# Sequence Based Personalised Problem Recommendations For Competitive Programming

Brij Bidhin Desai
IIIT Bangalore
India
BrijBidhin.Desai@iiitb.ac.in

Madhav Sood
IIIT Bangalore
India
Madhav.Sood@iiitb.ac.in

Yukta Rajapur
IIIT Bangalore
India
Yukta.Rajapur@iiitb.ac.in

Shlok Agrawal
IIIT Bangalore
India
Shlok.Agrawal@iiitb.ac.in

## ABSTRACT

Practicing programming problems on online automated judges is crucial for competitive programmers as it fosters the critical thinking necessary for tackling problems in programming contests. Competitive programmers often spend significant time searching for problems on these platforms that match their skill level, which can be inefficient and cause them to overlook available problems. In this paper, we propose a novel approach of providing personalised problem recommendations using Long Short-Term Memory (LSTM).

*Link to our work:* https://github.com/brij-desaii/Codeforces-Problem-Recommender

## KEYWORDS

Competitive Programming, Recommendation Systems, LSTM, NDCG, Seq2Seq, contests, ratings, Codeforces, API

**ACM Reference Format:**
Brij Bidhin Desai, Madhav Sood, Yukta Rajapur, and Shlok Agrawal. 2018. Sequence Based Personalised Problem Recommendations For Competitive Programming. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX



## 1 INTRODUCTION

Programming contests are competitive events held over the internet or local networks where participants write computer programs based on provided problem specifications. Our focus is on short-term contests with clearly defined tasks, such as the International Collegiate Programming Contest (ICPC), the International Olympiad in Informatics (IOI), and company-sponsored events like Google CodeJam and Facebook Hacker Cup. We also consider regular contests hosted on platforms like CodeForces, CodeChef, and the Algorithm track at TopCoder.

Most of these platforms employ an Elo-based rating system as a method to assign and update numerical ratings to participants based on their performance in contests. This rating system is dynamic and continuously adjusts participant ratings based on their recent contest performances, aiming to reflect their current skill level relative to other participants. As a result, users constantly try to excel in contests and improve their rating.

To excel in these contests and improve ones rating, practicing on online automated judges is essential. These judges are online systems that compile and execute code, testing it against predefined data and constraints such as time and memory limits. The system captures the output of submitted code, compares it to expected results, and returns the outcome. Popular online judges include Codeforces, UVa Online Judge, TopCoder, CodeChef, SPOJ, HackerRank, HackerEarth, Light OJ, Kattis, and URI Online Judge.

Currently, there is a vast array of programming problems available on numerous automated online judges across the internet. These problems cover various topics such as Ad Hoc, Mathematics, Data Structures, Dynamic Programming, Graphs, Geometry, and more. They are further classified into different difficulty levels, with problems within the same topic varying in complexity.

Despite the abundance of problems, effectively navigating this extensive collection can be a challenge for competitive programmers. Many online judging platforms lack a robust organization system, hindering the ability to efficiently locate problems that align with specific learning objectives. To address this issue and enhance the problem-solving experience for competitive programmers, we propose a recommendation system for programming problems.

## 2 RELATED WORK

Caro-Martinez and Jimenez-Diaz [1] utilize a user-based methodology by graphically representing users and applying various similarity functions. They focus on retaining only the most recent solution from each user for a given problem. Their findings emphasize the importance of selecting the most effective similarity metric to achieve optimal outcomes. Furthermore, they note that user-based techniques yield superior performance when using unweighted metrics.


*Conference acronym 'XX, June 03–05, 2018, Woodstock, NY*

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

Yera and Martínez [4] apply a collaborative filtering approach to recommendation systems, involving three primary steps: 1) Constructing an extended matrix of user problems, 2) Preprocessing this matrix to handle natural noise, and 3) Generating problem recommendations. Experimental findings indicate that steps 1) and 2) significantly enhance the accuracy of neighborhood formation, thereby improving recommendation precision. They evaluate their method using a dataset from an online judge platform. To determine the optimal value K (number of neighbors), they tested various K values and found that precision stabilizes K > 130, with increasing K values further improving performance within a certain range.

Pereira et al. [3] employed an online judge (CodeBench) from the Federal University of Amazonas in Brazil to assess problem-solving effort based on metrics such as average attempts, code lines, variables, algorithmic complexity, and other indicators per problem. They used cosine similarity as the distance metric for nearest neighbor analysis to gauge similarity between recommended and target problems. Utilizing the qualitative Kappa Cohen test, they achieved a commendable result of 0.83. Furthermore, employing Bonferroni's correction statistical test revealed that their model maximizes positive emotional states while minimizing frustration.

## 3 PROPOSED APPROACH

In contrast to the above works and traditional feedback-driven recommendation systems such as those used in e-commerce and content platforms, we believe that recommending programming problems based solely on user preferences is not suitable for fostering growth in competitive programming. The goal is not simply to cater to user likes, but rather to strategically challenge users with problems that expand their skill set and encourage learning.

To address this challenge, our approach focuses on a subset of proficient users termed "super users." These individuals have demonstrated significant improvement in their contest rankings within a short time frame. We hypothesize that the problem-solving strategies and sequences of practiced problems by these super users directly contribute to their success by enhancing critical thinking and imparting essential programming concepts.

Our motivation for this strategy is underscored by the popularity of platforms like A20J and C2 ladders, which demonstrate the efficacy of practicing the problems that are practiced by successful users.

We leverage this hypothesis by training a Seq2Seq LSTM (Long Short-Term Memory) model using timestamp-ordered sequences of problems solved by super users, along with their corresponding user profiles. These user profiles include the distribution of topics they have practiced and the distribution of problem difficulties they have tackled. The model is designed to predict the subsequent sequence of problems based on this comprehensive historical problem-solving data, enriched with user-specific learning patterns and preferences.

In the subsequent sections of this paper, we concentrate on the implementation and evaluation of our approach within the context of Codeforces. Codeforces is chosen for its widespread usage and accessible API, providing an ideal platform for applying and validating our recommendation system.

## 4 IDENTIFYING THE SUPER USERS

A user is considered as a super user if they satisfy the following two conditions:

- If the user was successful in increasing their rank within a predefined threshold time. This picks users who displayed an improvement in their problem solving skills with practice.
- If the user has practiced more than a predefined number of problems. This is done to filter out the users whose rank increase is not because of their practice, e.g alt accounts.

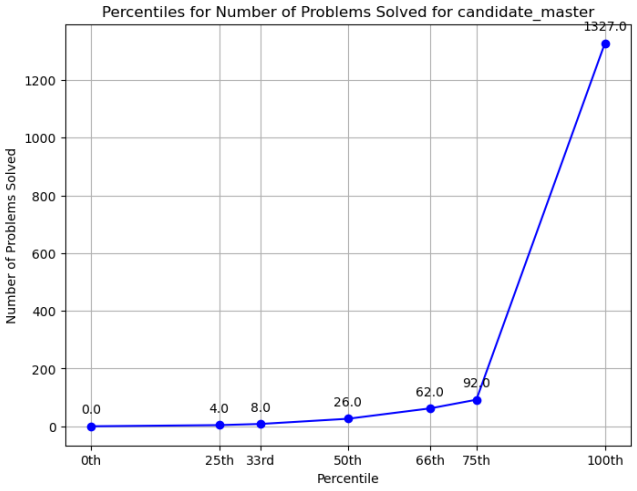


**Figure 1: Count of practise problems solved vs the percentile.**

### 4.1 Filtering Alt Accounts

To distinguish genuine users from alt accounts, we analyze user activity data from a Kaggle dataset of Codeforces IDs (reference dataset). Our primary measure for alt accounts is the practice activity outside of contests. We establish a threshold for the number of practice problems solved during the transition period between two consecutive ranks. For example, to filter out alt accounts between the ranks of Candidate Master and Master, we set a threshold of 62 practice problems. This threshold places users at the 66th percentile among those who reached the Master rank, effectively filtering out 66% of accounts, as can been seen in Figure 1.

### 4.2 Identifying Rapid Rank Improvements

To identify users who exhibit rapid skill improvement, we analyze the time taken to transition between ranks. We plot the distribution of transition times for various rank changes and select a maximum time threshold that balances speed with statistical significance. For instance, for the transition from Candidate Master to Master, we set a maximum threshold of 147 days. This threshold places users at the 50th percentile, ensuring a sufficient number of super users while maintaining a reasonable timeframe for rank improvement. This is illustrated in Figue 2.

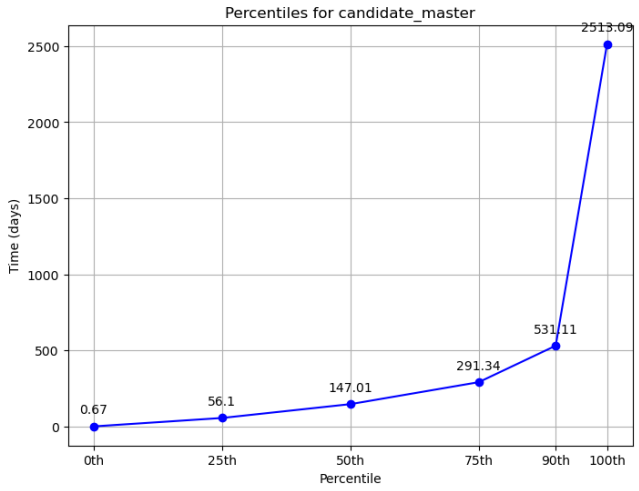


**Figure 2: Time taken vs the percentile.**

### 4.3 Results

Our methodology yielded 116 super users within the specified rank transition range, meeting both criteria of rapid rank improvement and high practice activity. These super users represent a subset of the community that has demonstrated exceptional dedication and skill in competitive programming in the given rating threshold.

## 5 METHODOLOGY

### 5.1 Dataset Curation

After identifying the super users, we developed a script that utilizes the Codeforces API to extract the super users of each rating bucket. We created the user-problem interaction dataset as follows: For each identified super user, we retrieved a detailed list of problems they practiced within the specified time period along with the timestamps for each problem solved.

Furthermore, we gathered additional super user data that includes the distribution of problem difficulty levels solved by each super user. Additionally, we collected information on the distribution of topics tackled by super users, encompassing categories such as ad-hoc, mathematics, binary search, and more. This comprehensive data captures detailed user profiles, enriching our dataset for analysis and model training.

### 5.2 Preprocessing

After assembling a dataset comprising user-problem interaction histories, along with distributions of topics and problem difficulties per user, we conducted essential preprocessing steps to prepare the data for model ingestion.

Initially, we performed basic data cleaning procedures, including the removal of null values to ensure data integrity. Subsequently, we employed PyTorch to create user and problem vocabularies, facilitating the conversion of unique user and problem identifiers into numerical embeddings suitable for model input. Both the user and problem embedding dimensions were set to 128.

Next, for each user, we organised the raw user-problem interactions into timestamp sorted sequences of length $k$ , where $k$ is a hyperparameter.

Following the sequence generation, we randomly partitioned 85% of these sequences for training purposes, reserving the remaining 15% for validation.

### 5.3 Model Architecture

Long Short-Term Memory (LSTM) [2] is a specialized recurrent neural network architecture designed to handle sequential data with long-range dependencies. Developed by Hochreiter and Schmidhuber in 1997, LSTMs use gated mechanisms to selectively retain or forget information over time, overcoming issues like the vanishing gradient problem in traditional RNNs. In this paper, we apply LSTM networks to sequence modeling for personalized problem recommendations in competitive programming, leveraging their capability to capture complex temporal patterns efficiently.

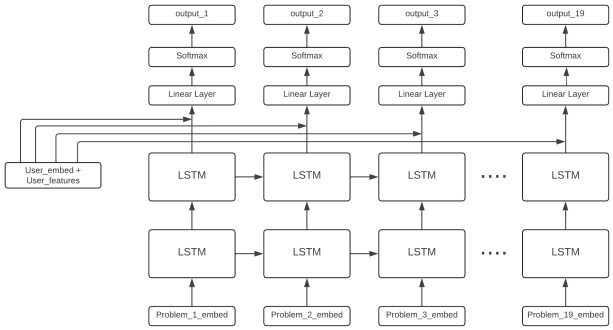


**Figure 3: Diagram of the model architecture.**

Figure 3 illustrates the model architecture for $k = 20$. The model comprises two LSTM layers, with the initial layer accepting a sequence of problem embeddings of length $k$ as input. The output from these LSTM layers yields another sequence of intermediate problem embeddings, each of size 128. These intermediate embeddings are concatenated with the user identifier embedding and user features (including topics and problem difficulty distributions). This combined representation is then passed through a linear layer, followed by a softmax function to generate a probability distribution over all possible problems in the dataset. This probability distribution indicates the likelihood of each problem being the next one attempted by the user. The loss function employed is Cross Entropy, where the ground truth for each output block corresponds to the subsequent problem in the input sequence. For instance, the ground truth for output_1 is problem_2_embed, and so forth.

### 5.4 Model Training

The model was trained using the Stochastic Gradient Descent optimiser for 30 epochs but with early stopping it completed training at 20 epochs. Figure 4 illustrates the validation loss vs epochs graph.

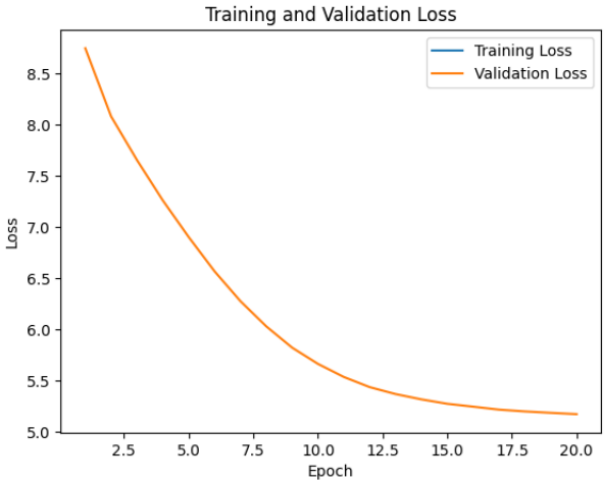


**Figure 4: Validation Loss Vs Epochs.**

### 5.5 Generating Recommendations

To provide personalized problem recommendations for a target user on Codeforces, we initiated the process by inputting the user's Codeforces handle to retrieve metadata, including the problems they have solved. This metadata was then processed using our developed method, which accepts user-specific data such as a user ID and a sequence of solved problems, to generate recommendations for the next problem the user should attempt.

We preprocess the user ID and problem sequence by tokenizing and numerically encoding them, converting them into tensors suitable for input to a neural network model. Similarly, additional features associated with the user or problem sequence are processed and stored in a dictionary.

Subsequently, we pass these encoded tensors through the neural network model to obtain predictions for the next problem. These predictions are probabilities indicating the likelihood of each problem being the subsequent challenge for the user.

We select the top-k predicted problems, filtering out those that the user has already solved. Finally, we output a list of recommended problems for the user to tackle next, leveraging this approach to enhance problem-solving experiences in competitive programming contexts.

## 6 EVALUATION AND RESULTS

### 6.1 Metric

We used the Normalised Discount Cumulative Gain as a metric of evaluating the performance of our model.

The Normalized Discounted Cumulative Gain (NDCG) is a metric commonly used to evaluate the effectiveness of ranked retrieval systems, such as recommendation algorithms or search engines. It measures the quality of a ranked list of items by considering both the relevance of each item and its position in the list.

The NDCG score is computed based on the relevance labels of items and the ranking order produced by a system. It takes into account the graded relevance of items, where higher relevance corresponds to a higher score.

The formula for computing DCG (Discounted Cumulative Gain) at a given rank $k$ is as follows:

$$\mathrm{DCG}_k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

Here, $rel_i$ denotes the relevance of the item at rank $i$ in the ranked list.

The ideal DCG (IDCG) is calculated by sorting the relevance scores in descending order:

$$\mathrm{IDCG}_k = \sum_{i=1}^{\min(k,n)} \frac{2^{rel_{(i)}} - 1}{\log_2(i+1)}$$

where $n$ is the total number of items.

Finally, the NDCG score at rank $k$ is computed as:

$$\mathrm{NDCG}_k = \frac{\mathrm{DCG}_k}{\mathrm{IDCG}_k}$$

The NDCG score ranges from 0 to 1, where a higher score indicates a better quality of the ranked list in terms of relevance and order.

### 6.2 Comparison with baseline

To demonstrate the efficacy of our model we compared it with a naive baseline model that recommends the most popular problems based on the frequency of users that solved them.

We obtained the most popular problems by taking the top k most frequently solved problems and compared it to our top k recommended problems from the validation data.

We used the sci-kit-learn library's `ndcg_score function` to assess the performance of the recommended problems from the model and the popular ones as a baseline. It constructs a representative array to simulate recommendation scores and iterates over different top-k values to compute the NDCG score for each system.

(Table 1) displays the NDCG@3, NDCG@5 and NDCG@10 values of our model along with the baseline. Overall we saw an average of 1251.9% improvement over the baseline model.

## 7 CONCLUSION

In this paper, we proposed a novel approach for personalized problem recommendations in competitive programming using Long Short-Term Memory (LSTM) models. Our methodology focused on leveraging the problem-solving strategies of super users to train LSTM models capable of predicting the next sequence of problems for individual users. We addressed the limitations of traditional recommendation systems by emphasizing the importance of strategically challenging users with diverse problems to foster skill expansion and learning.

The key contributions of our work include:

- Identifying super users based on significant contest ranking improvements within a short time frame.
- Curating a comprehensive dataset comprising user-problem interactions, problem difficulty distributions, and topic distributions.

**Table 1: NDCG Metric Results**

| k | Model | Baseline | Percentage Improvement |
|---|---|---|---|
| 3 | 0.0168 | 0.0012 | 1300% |
| 5 | 0.0225 | 0.0016 | 1312.5% |
| 10 | 0.0298 | 0.0023 | 1191.3% |

- Developing an LSTM-based Seq2Seq recommendation model that incorporates user-specific learning patterns and preferences.
- Demonstrating significant improvements over a naive baseline model through rigorous evaluation using the Normalized Discounted Cumulative Gain (NDCG) metric.

Our experimental results showed an average improvement of 1251.9% in NDCG scores compared to the baseline, highlighting the effectiveness of our personalized recommendation approach. By capturing temporal patterns in problem-solving sequences and user profiles, our LSTM-based model successfully generated tailored problem recommendations aligned with users' skill levels and learning objectives.

## 8 FUTURE SCOPE

The future scope of this project holds promise in real-world applications, which include integration as a user-friendly browser extension or a standalone tool. This would empower users to seamlessly access customized problem recommendations based on their skill level and knowledge, enhancing their competitive programming skills. Also, incorporating additional user features and problem attributes such as leveraging problem tags as embeddings for each problem is a potential option for improving recommendation accuracy thereby further optimizing the learning journey for users.



## ACKNOWLEDGMENTS

We extend our gratitude to Professor Raghuram Bharadwaj for his exceptional teaching style and support throughout the course AI 705 Recommendation Systems, which has greatly contributed to the success of our project.

## REFERENCES

[1] Marta Caro-Martinez and Guillermo Jimenez-Diaz. 2017. Similar Users or Similar Items? Comparing Similarity-Based Approaches for Recommender Systems in Online Judges. In *Case-Based Reasoning Research and Development*, David W. Aha and Jean Lieber (Eds.). Springer International Publishing, Cham, 92–107.

[2] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[3] Felipe D. Pereira, Hermino B.F. Junior, Luiz Rodriquez, Armando Toda, Elaine H.T. Oliveira, Alexandra I. Cristea, David B.F. Oliveira, Leandro S.G. Carvalho, Samuel C. Fonseca, Ahmed Alamri, and Seiji Isotani. 2024. A Recommender System Based on Effort: Towards Minimising Negative Affects and Maximising Achievement in CS1 Learning. , 466-480 pages. https://doi.org/10.1007/978-3-030-80421-3_51 EPrint Processing Status: Full text deposited in DRO.

[4] Raciel Yera and Luis Martínez. 2017. A recommendation approach for programming online judges supported by data preprocessing techniques. *Applied Intelligence* 47, 2 (01 Sep 2017), 277–290. https://doi.org/10.1007/s10489-016-0892-x