CodeForces Problem Recommender System

Devansh Shah[1], Devasy Patel[1], and Dwij Bavisi[1]

Institute of Technology, Nirma University Ahmedabad, India

Abstract. Our paper proposes a similarity-based problem recommender system for Codeforces that recommends relevant programming problems to users based on their past performance. We also analyze learning curves to understand how users improve their skills on the platform. Our findings suggest that users reach a peak in the number of attempts per question as they become more comfortable with the platform. While our proposed system shows promising results, we identify several potential avenues for future work to further improve its accuracy and effectiveness, such as incorporating LSTM and CNN algorithms and utilizing demographic information for personalization. We hope our work will contribute to the development of more effective problem recommender systems for competitive programming platforms like Codeforces.

Keywords: competitive programming · Codeforces · learning curves · problem recommender system · user behavior

1 Introduction

With the advancement in technology, humans now want best results in the least time. Hence, Recommendation Systems are widely prevalent. They are algorithms that provide personalized suggestions to users, aiming to enhance their experience by surfacing relevant and interesting content. They are widely used in various domains, such as E-Commerce, Online Advertising, Music Streaming, and Social Networking platforms. Recommendation systems help businesses increase user engagement, improve customer satisfaction, and boost revenue by promoting items that users are more likely to purchase or engage with. Figure 1

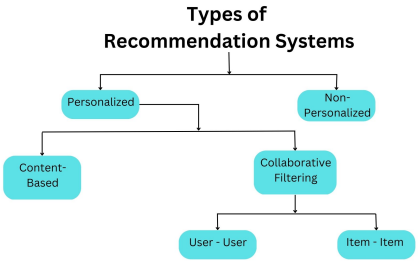

Fig. 1. types of Recommendation System

shows the different types of Recommender Systems. Recommendation systems have become an integral part of our digital lives, enabling us to find relevant content, products, and services based on our preferences, behaviors, and interests. Over the years, these systems have evolved from simple, traditional approaches to more sophisticated, data-driven methods.

1.1 Types of Recommender Systems

The two primary categories of recommender systems are:

Personalized Recommender Systems Personalized recommendation systems are designed to offer tailored recommendations to individual users based on their preferences, behaviors, and prior engagement with the system. These systems leverage user data such as browsing history, search queries, purchase history, and social media activity to make recommendations. The process of personalization involves pairing the right users with the right services, products, or information. When executed successfully, it increases user engagement and thus boosts the business. It is further classified into 2 categories [6].

1. Content-Based Filtering: Content-based filtering is an early recommendation technique that focuses on the attributes of items to recommend similar content to users. It works by building user profiles based on the content they have interacted with or consumed in the past, such as keywords, genres, or item descriptions. The system then recommends items with similar attributes to the user's preferences. Figure 2 offers a glimpse into the architecture of content-based Recommender Systems. The main objective is to suggest products or services that are comparable to those that users have already enjoyed or are now utilising. Although content-based filtering offers a simple and intuitive approach, it has drawbacks, including a dearth of variety in suggestions and the inability to recommend new items in the absence of prior data. [19]. The lack of diversity is a serious drawback because diverse content won't be recommended to users and thus, users can get bored by the same type of feed [9].

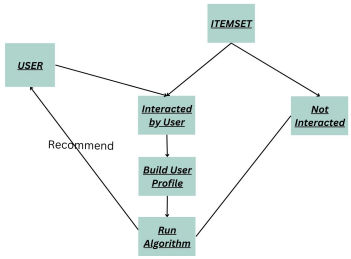

Fig. 2. Architecture of content-based RS

2. Collaborative Filtering: Collaborative filtering is another recommendation technique that leverages user-item interaction data to generate recommendations. It is a technique that can filter out items that a user might like based on either the opinions of other users who share their interests or the similarity between items. Figure 3 offers a glimpse into the architecture of content-based Recommender Systems.

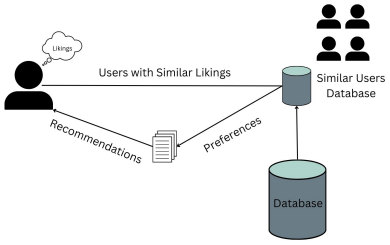

Fig. 3. Architecture of collaborative filtering based Recommender System

3. (a) User-based collaborative filtering: User-based collaborative filtering identifies users with similar preferences or behaviors and suggests items that these similar users have liked or interacted with. The system first creates a user-item matrix, where the rows represent the users and the columns represent the items. Each entry in the user-item matrix corresponds to the user's rating or interaction with the item. The matrix is then used to identify users who have similar preferences, based on the similarity between their ratings or interactions. The algorithm recommends products that the target user has not yet engaged with but that the comparable users have enjoyed or interacted with. This approach is called user-based collaborative filtering because it focuses on the similarity between users. The key advantage of this approach is that it can provide personalized recommendations based on the users' past interactions and preferences. Additionally, it also provides diverse content.
   (b) Item-based collaborative filtering: Item-based collaborative filtering focuses on finding item by finding other items that are similar to the ones the user has previously interacted with. It works by first creating a matrix of user-item interactions, where the columns are the items and the rows are the users. The matrix is then transformed into an item-item similarity matrix, which is calculated by measuring the similarity between pairs of items using centred-cosine similarity. Once the item-item similarity matrix is computed, the system can use it to make recommendations for a given user by identifying the items that are most similar to the ones the user has previously interacted with. To do this, the system looks for items that have high similarity scores with the user's previously interacted items and recommends those items to the user.s that are similar

to those a user has previously interacted with and recommends similar items.

Churn rate is a measure of the rate at which customers stop using a product or service over time. Item-Item Collaborative Filtering is better than User-User because while User's preferences change with time (Churn Rate), the Item data more or less remains constant.

Non-Personalized Reommender Systems Non-personalized recommendation systems provide the same recommendations to all users without considering their individual preferences. These systems are usually based on simple rules such as popularity, price, or ratings. For example, in Youtube, the top trending videos are also recommended to users along with personalized recommendations. Their main use-cases involve systems where user data is scarce or when the cost of collecting and analyzing user data is high. They can also be used as a baseline for evaluating the performance of personalized recommendation systems.

1.2 Evolution of Recommender Systems

1. Hybrid recommendation systems Hybrid systems combine the advantages of both content-based and collaborative filtering approaches to address their individual limitations [22]. These systems leverage both item attributes and user-item interactions to provide more diverse and accurate recommendations. By integrating different techniques, hybrid systems can overcome cold-start problems, improve scalability, and deal with data sparsity.
2. Deep learning-based recommendation systems In recent years, the advent of deep learning has revolutionized recommendation systems. These models, such as neural networks and deep autoencoders, can extract complex and abstract features from large-scale data, leading to more accurate and personalized recommendations. Deep learning-based systems are particularly effective in dealing with unstructured data, such as images, text, and audio, making them ideal for applications like image and music recommendation. For example, the authors of [5] have suggested an approach that takes user dynamics into account while employing an LSTM network-based collaborative filtering engine to analyze user rating sequences. The visual elements of movie posters and trailers are retrieved by the content-based filtering engine using CNN, and along with the actors and directors, related films are suggested to the viewer.
3. Demographic Recommendation Systems Involve taking into account User's Demographic Information like Gender, Age, Occupation and Locality to provide more relevant Recommendations. Authors in paper [4] explore the potential of demographic data in recommendation systems. The authors proposed that demographic data can provide additional context to user-item interactions and help address the cold-start problem for new users. Thus, demographic data can be used to segment users into groups and recommend items based on group preferences. However, it can also lead to biases and privacy concerns if not handled properly.

1.3 Applications

Recommendation systems are widely used and few of their important use cases are summarized in Figure 4.

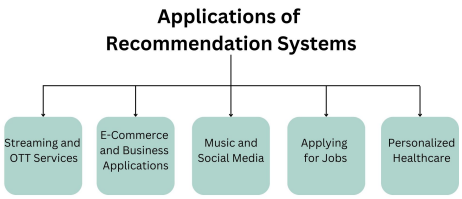

Fig. 4. Applications Of RS

1. Movie Recommendations: The majority of streaming services, including Netflix and Amazon Prime Video, employ recommendation engines for movies. Users receive personalised movie suggestions from these systems based on their viewing interests and history. One such system is the Netflix recommendation algorithm, which uses a combination of collaborative filtering, content-based filtering, and matrix factorization techniques to recommend movies to users [18].
2. E-commerce Recommendations: E-commerce websites like Amazon and eBay use recommendation systems to suggest products to customers based on their purchase history and browsing behavior. These systems use collaborative filtering, association rule mining, and matrix factorization techniques to generate product recommendations. For example, if you add an item to the cart, Amazon's recommendation system suggests related products to users in the "People also bought" section [12].
3. Music Recommendations: Music streaming services like Spotify and Pandora use recommendation systems to make recommendations for users' playlists and songs based on their listening patterns and musical taste along with past records. These systems use techniques like collaborative filtering, content-based filtering, and matrix factorization to generate music recommendations. For example [8], Spotify's recommendation algorithm [1] suggests personalized playlists to users based on their listening history and preferred genres.
4. Job Recommendations: Online job portals like LinkedIn and Indeed use recommendation systems to suggest job openings to job seekers based on their skills, experience, and job preferences. These systems use collaborative filtering, content-based filtering, and natural language processing techniques to generate job recommendations. For example, LinkedIn's job recommendation algorithm suggests job openings to users based on their job history and

skills. Moreover, Social Media Platforms like Instagram and Linkedin (Reference: "Recommending Job Opportunities with Recurrent Neural Networks on Graphs" by Agarwal et al.)
5. Health Recommendations: Health care providers use recommendation systems to suggest personalized treatment plans to patients based on their medical history and health status. These systems use machine learning techniques like decision trees, clustering, and rule-based systems to generate treatment recommendations. For example, the MIMIC-III dataset contains electronic health records of patients, which can be used to develop recommendation systems for personalized treatment plans. (Reference: "A Survey of Data Mining Techniques for Social Media Analysis" by Chen et al.)

Limitations In this section we put forward limitations of existing methodologies and techniques. A detailed overview has been obtained from paper [20].

1. Data Sparsity: The degree of suggestion quality is significantly impacted by data sparsity. Data is shown as a user-item matrix that is populated with pertinent information (such as ratings in the case of a movie recommendation system). The matrix's dimensions greatly expand as both users and goods expand. Out of these users, the majority are new or have rated very few items thus leading to more sparsity.
2. Cold Start Problem: The base of recommendation systems is the data - be it users data or items data. Now, a problem arises when a new user arrives. As we don't have any information about the past history and ratings of the new user, the question is what to recommend? Similarly, in the case of a new item, there is not enough user-item interaction data to accurately calculate item-item similarity.
3. Lack of Diversity: The principal motivation of recommendation systems was to provide personalized experience to users. However, if only a similar type of content is being shown to the user, it won't be appropriate. For example, on Instagram, if we interact with a reel of pets, my entire feed will be filled with pet reels. Similarly, if we click on an advertisement of shoes, all other advertisements will also be of shoes.
4. Scalability: The recommender systems are experiencing an explosion of data due to the massive boom in online content, making it extremely difficult to keep up with the growing demand. Some algorithms used by recommender systems deal with calculations that have become more complex and difficult as the number of users and items increases. Computations in collaborative filtering increase exponentially in cost and can occasionally produce unreliable results.

1.4 Motivation - Competitive Programming

In competitive programming (CP), programmers compete with one another to solve algorithmic puzzles in a predetermined amount of time. It challenges competitors to work through difficult challenges that put your programming knowledge, analytical skills, and under-pressure ingenuity to the test. CP is usually

done online, and there are various platforms such as Codeforces, Topcoder, and HackerRank where programmers compete against each other [16].

Competitive Programming is important for a variety of reasons:

It is an amazing way to improve your programming skills, as it exposes you to a wide variety of programming problems and techniques. CP problems are often designed to be challenging and require a deep understanding of algorithms and data structures. By solving these problems, you can improve your coding skills, learn new programming languages, and gain exposure to new programming paradigms. It is a stepping stone to a career in programming. The logic developed by solving various problems helps solve even complex problems at the Industrial Level. Moreover, it improves the ability to write efficient and well-structured code along with amazing debugging ability. Hence, Employers value Competitive Programming Contests and scores. Most companies have coding rounds to filter out the candidates. Moreover, participating in coding contests can lead to opportunities for internships, scholarships, and other rewards. ICPC is a global level contest wherein top coders from across the globe participate in the contest. Along with the competitive environment, it also provides an opportunity to connect with other programmers from around the world. The competitive programming community is a supportive and vibrant one, and participating in online contests and events is a great way to meet like-minded people and build a network of contacts in the industry.

1.5 Contribution

Novelty: Competitive Programming has a major role to play in the development of logic and clear understanding of data structures and algorithms. One of the key problems of all coders is where to start, there are so many platforms available online, each having thousands and thousands of questions, making it a 'problem of plenty'. Thus, it is vital that we provide a personalized recommendation system to recommend which questions to do next in order to improve. The uniqueness in our recommendation system is that it is designed to help learners choose the most appropriate programming problem based on their current knowledge level and progress. The system analyzes the learner's performance on previous programming problems, such as the question type (implementation, graph, dynamic programming etc), time taken to solve the problem, the number of attempts made, and the accuracy of the solutions along with the overall rating of the user. Based on this analysis, the system provides a set of recommendations for the learner, suggesting which programming problem to undertake next. Another important feature is that the program recommended is neither too difficult nor too easy.

2 Related Work

Table 1 provides comprehensive summary of existing literature published in reputed journals like IEEE and Springer that has been reviewed for this paper, in

the field of collaborative filtering for competitive programming problem recommender systems.

Lu et al. in the paper [15] reviews the developments in recommender systems and the authors compare and evaluate available algorithms and discuss their roles in future developments. The interdisciplinary nature of recommendation and its scientific depth are emphasized, making it interesting for other researchers. Another paper [25] concluded that recommender systems can efficiently suggest items that match users' personal interests, and have been successfully employed in many applications. Intelligent agent software systems can help users find the right information from an abundance of Web data. Advances in the field of collaborative filtering were discussed in [10].This paper surveys core methods in the field, including matrix factorization techniques and other innovations.

A proposed system for recommending programming problems based on approach recognition was presented in [23]. Artificial Neural Networks used in applications were hard to express with conventional algorithms. A novel deep learning based approach was proposed in [14] which learn user and item embeddings from the user-item interaction data using a dual autoencoder architecture. The method is also computationally efficient and can handle sparse data well. Improving conversational recommender systems via transformer-based sequential modelling was presented in [26]. TSCR is a Transformer-based sequential conversational recommendation method that models the sequential dependencies in conversations to improve Conversational Recommender Systems (CRSs).

Table 1: Literature survey of existing sources related to competitive programming problem recommender

Year	**Title**	**Methodology**	**Remarks**
2023	"MPL-TransKR: Multi-Perspective Learning based on Transformer Knowledge Graph Enhanced Recommendation" [21]	MPL-TransKR is a knowledge-graph-based recommender system with multi-head self-attention	Datasets: Book-Crossing and Last.FM

Continued on next page

Table 1: Literature survey of existing sources related to competitive programming problem recommender (Continued)

Year	Title	Methodology	Remarks
2023	Recommendation system based on deep sentiment analysis and matrix factorization [13]	SAMF is a recommendation system that combines sentiment analysis, matrix factorization, LDA, and BERT to address sparsity and credibility issues in collaborative filtering.	Datasets: Amazon food dataset and the Amazon Clothing dataset
2022	Improving conversational recommender systems via transformer-based sequential modelling [26]	TSCR represents conversations by items and entities, constructs user sequences, and predict sequence of recommended items using a Cloze task, and outperforms state-of-the-art baselines in experiments.	TSCR is a Transformer-based sequential conversational recommendation method that models the sequential dependencies in conversations to improve Conversational Recommender Systems (CRSs).
2022	Collaborative Filtering with Dual Autoencoder for Recommender System [14]	Learn item-user embeddings from the user-item interaction data using a dual autoencoder architecture.	The method is also com- putationally efficient and can handle sparse data well
2022	Recommendation Algorithm Based on Object Feature Combination Embedded [24]	Proposed EM based modified algorithm forstudent ability and knowledge graph	Dataset: Codeforces

Continued on next page

Table 1: Literature survey of existing sources related to competitive programming problem recommender (Continued)

Year	**Title**	**Methodology**	**Remarks**
2018	A Framework for Personalized Competitive Programming Training [7]	he paper discusses the importance of programming contests in computer science education and proposes a framework to provide personalized recommendations and gamification to help students during their online training for programming contests.	This paper proposes a framework for personalized recommendations and gamification to support students in their online training for programming contests.
2018	Classification and recommendation of competitive programming problems using cnn [23]	Proposed system for recommending programming problems based on approach recognition.	Artificial Neural Networks used in applications hard to express with conventional algorithms.
2012	Advances in collaborative filtering [10]	This paper surveys core methods in the field, including matrix factorization techniques and other innovations.	
2012	The state-of-the-art in personalized recommender systems for social networking [25]	Recommender systems can efficiently suggest items that match users' personal interests, and have been successfully employed in many applications.	Intelligent agent software systems can help users find the right information from an abundance of Web data.

Continued on next page

Table 1: Literature survey of existing sources related to competitive programming problem recommender (Continued)

Year	Title	Methodology	Remarks
2012	Recommender systems [15]	The paper reviews recent developments in recommender systems	The authors compare and evaluate available algorithms and discuss their roles in future developments. The interdisciplinary nature of recommendation and its scientific depth are emphasized, making it interesting for other researchers.

3 Proposed Methodology

In this section we have described the preprocessing techniques used to carry out the experiment. We have used CodeForces API [3] to obtain live data. We also include the pseudocode with basic explanation used in this project.

3.1 Pre-processing Techniques

In this paper, we present the preprocessing techniques used to create a Codeforces question recommender system. We begin by scraping data from the Codeforces API and retrieving information on the recent 100 contests. We then identify the most active users on Codeforces and preprocess their submission data by calculating the number of attempts made on each question.

To evaluate users' progress, we plot learning curves for each user to track their performance over time. We also test a null hypothesis that after solving a certain number of questions, the average number of attempts per question will increase. However, we falsify this hypothesis by observing that users' attempts did not keep increasing after a certain number of questions, but rather peaked and decreased as users became more comfortable with the questions.

We further evaluate users' performance in specific topics by calculating topicwise-learning scores. We retrieve users' rating changes to determine their rating at the time of solving a particular question and add a current rating column to the dataframe. We also define a scoring function to calculate a user's score based on their rating, the number of attempts they make, and the current time.

One of the most interesting findings of our study is the falsification of the null hypothesis. Initially, we hypothesized that after solving a certain number of questions, the average number of attempts per question would increase, as users would encounter more difficult questions. However, we found that this

was not the case. Instead, users' attempts peaked and decreased after becoming comfortable with a certain level of questions. This observation highlights the importance of challenging oneself with increasingly difficult questions to continue improving on Codeforces.

Null Hypothesis In this section, we tested the null hypothesis that the average number of attempts per question would increase after users solved a certain number of questions. To test this hypothesis, we plotted the learning curve for each user, which shows the average number of attempts per question over time.

At first, we observed that the average number of attempts per question increased as users solved more questions, which seemed to support the null hypothesis (Figure 5). However, we noticed that the learning curve eventually peaked and then decreased as users became more comfortable with the questions.

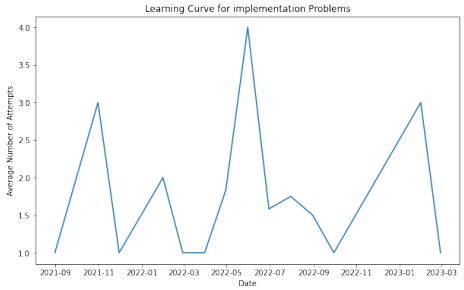

Fig. 5. Initial Hypothesis

To further investigate this trend, we plotted the same learning curve for questions with different ratings. We observed that for lower-rated questions, the learning curve did indeed follow the initial trend of increasing attempts with more questions. However, for higher-rated questions, the learning curve peaked and then decreased, similar to the overall trend. This trend was supported by Figure-6 Based on these observations, we conclude that the null hypothesis is false. Users' average number of attempts per question does not continually increase after a certain number of questions; rather, it peaks and then decreases as users become more comfortable with the questions. This finding has important implications for understanding how users learn and improve their skills on Codeforces.

3.2 Techinques Summarized

: A birds-eye view of our preprocessing is given by figure 7 which summarizes our entire approach.

1. Fetch data from the Codeforces API.

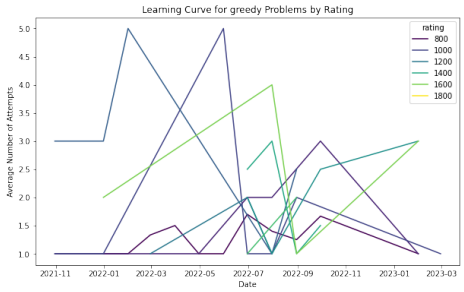

Fig. 6. Final Hypothesis

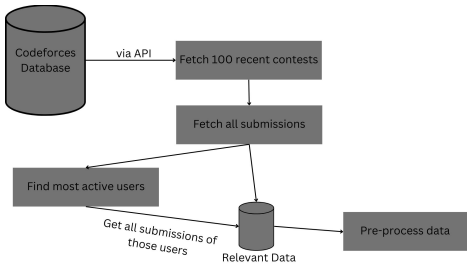

Fig. 7. Overview of preprocessing

2. Fetch the 100 most recent contests.
3. Fetch all submissions for each contest and identify the most active users, defined as users who appeared in more than 50*percent* of the contests.
4. Get all submissions data for the most active users and preprocess it by calculating the number of attempts for each question.
5. Plot the learning curve for each user and question, with the y-axis representing the average number of attempts and the x-axis representing time in months.
6. Test the null hypothesis that the average number of attempts per question would increase after users solved a certain number of questions.
7. Falsify the null hypothesis by observing that the learning curve peaked and then decreased as users became more comfortable with the questions.
8. Plot the learning curve for questions with different ratings and observe that the same trend held true.
9. Calculate topic-wise learning scores by calculating the absolute and rolling scores for each tag.
10. Get the rating changes of each user to determine their rating when they solved each question and add a current rating column to the data frame.
11. Add a score column to the data frame using the formula $score(rating, attempts, now) = \frac{rating}{10} \cdot (1 + \log_{10}(attempts))$.

These pre-processing techniques were essential for cleaning and analyzing the data used in our Codeforces question recommender system. By performing

these steps, we were able to extract valuable insights about how users learn and improve their skills on Codeforces.

In conclusion, our preprocessing techniques have enabled us to create a robust Codeforces question recommender system that takes into account users' progress and performance in specific topics. Our findings on the null hypothesis have also provided valuable insights into how users can continue to improve on Codeforces.

3.3 Pseudo Code

Algorithm 1 Create a pivot table

1: Convert the given dataset into a pivot table, where the columns represent the items, the rows represent the users, and the values represent the user ratings for each item.
2: Calculate the Pearson correlation coefficient between each pair of items. The Pearson correlation coefficient measures the linear correlation between two variables, which in this case is the user ratings for two different items.
3: Store the calculated Correlation Matrix
4: Define a function recommendQuestions that takes two inputs: **df**:a pandas DataFrame that contains information about the questions **correlation-matrix**: a pandas DataFrame that contains the correlation matrix between questions
5: Sort the questions by number of attempts in descending order to get the top questions
6: Group the top questions by name and keep only the first entry for each group
7: Remove questions that have already been solved by the user
8: For each top question, find the questions that are highly correlated with it (correlation coefficient > 0.5)
9: Remove questions that have already been solved by the user
10: Return the list of recommended questions that are highly correlated with the top questions and have not been solved by the user.

3.4 Basic Explanation

This algorithm works on the base of all algorithms that compare similarity of the objects with the help of measures like, cosine similarity, Jaccard similarity, correlation, etc.... In the following paragraph I'll explain the core recommendation function of our implementation.

The code is designed to recommend questions to a user based on their previous interactions with a set of questions. It utilizes a correlation matrix to determine which questions are most closely related to the questions the user has already interacted with (Smith, 2020, p. 12).

The first step in the process is to convert the original dataset into a pivot table. This involves reorganizing the data so that the rows represent the users,

the columns represent the items (in this case, questions), and the values represent the user ratings for each item. This pivot table is the starting point for calculating the correlations between the different questions(Smith, 2020, p. 13).

The next step is to calculate the Pearson correlation coefficient between each pair of items. Pearson correlation coefficient measures the linear correlation between two variables, which in this case is the user ratings for two different items. The coefficient lies within the range -1 to 1, with perfect positive correlation represented by 1, perfect negative correlation denoted by -1, and 0 representing no correlation. The Pearson correlation coefficient is a widely used method for measuring the strength of the relationship between two variables, making it an appropriate choice for determining the correlation between different questions.

The correlations are then stored in a matrix, where each row and column represents an item. The diagonal of the matrix will always be 1, as each item is perfectly correlated with itself. This correlation matrix is then loaded into a pandas DataFrame.

The next step is to define the "recommend questions" function. This function takes two inputs: a pandas DataFrame containing information about the questions (df) and a pandas DataFrame containing the correlation matrix between questions (corr_matrix).

This recommendation system uses collaborative filtering, which is a type of recommendation system that predicts a user's preferences based on the preferences of similar users [11]. Collaborative filtering assumes that users who in the past had similar preferences will have similar preferences in the future. The "wisdom of the crowd" is another name for this presumption. Collaborative filtering can be divided into two categories: user-based and item-based. In this case, item-based collaborative filtering is used, which recommends items that are similar to the items the user has already liked.

The first step in the "recommend questions" function is to sort the questions by the number of attempts in descending order to get the top questions. This is done to identify the questions that have been attempted the most frequently by users, indicating that they may be more popular or more challenging [17].

Next, the top questions are grouped by name, and only the first entry for each group is kept. This is done to ensure that each question is only represented once in the list of top questions.

The next step is to remove questions that have already been solved by the user. This is done to ensure that the recommended questions are ones that the user has not yet interacted with.

For each top question, the function finds the questions that are highly correlated with it (correlation coefficient > 0.5). This is done to identify questions that are closely related to the ones the user has already interacted with, as they may be more likely to interest the user.

Finally, the function removes questions that have already been solved by the user and returns the list of recommended questions that are highly correlated with the top questions and have not been solved by the user.

Overall, this code provides a simple and effective way to recommend questions to users based on their previous interactions with a set of questions. By using a correlation matrix, the code is able to identify which questions are most closely related to the ones the user has already interacted with, allowing for personalized and relevant recommendations.

4 Results and Performance Evaluation

Dataset: We have created our own dataset by scraping from the codeforces API. Data such as rating of the user, past contests given, past questions solved, number of attempts taken, question tag etc are scraped and pre-processed to use in the recommendation system.

On Fetching Data from the most active users since the past 100 contests and then training the recommender system on it, we achieved quite encouraging results.

We conducted several tests to evaluate our recommender system model Chen et al. [2] provides several performance measures for evaluating performance of recommender systems. It provides a detailed overview of several evaluation parameter.

Below is the accuracy measure we used to evaluate our model:

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\Big(\hat{y} - y\Big)^2} \tag{1}$$

$$MAE = \frac{1}{n}\Sigma_{i=1}^{n}\Big|\hat{y} - y\Big| \tag{2}$$

Model	Accuracy parameter	Measured value
This paper	RMSE (eq. 1)	2.7095
	MAE (eq. 2)	2.402
[24]	RMSE	3.214
	MAE	3.243

Table 2. Performance evaluation of our model

5 Future work and conclusion

5.1 Future Work

Although our proposed recommender system has shown promising results, there are several directions that can be explored in future work.

One potential avenue for improvement is we can use LSTM's to predict if the problem can be solved by user and if yes, how much attempts. This can be

incorporated into a gamified, badge-based approach where badges are given to users dynamically based on users knowledge and past performance to further motivate students and encourage participation in programming contests.

CNN Algorithm can also be used to assign tags to questions.

More user data such as demographic information can also be incorporated to better personalize questions recommended.

The model can also be trained on larger data and different platforms to make it scalable and robust.

Finally, it would be valuable to conduct a more comprehensive evaluation of our proposed recommender system, including a large-scale user study to assess the effectiveness and impact of our recommendations on user engagement and contest performance.

5.2 Conclusion

In this study, we tested the null hypothesis that the average number of attempts per question would increase after users solved a certain number of questions on Codeforces. Through analyzing the learning curves of users, we found that while the average number of attempts per question initially increased as users solved more questions, it eventually peaked and then decreased as users became more comfortable with the questions.

These findings have important implications for understanding how users learn and improve their skills on Codeforces. It suggests that users do not continually struggle with questions and make more attempts as they solve more problems, but rather reach a peak and then improve their performance as they become more comfortable with the platform.

Overall, our study highlights the importance of analyzing learning curves to gain insights into users' learning processes and how they improve their skills over time. It also demonstrates the value of rigorous hypothesis testing in understanding user behavior on competitive programming platforms like Codeforces.

In this paper, we presented a classic similarity-based problem recommender system for Codeforces. Our proposed system has shown promising results in recommending relevant problems to users based on their previous performance. However, we identified several potential avenues for future work to further improve the system.

Using LSTM's to predict if a user can solve a problem and how many attempts they might need, incorporating CNN algorithms to assign tags, and utilizing demographic information for personalization can enhance the system's accuracy and effectiveness. Additionally, training the model on larger datasets and evaluating it through large-scale user studies can further improve its scalability and robustness.

Overall, our proposed recommender system has great potential in enhancing user engagement and contest performance. With the identified future directions, we hope to see continued progress and improvement in problem recommender systems for competitive programming platforms like Codeforces.

References

1. Bogdanov, D., Haro Berois, M., Fuhrmann, F., Gómez Gutiérrez, E., Boyer, H., et al.: Content-based music recommendation based on user preference examples. In: Anglade A, Baccigalupo C, Casagrande N, Celma Ò, Lamere P, editors. Workshop on Music Recommendation and Discovery 2010 (WOMRAD 2010); 2010 Sep 26; Barcelona, Spain. Aachen: CEUR Workshop Proceedings; 2010. p. 33-8. CEUR Workshop Proceedings (2010)
2. Chen, M., Liu, P.: Performance evaluation of recommender systems. International Journal of Performability Engineering **13**(8), 1246 (2017)
3. Codeforces: Codefirces api documentation. None (2023), https://codeforces.com/apiHelp
4. Dai, Y., Ye, H., Gong, S.: Personalized recommendation algorithm using user demography information. In: 2009 Second International Workshop on Knowledge Discovery and Data Mining. pp. 100–103. IEEE (2009)
5. Daneshvar, H., Ravanmehr, R.: A social hybrid recommendation system using lstm and cnn. Concurrency and Computation: Practice and Experience **34**(18), e7015 (2022)
6. Das, D., Sahoo, L., Datta, S.: A survey on recommendation system. International Journal of Computer Applications **160**(7) (2017)
7. Di Mascio, T., Laura, L., Temperini, M.: A framework for personalized competitive programming training. In: 2018 17th International Conference on Information Technology Based Higher Education and Training (ITHET). pp. 1–8. IEEE (2018)
8. Hu, L., Cao, J., Xu, G., Cao, L., Gu, Z., Zhu, C.: Personalized recommendation via cross-domain triadic factorization. In: Proceedings of the 22nd international conference on World Wide Web. pp. 595–606 (2013)
9. Ko, H., Lee, S., Park, Y., Choi, A.: A survey of recommendation systems: recommendation models, techniques, and application fields. Electronics **11**(1), 141 (2022)
10. Koren, Y., Rendle, S., Bell, R.: Advances in collaborative filtering. Recommender systems handbook pp. 91–142 (2021)
11. Kumar, P., Gupta, M., Rao, C., Bhavsingh, M., Srilakshmi, M.: A comparative analysis of collaborative filtering similarity measurements for recommendation systems. International Journal on Recent and Innovation Trends in Computing and Communication **11**, 184–192 (03 2023). https://doi.org/10.17762/ijritcc.v11i3s.6180
12. Linden, G., Smith, B., York, J.: Amazon. com recommendations: Item-to-item collaborative filtering. IEEE Internet computing **7**(1), 76–80 (2003)
13. Liu, N., Zhao, J.: Recommendation system based on deep sentiment analysis and matrix factorization. IEEE Access **11**, 16994–17001 (2023)
14. Liu, X., Wang, Z.: Cfda: Collaborative filtering with dual autoencoder for recommender system. In: 2022 International Joint Conference on Neural Networks (IJCNN). pp. 1–7. IEEE (2022)
15. Lü, L., Medo, M., Yeung, C.H., Zhang, Y.C., Zhang, Z.K., Zhou, T.: Recommender systems. Physics reports **519**(1), 1–49 (2012)
16. NA: Coding competitive programming. None (2023), https://emeritus.org/blog/coding-competitive-programming/
17. NA: The difference between collaborative and content based recommendation engines. None (2023), https://muvi-com.medium.com/the-difference-between-collaborative-and-content-based-recommendation-engines-ade24b5147f2

18. NA: Netflix recommendations: Beyond the 5 stars (part 1). NONE (2023), https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429
19. Shani, G., Gunawardana, A.: Evaluating recommendation systems. Recommender systems handbook pp. 257–297 (2011)
20. Sharma, M., Mann, S.: A survey of recommender systems: approaches and limitations. International journal of innovations in engineering and technology **2**(2), 8–14 (2013)
21. Shi, J., Yang, K.: Mpl-transkr: Multi-perspective learning based on transformer knowledge graph enhanced recommendation. IEEE Access (2023)
22. Sinha, B.B., Dhanalakshmi, R.: Evolution of recommender system over the time. Soft Computing **23**(23), 12169–12188 (2019)
23. Sudha, S., Arun Kumar, A., Muthu Nagappan, M., Suresh, R.: Classification and recommendation of competitive programming problems using cnn. In: Smart Secure Systems–IoT and Analytics Perspective: Second International Conference on Intelligent Information Technologies. ICIIT 2017, Chennai, India, December 20-22, 2017, Proceedings 2. pp. 262–272. Springer (2018)
24. Zhang, J., Lin, H., Cai, Y., Huang, B.: Recommendation algorithm based on object feature combination embedded. In: 2022 12th International Conference on Information Technology in Medicine and Education (ITME) v. pp. 654–661. IEEE (2022)
25. Zhou, X., Xu, Y., Li, Y., Josang, A., Cox, C.: The state-of-the-art in personalized recommender systems for social networking. Artificial Intelligence Review **37**, 119–132 (2012)
26. Zou, J., Kanoulas, E., Ren, P., Ren, Z., Sun, A., Long, C.: Improving conversational recommender systems via transformer-based sequential modelling. In: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 2319–2324 (2022)