracle, particularly within the NetBeans world.

4.2.2.1.3 Eclipselink Library

EclipseLink is the open source Eclipse Persistence Services Project from the Eclipse Foundation. The software provides an extensible framework that allows Java developers to interact with various data services, including databases, web services, Object XML mapping (OXM), and Enterprise Information Systems (EIS).

4.2.2.1.4 JPA Modelgen Processor Library

The Java Persistence API (JPA) is a Java application programming interface specification that describes the management of relational data in applications using Java Platform, Standard Edition and Java Platform, Enterprise Edition.

4.2.2.1.5 Javaee-Web-API Library

Java Platform, Enterprise Edition or Java EE is a widely used enterprise computing platform developed under the Java Community Process. The platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications.

4.2.2.1.6 JSON Library

JSON is a light-weight, language independent, data interchange format. The files in this package implement JSON encoders/decoders in Java. It also includes the capability to convert between JSON and XML, HTTP headers, Cookies, and CDL. This is a reference implementation.

4.2.2.1.7 JSOUP Library

jsoup is an open-source Java library of methods designed to extract and manipulate data stored in HTML documents.

4.2.2.1.8 - Mahout-MR Library

Apache Mahout is a project of the Apache Software Foundation to produce free implementations of distributed or otherwise scalable machine learning algorithms focused primarily in the areas of collaborative filtering, clustering and classification. Many of the implementations use the Apache Hadoop platform.

4.2.2.1.9 Commons-IO Library (Collection of I/O Utilities)

The Apache Commons is a project of the Apache Software Foundation, formerly under the Jakarta Project. The purpose of the Commons is to provide reusable, open source Java software. The Commons is composed of three parts: proper, sandbox, and dormant.

4.2.2.1.10 Jackson-Data Bind Library

General data-binding functionality for Jackson which works on core streaming API.

4.2.2.1.11 Jackson-Annotations Library

Core annotations used for value types, used by Jackson data binding package.

4.2.2.1.12 Jackson-Core Library

Core Jackson abstractions, basic JSON streaming API implementation.

4.2.3 JSF Framework

JavaServer Faces (JSF) is a Java specification for building component-based user interfaces for web applications. It was formalized as a standard through the Java Community Process and is part of the Java Platform, Enterprise Edition.

4.2.4 Payara Server

Payara Server is a drop in replacement for GlassFish Server Open Source Edition, with the peace of mind of quarterly releases containing enhancements, bugs fix and patches.

Payara Server is the Open Source platform of choice for developing production Java EE applications. Payara Server is a compelling, GlassFish-derived alternative to Oracle's own product roadmap, which requires migration to WebLogic Server. With no need for code rewrites or application re-architecting, Payara Server is a credible solution on which to build your Java middleware platform.

4.2.5 ACE Responsive Admin Template 1.3.3

Ace (v1.3.3) is a lightweight, feature-rich and easy to use admin template based on Bootstrap 3.3.1

4.2.6 Netbeans IDE 8.1

NetBeans is an open-source project dedicated to providing rock solid software development products (the NetBeans IDE and the NetBeans Platform) that address the needs of developers, users and the businesses who rely on NetBeans as a basis for their products; particularly, to enable them to develop these products quickly, efficiently and easily by leveraging the strengths of the Java platform and other relevant industry standards.

4.2.7 Linux Mint Operating System

Linux Mint is the most popular desktop Linux distribution and the third most widely used home operating system behind Microsoft Windows and Apple Mac OS.

The purpose of Linux Mint is to produce a modern, and comfortable operating system which is both powerful and easy to use.

4.3 Architecture Diagram and Overview of System Component

- **Overview of whole system**

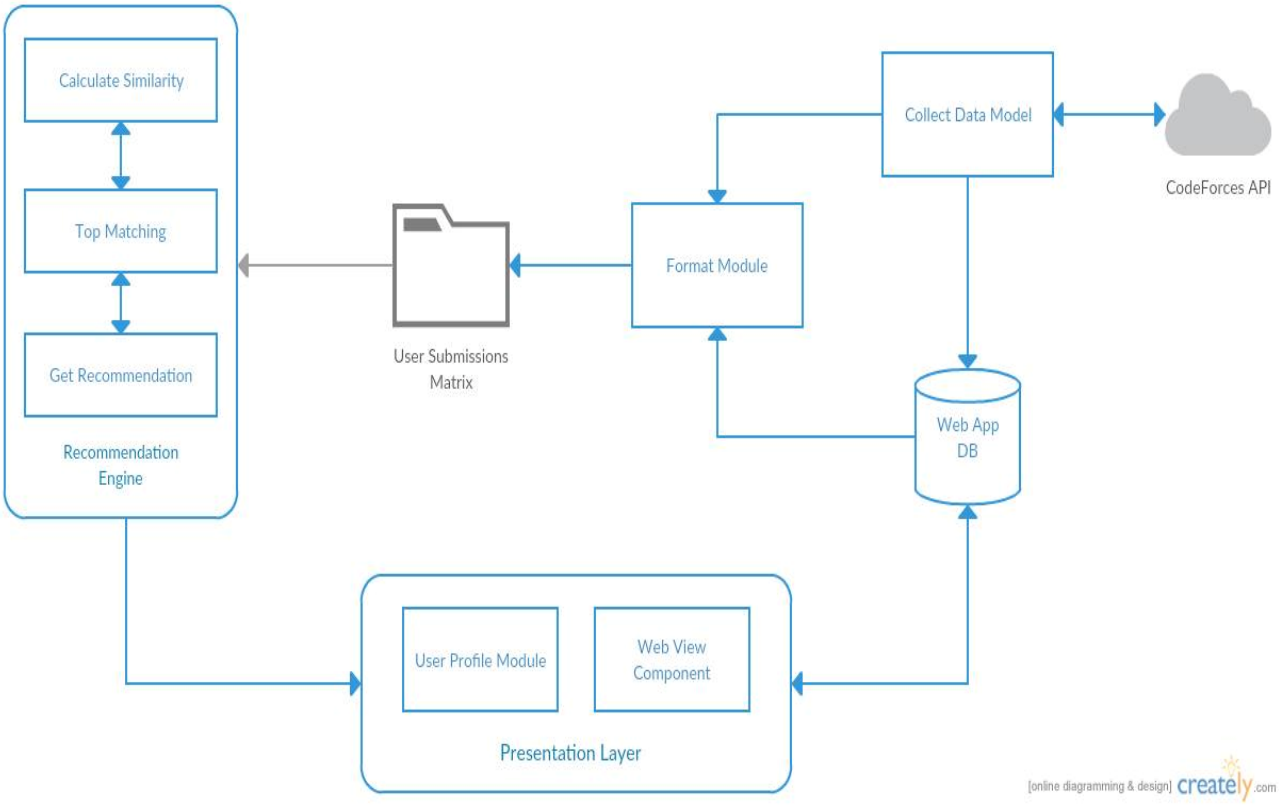

Figure 9:Architecture Diagram

- **Overview of recommendation engine component**

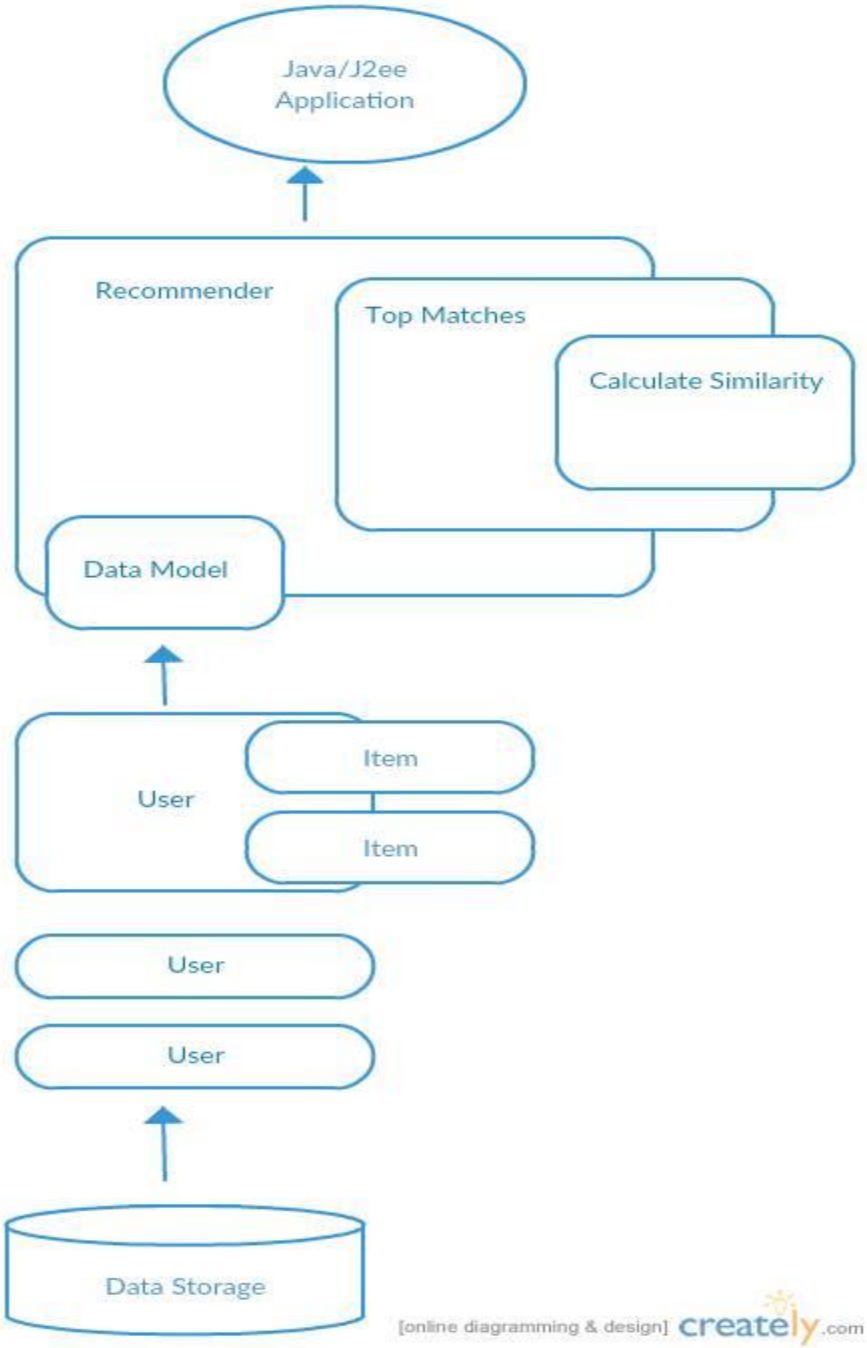

Figure 10:recommendation engine architecture

4.4 Use Case

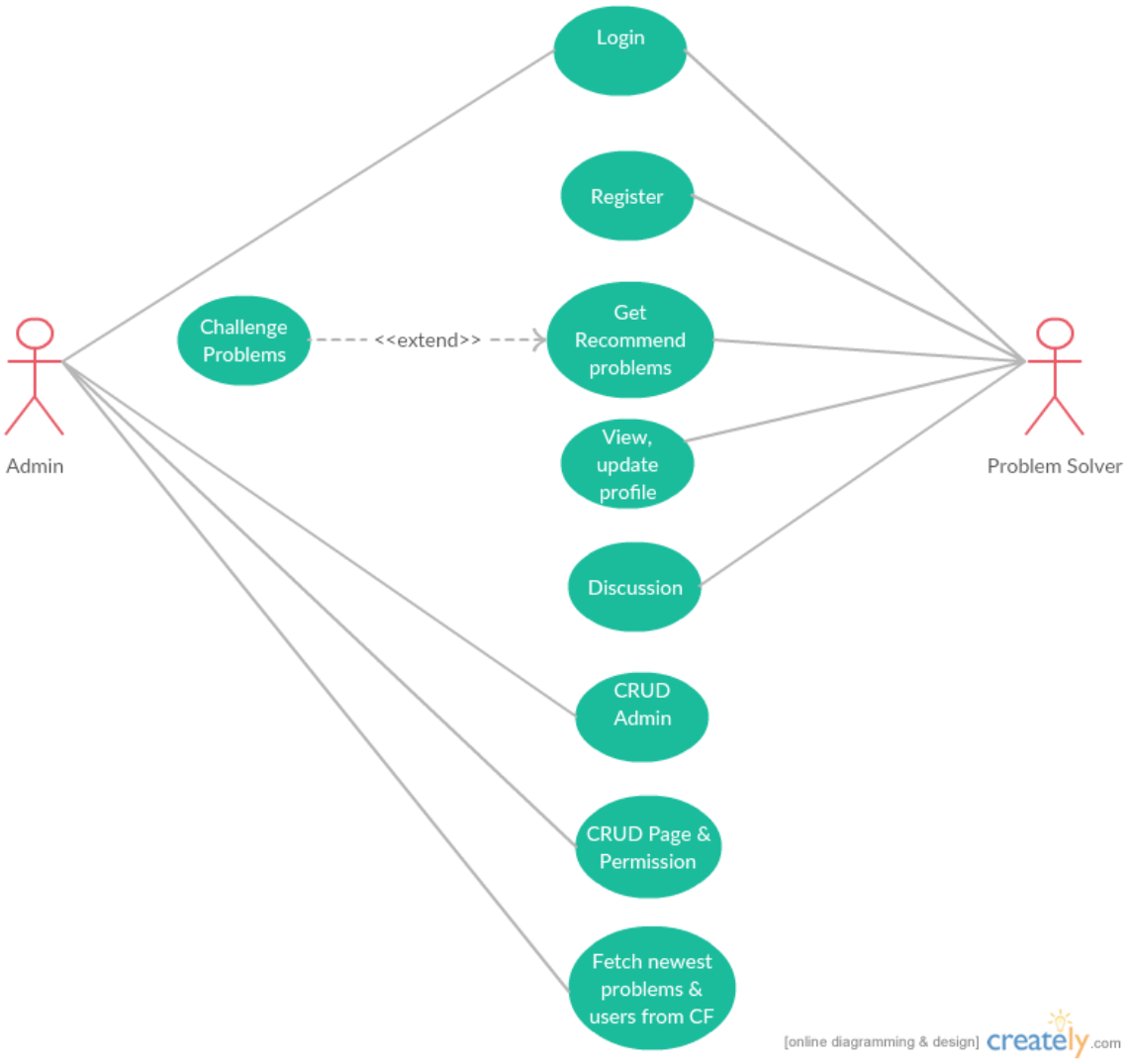

Figure 11:Use Case

In use case diagram above we show the functionality of users (admin, problem solver)

Table 9:Use Case-1

Use case name	Fetch newest problems and users from CF
Use case	Uc-1

ID		
Primary actor	Admin	
Secondary actors		
precondition	The admin is logged into the system	
History	28/2/2016	
Description	The admin can add trigger a function to update the system database with new users and problems	
trigger	The admin choose the monitor section in the site	
Normal flow	**Steps**	
	1	The admin select “update database” button from the admin panel
	2	The system then check for the new problems / users in the system.
	3	The system then update the database if found any new records
	4	The admin then get notified for any changes habbend

Table 10:Use Case-2

Use case name	Get recommendation
Use case id	Uc-2

Primary actor	Problem Solver	
Secondary actors		
Preconditio n	Logged in the system and has a valid handle in CodeForces	
History	Created : 28/2/2016	
Description	This use case enables problem solver to obtain recommended problems to solve based on their profile (History ,rating ,similar users)	
Trigger	When user log in to the system or press the “update problems” button	
Normal flow	Step	
	1	The system gets the problem solver CF-rating ,score, neighbors from his profile
	2	The system generates a list of recommended problems to the user.
	3	The system displays the generated problems to the user for closest neighbors
	4	The solver click the problem link from the listed problems
	5	The system adjusts user’s profile and score after solving each of the recommended problem to adjust his level and generate new sets of recommend problems
	6	The user log out

	7	The system saves the updated user profile and use case ends
Alternative flows	User select challenging problem steps.	
	3.1	The system generates a list of recommended problems to the user higher level of difficulty than he used to base in his history of solved problems
	3.2	The use case continues like the normal flow and goes to step 4.

Table 11:Use Case-3

Use case name	View profile statistics	
Primary actor	Problem solver	
Use case ID	Uc-3	
Secondary actors		
precondition	The user is logged into the system	
History	28/2/2016	
description	The admin can monitor users actions in the site and	
trigger	The User choose the profile statistics section in the site	
Normal flow	**Steps**	

	1	The user select the “user statistics” tap
	2	view his score and solved problems and problems he solved per tag

4.5 Role Based Security

In our web application, we are using an important feature which enable us to provide high security and easy management for the application.

The idea is very simple but very powerful, it is based on defining all pages in application in a database table, so the administrator can indicate which pages can be viewed by which role.

When user logs into the system, the left menu build itself dynamically based on logged user role.

4.5.1 System Users Database Table

In system users table, all users which have an access to the system are registered.

4.5.2 System Pages Database Table

In system pages’ table, all accessible pages are defined, regardless which role can access which page.

4.5.3 System Roles Database Table

In system roles table, all defined roles in the system are defined, so the administrator can assign any pages to any defined role.

4.5.4 System Menus Database Table

There are two table for the menus in the database - Main Menus and Submenus - so, the administrator can define which page is submenu under which main menu, and also he can define a page as main menu or define just labeled main menu which contains submenus.

4.5.5 Permissions Database Table

Under permissions table, the administrator can enable a page for a permission or disable it.

All those functionalities are defined and accessible for system administrators through Administrator Module and we will talk about to explain in details later.

Upon system user logging, the system build left menu dynamically from the defined data fetched from database, so every user only can view the pages which have defined view permission assigned to his role.

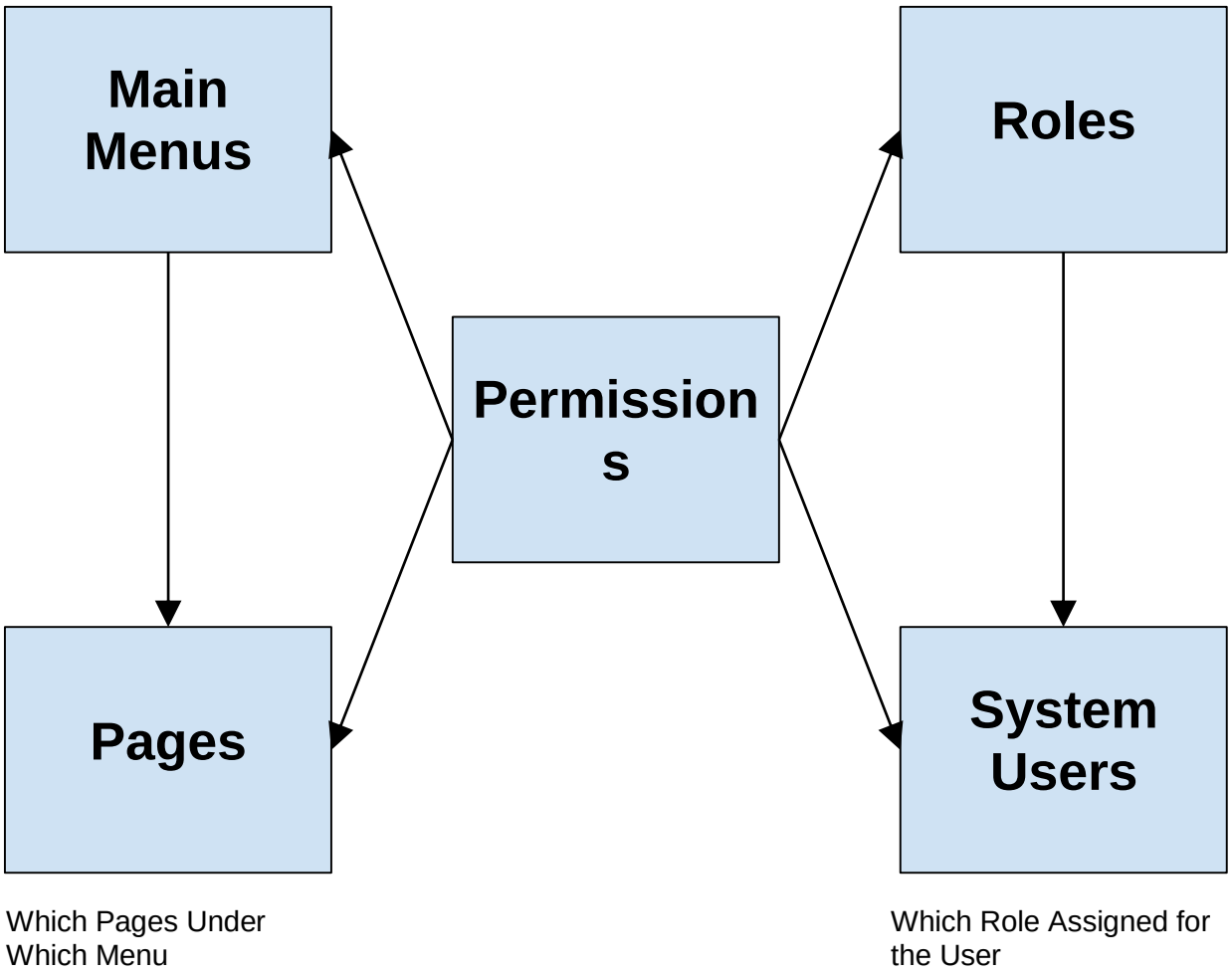
Main Menus
Roles
Permissions
Pages
System Users
Which Pages Under Which Menu
Which Role Assigned for the User

Figure 12

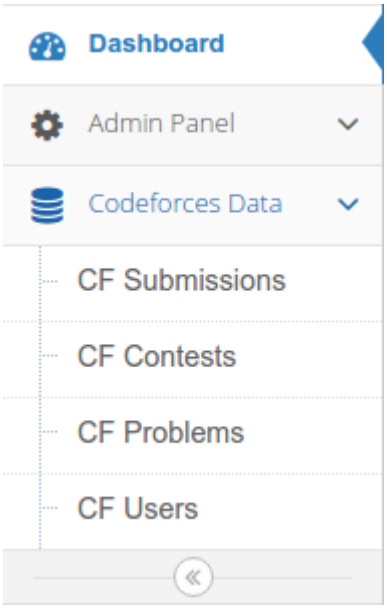

Figure 13

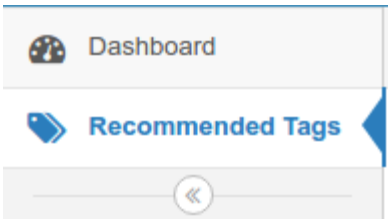

Figure 14

Two screenshots, the first one for administrator and the second one for trainee.

4.6 Administrator Module

In administrator module, the administrator has the ability of using the functionality of role based security in addition to the following:

4.6.1 Update Codeforces Contests

Codeforces is running regular online contests which have from 5 to 7 problems, then contestants register themselves in those contests and compete to level up ranks.

Administrator can update the data with latest contests through codeforces API.

4.6.2 Update Codeforces Problems

Administrator also can update the problems with latest ones through codeforces API to keep the data rich with problems which will be recommended to the users. This done by crawling contests pages to retrieve contest’s problems and it's solved count.

4.6.3 Update Codeforces Users

When users register in our system, they must be registered in the database, so the administrator update the database with latest registered users on codeforces. This done by crawling codeforces website.

4.6.4 Update Codeforces Submissions

Contests submissions is the most important data, because our system relay on it to can generate recommendations for the users, so administrator can update the submissions for the user manually from his dashboard.

All submissions retrieved through codeforces API and returned in form JSON.

All Codeforces APIs responses returned in JSON Format, the system parse this response and generate information from it, then persist this information in the database to can be useable.

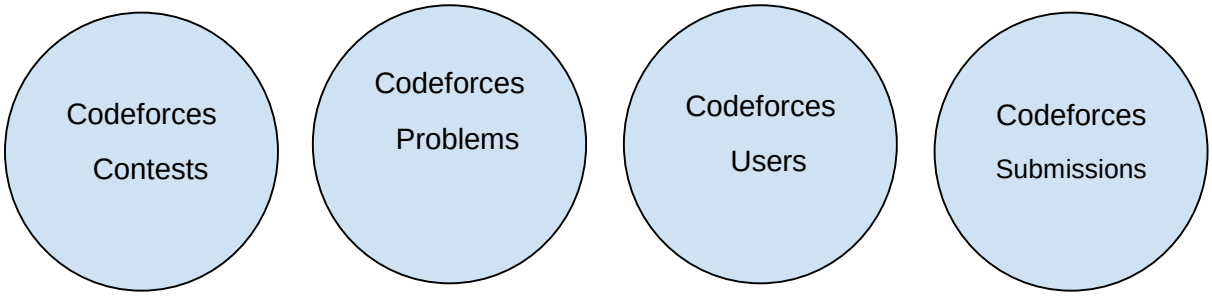

Figure 15

4.7 Trainee Module

The main target of our system is trainees, so they can solve recommended problems and level up.

In this section we will go through the displayed information for the trainees.

4.7.1 Trainee Dashboard

When user logged into the system he will be redirected to his dashboard, where he can view all recommended problems for him from the system.

He can open any problem on codeforces by clicking on its URL, then he can solve the problem and so on.

4.7.2 Recommended Problem's Similar Problems

In the dashboard, user also can get some problems which are similar to a certain recommended problem for him, so he can also solve these problems to level up.

4.7.3 Trainee Recommended Tags

In the left menu, trainee can also go to recommended tags page which contains groups of problems categorized by its tags - as we know each problem in codeforces have some tags e.g. greedy / implementation / binary search, etc…

So, trainee can solve some recommended problems in certain category which recommended to him by the system.

4.7.4 Refresh Submissions

Trainee can also refresh his submissions to update registered submissions on the system, as soon as user refresh his submissions, recommended problems for him changes immediately based on his latest submissions fetched from codeforces through codeforces API.

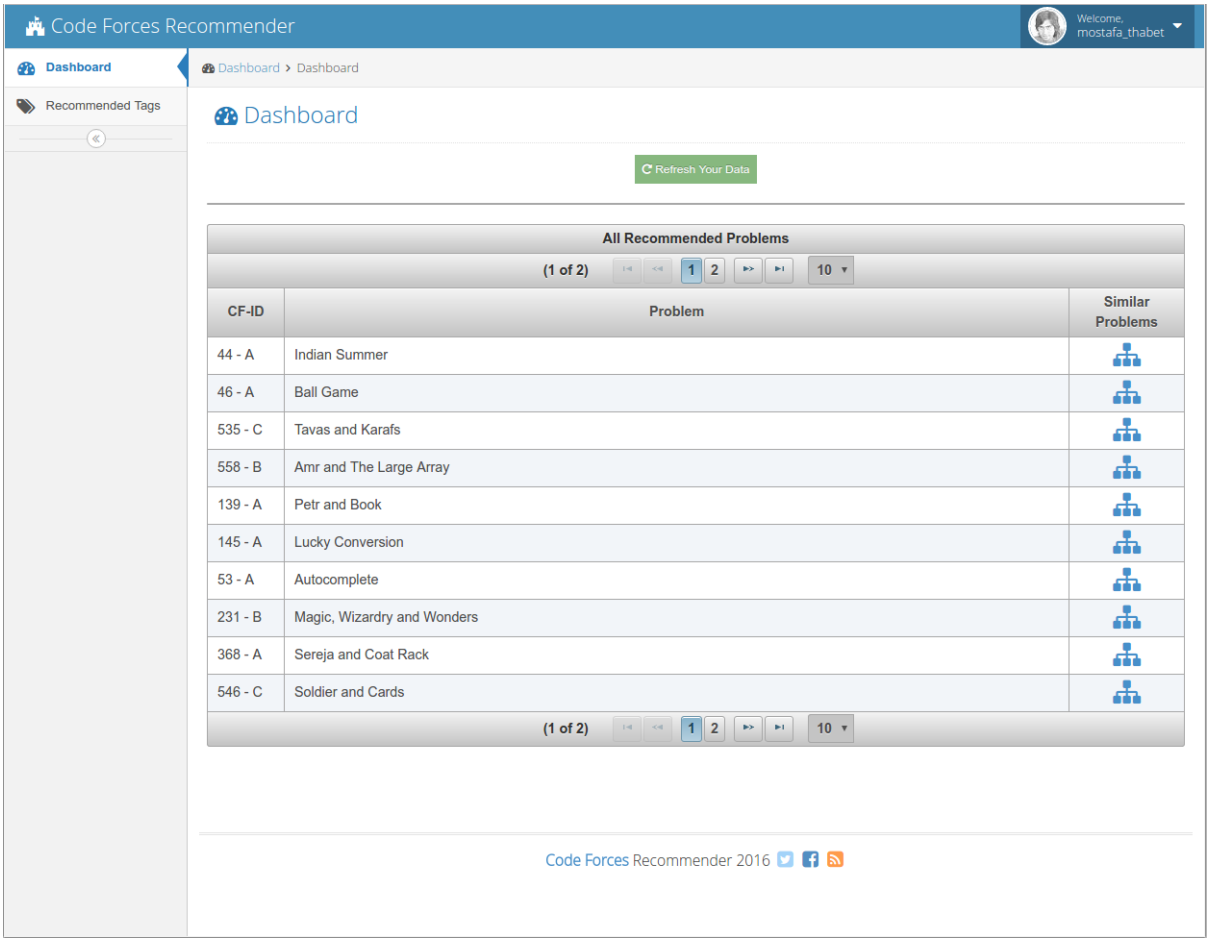

Figure 16:Screenshot from Trainee Dashboard

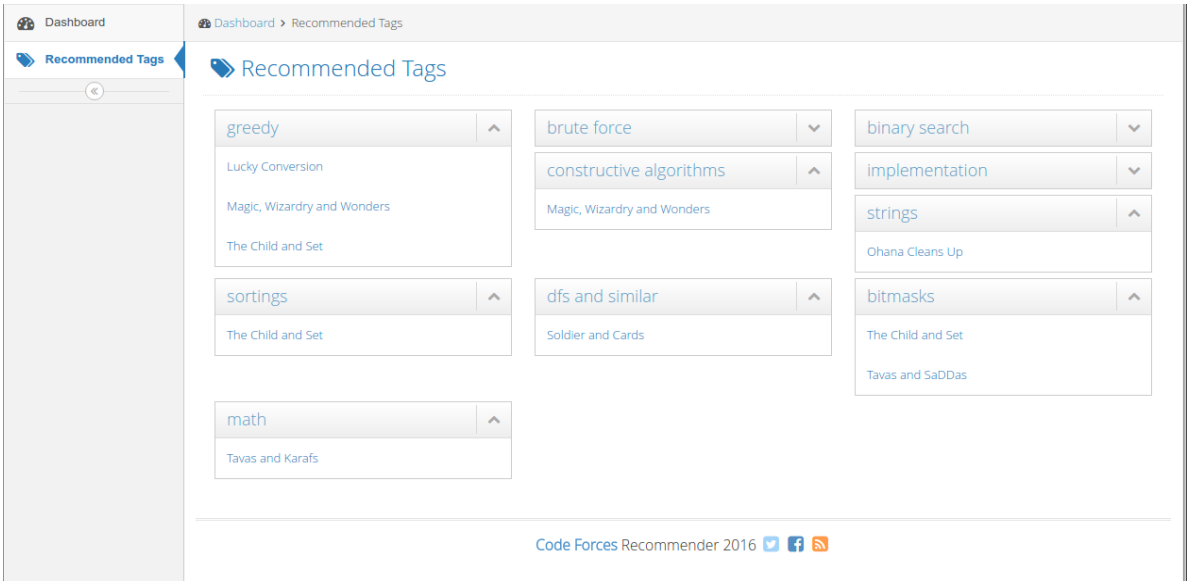

Figure 17:Screenshot from Trainee Recommended Problems

4.8 Codeforces APIs

With codeforces API we can get access to some of codeforces data in machine-readable JSON format.

To access the data, we just send a HTTP request to address http://codeforces.com/api/{methodName} with method specific parameters.

In this section we will go through used codeforces API methods in our system.

4.8.1 Contest.list

Returns information about all available contests.

Parameter	Description
gym	Boolean. If true — than gym contests are returned. Otherwide, regular contests are returned.

Return value: Returns a list of Contest objects. If this method is called not anonymously, then all available contests for a calling user will be returned too, including mashups and private gyms.

Figure 18

4.8.2 Contest.status

Returns submissions for specified contest. Optionally can return submissions of specified user.

Parameter	Description
contestId (Required)	Id of the contest. It is **not** the round number. It can be seen in contest URL. For example: /contest/**566**/status
handle	Codeforces user handle.
from	1-based index of the first submission to return.
count	Number of returned submissions.

Return value: Returns a list of Submission objects, sorted in decreasing order of submission id.

Figure 19

4.8.3 User.info

Returns information about one or several users.

Parameter	Description
handles (Required)	Semicolon-separated list of handles. No more than 10000 handles is accepted.

Return value: Returns a list of User objects for requested handles.

Figure 20

4.8.4 User.status

Returns submissions of specified user.

Parameter	Description
handle (Required)	Codeforces user handle.
from	1-based index of the first submission to return.
count	Number of returned submissions.

Return value: Returns a list of Submission objects, sorted in decreasing order of submission id.

Figure 21

4.8.5 Samples from Returned Objects from Codeforces APIs

4.8.5.1 Problem

Field	Description
contestId	Integer. Id of the contest, containing the problem.
index	String. Usually a letter of a letter, followed by a digit, that represent a problem index in a contest.
name	String. Localized.
type	Enum: PROGRAMMING, QUESTION.
points	Floating point number. Can be absent. Maximum ammount of points for the problem.
tags	String list. Problem tags.

Figure 22

4.8.5.2 Submission

Field	Description
id	Integer.
contestId	Integer.
creationTimeSeconds	Integer. Time, when submission was created, in unix-format.
relativeTimeSeconds	Integer. Number of seconds, passed after the start of the contest (or a virtual start for virtual parties), before the submission.
problem	Problem object.
author	Party object.
programmingLanguage	String.
verdict	Enum: FAILED, OK, PARTIAL, COMPILATION_ERROR, RUNTIME_ERROR, WRONG_ANSWER, PRESENTATION_ERROR, TIME_LIMIT_EXCEEDED, MEMORY_LIMIT_EXCEEDED, IDLENESS_LIMIT_EXCEEDED, SECURITY_VIOLATED, CRASHED, INPUT_PREPARATION_CRASHED, CHALLENGED, SKIPPED, TESTING, REJECTED. Can be absent.
testset	Enum: SAMPLES, PRETESTS, TESTS, CHALLENGES, TESTS1, ..., TESTS10. Testset used for judging the submission.
passedTestCount	Integer. Number of passed tests.
timeConsumedMillis	Integer. Maximum time in milliseconds, consumed by solution for one test.
memoryConsumedBytes	Integer. Maximum memory in bytes, consumed by solution for one test.

Figure 23

Chapter 5: Results, Conclusion and future work

5.1 Results

Item based

As we stated previously in the Methodology chapter under the item based collaborative filtering we tried more than one similarity measure between problems to retrieve the most relevant problems to specific problem.

To use the most suitable similarity measure in our case we make a comparison between the 4 similarities that we use we gave each similarity algorithm a problem and it's supposed to response with set of relevant problems recommended to the user based on the similarity measurements between the given problem and all other problems.

Experiment

We gave a problem of type A to the four algorithms and the result was as follow

Table 12:show the recommendations from different similarity measures

Problem Type	Jaccard	Log likelihood	Manhattan	Cosine
A	12	14	13	2
B	2			3
C				3
D				3
E				3

From the previous recommendations we found that the Jaccard is the most accurate similarity measure as it retrieve similar problems to the given one but other similarities give bad results and not relevant for the given problem.

We also tested the Jaccard item based similarity with a group of problem solvers and they found it more accurate than the others similarities as it provides them with similar and relevant problems suitable to their style.

User based

We tested the recommendation engine throw few live users as it's proved to be the best way to test recommendation systems users who used the actual systems

Rating Bounds	Color	Title	Division	Number	Number (by color)
2900+	Red	Legendary Grandmaster	1	4	183
2600 — 2899	Red	International Grandmaster	1	46	
2400 — 2599	Red	Grandmaster	1	133	
2300 — 2399	Orange	International Master	1	163	380
2200 — 2299	Orange	Master	1	217	
1900 — 2199	Violet	Candidate Master	1	1253	1253
1600 — 1899	Blue	Expert	2	5095	5095
1400 — 1599	Cyan	Specialist	2	8202	8202
1200 — 1399	Green	Pupil	2	5736	5736
0 — 1199	Gray	Newbie	2	2319	2319

Figure 24: Codeforces Ranks

And most of their feedback were positive that the recommended problems were at their level and we also tested some of the users handles ourselves and found that closest neighbors to the user were similar to users.

Example 1

We gave the algorithm a user called Tourist (red user with rate 3525) and the algorithms recommended 2 nearest users

Table 13: 2 recommended users for tourist

Name	Rating	title
KADR	2480	International grandmaster
Ra16bit	2457	International grandmaster

As we see in the table 11 the two recommended users are so close to tourist and they have the same title and same color.

Example 2

In example 2 we gave the algorithm a user called adel (gray user with rate 989) and the algorithms recommended 2 most similar users

Table 14:similar users to adel

Name	Rating	title
hatem_toma	993	International grandmaster
pandusonu	1478	specialist

In table 12 the two recommended users to adel falls in his range and not only the rate the controller of the recommendation but also the number of problems and its level plays a high role.

Example 3

In example 3 we gave the algorithm a user called mostafa_thabet (green user with rate 1258) and the algorithms recommended 2 most similar users

Table 15:2 recommended users for mostafa_thabet

Name	Rating	title
Code Tamer	1417	specialist
Ahmed_makram07	1266	pupil

Table 13 shows the two recommended users and they fall in the trainee range.

We have tried to recommend problems using data from code forces only. This project could be turned to a bigger platform that include many other online judges and users' data from various website and as we kept scalability in mind we our project will allow for that and more things could be added like

1-analyzing the submission code for problems and provides hints for coder style

2- Implementing expert system module that take many ACM coaches approaches in training users with various technique

And as we used DataModel in representing data this platform could be used to work above apache spark (it is a fast and general engine for large scale data processing.) for more scalability and will allow for fast in memory data processing.

References

[1] C´esar Vialardi, Javier Bravo, Leila Shafti, and Alvaro Ortigosa. Recommen- ´ dations in Higher Education Using Data Mining Techniques. In Proceedings of the 2nd International Conference on Educational Data Mining (EDM09), pages 190–199, 2009.
[2] Anne Yun-An Chen and Dennis McLeod. Collaborative filtering for information recommendation systems. Encyclopedia of E-Commerce, E-Government, and Mobile Commerce, IGI Global, pages 118–123, January 2006 (good for types).
[3] G. Linden, B. Smith, and J. York. Amazon.com recommendations: itemto-item collaborative filtering. IEEE Internet Computing, 7(1):76–80, 2003.
[4] J. Bobadilla, F. Ortega, A. Hernando, J. Alcalá, Improving collaborative filtering recommender systems results and performance using genetic algorithms, Knowledge Based Systems 24.
[5] J. Bobadilla, F. Serradilla, A. Hernando, Collaborative filtering adapted to recommender systems of e-learning, Knowledge Based Systems 22 (2009)
[6] X. Yang, Y. Guo, Y. Liu, and H. Steck, “A survey of collaborative filtering based social recommender systems”, IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 6, No 2, November 2013 ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784 www.IJCSI.org 166 In the Journal of Computer Communications, 2013.
[7] L. Candillier, F. Meyer, M. Boullé, Comparing state-of-the-art collaborative filtering systems, Lecture Notes in Computer Sciece 4571 (2007).
[8] D.H. Park, H.K. Kim, I.Y. Choi, J.K. Kim, A literature review and classification of recommender Systems research, Expert Systems with Applications 39 (2012).

[9] Berry, M. J. A and G. S. Linoff, Data Mining Techniques for Marketing, Sales and Customer Relationship Management, Second Edition, Wiley, 2004.
[10] Cho, Y. H., J. K. Kim and S. H. Kim, “A person- Deuk Hee Park. Hyea Kyeong Kim. Il Young Choi. Jae Kyeong Kim 150 alized Recommender System Based on Web Usage Mining and Decision Tree Induction”, Expert System Applications, Vol.23, No.3(2002).
[11] Berry, M. J. A and G. S. Linoff, Data Mining Techniques for Marketing, Sales and Customer Relationship Management, Second Edition, Wiley, 2004
[12] Lihua, W., L. Lu, L. Jing, and L. Zongyong, “Modeling User Multiple Interests by an Improved GCS Approach”, Expert Systems with Applications, Vol.29, No4.(2005).
[13] Kim, J. K., Y. H. Cho, W. J. Kim, J. R. Kim and J. H. Suh, “A Personalized Recommendation Procedure for Internet Shopping”, Electronic Commerce Research and Applications, Vol.1, No.3-4(2002).
[14] Kim H. K., J. K. Kim, and Y. U. Ryu, “Personalized Recommendation over a Customer Network for Ubiquitous Shopping”, IEEE Transactions on Services Computing, Vol.2, No.2 (2009)
[15] Anders, U. and O. Korn, “Model Selection in Neural Networks”, Neural Networks, Vol.12, No.2(1999).
[16] Malhotra, N. K., Marketing Research: An Applied Orientation, Fifth Edition, Pearson Education Inc, 2007.
[17] M. Balabanovic, Y. Shoham, Content-based, collaborative recommendation, Communications of the ACM 40 (1998)
[18] K. Yu, A. Schwaighofer, V. Tresp, X. Xu, H.-P. Kriegel, Probabilistic memory based collaborative filtering, IEEE Transactions on Knowledge and Data Engineering 16 (2004)
[19] K. Yu, X. Xu, J. Tao, M. Ester, H.-P. Kriegel, Instance selection techniques for memory-based collaborative filtering, in: Proceeding of the Second SIAM International Conference on Data Mining (SDM’ 02), 2002

[20] G. Takács, I. Pilászy, B. Németh, D. Tikk, Scalable collaborative filtering approaches for large recommender systems, Journal of Machine Learning Research 10 (2009)
[21] Good, N., Schafer, B., Konstan, J., Borchers, A., Sarwar, B., Herlocker, J., and Riedl, J. (1999). Combining Collaborative Filtering With Personal Agent for better Recommedations. In Proceedings of the AAAI'99 conference, pp. 439-446
[22] Konstan, J., Miller, B., Maltz, D., Herlocker, J., Gordon, L., and Riedl, J. (1997). GroupLens: Applying Collaborative Filtering to Usenet News. Communications of the ACM, 40(3), pp. 77-87.
[23] Resnick, p., lacovou, N., Suchak, M., Bergstrom, p>, and Riedl, J. (1994). GroupLens: An Open Architechture for Collaborative Filtering of Netnews. In Proceedings od CSCW '94, Chapel Hill, NC.
[24] Shardanand, U., and maes, P. (1995). Social Information Filtering: Algorithms for Automating 'Word of Mouth'. In Proceedings of CHI '95. Denver, CO.
[25] Hill, W., Stead, L., Rosenstein, M., and furnas, G. (1995). Recommendation and Evaluating Choices in a Virtual community of Use. In Proceedings of CHI '95.
[26] Resnick, P., and Varian, H. R. (1997). Recommender Systems. Special issue of Communications of the ACM. 40(3).
[27] Hill,W., Stead,L., Rosenstein,M., Furnas,G.W., 1995.Recommending and Evaluating Choices in a Virtual Community of Use. Proceedings of ACM CHI'95 Conference on human factors in computing systems. Denver, CO., (pp. 194-201).
[28] Explaining Collaborative Filtering Recommendations (Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl Dept. of Computer Science and Engineering University of Minnesota Minneapolis, MN 55112 USA)
[29] Konstan,J.A., Miller,B.N., Maltz,D., Herlocker,J.L., Gordon,L.R., Riedl,J., 1997. GroupLens: Applying collaborative filtering to Usenet news. Communications of the ACM 40 (3), 77-87.

[30] Resnick,P., Iacovou,N., Suchak,M., Bergstrom,P., Riedl,J., 1994. GroupLens: An open architecture for collaborative filtering of netnews. Proceedings of1994 Conference on Computer Supported Collaborative Work. (pp. 175-186).
[31] Shardanand,U., Maes,P., 1995. Social Information Filtering: Algorithms for Automating "Word of Mouth". Proceedings of ACM CHI '95. Denver, CO., (pp. 210-217).
[32] Dahlen,B.J., Konstan,J.A., Herlocker,J.L., Good,N., Borchers,A., Riedl,J., 1998. Jump-starting movielens: User benefits of starting a collaborative filtering system with "dead data".University of Minnesota TR 98-017.