A Cloud Based Recommender System for Competitive Programming Platforms with Machine and Deep Learning

Wilson Julca-Mejia[1], **Herminio Paucar-Curasma**[1,2]

[1]National University of San Marcos (UNMSM)
Lima, Peru

[2]University of São Paulo (USP)
São Paulo, Brazil.

wilson.julca@unmsm.edu.pe, herminiopaucar@usp.br

Abstract. *Nowadays, online judges are very important to improve programming skills for education and technology companies. For this reason, there are many online judges that include large sets of programming challenges. This creates an information overload problem that affects students due to their lack of expertise in choosing the correct challenge to solve, resulting in frustration and a loss of interest in this topic. To solve this scenario, recommender systems appear, but programming judges have not delved much into it. Consequently, this research aims to evaluate the performance of six selected collaborative filtering techniques via a cloud-based software architecture. To validate our experiments we used real online programming judges like CodeChef and NinjaCoding using cloud based architecture with Amazon Web Services, evaluated through Friedman and Wilcoxon statistical tests. The results indicated that Singular Value Decomposition is the best model evaluated with RMSE metric and the fastest in execution time with big datasets.*

Resumo. *Hoje em dia, os juízes online são muito importantes para melhorar as habilidades de programação para empresas de educação e tecnologia. Por esse motivo, existem muitos juízes online que incluem grandes conjuntos de desafios de programação. Isso cria um problema de sobrecarga de informações que afeta os alunos devido à falta de experiência em escolher o desafio correto para resolver, resultando em frustração e perda de interesse por esse tópico. Para resolver esse cenário, surgiram os sistemas de recomendação, mas os juízes de programação não se aprofundaram muito nisso. Consequentemente, esta pesquisa visa avaliar o desempenho de seis técnicas de filtragem colaborativa selecionadas por meio de uma arquitetura de software baseada em nuvem. Para validar nossos experimentos, usamos juízes de programação online reais como CodeChef e NinjaCoding usando arquitetura baseada em nuvem com Amazon Web Services, avaliados por meio de testes estatísticos de Friedman e Wilcoxon. Os resultados indicaram que a Singular Value Decomposition é o melhor modelo avaliado com a métrica RMSE e o mais rápido em tempo de execução com grandes conjuntos de dados.*

Keywords: *Programming Online Judges, Recommender Systems, Collaborative Filtering, Machine Learning, Deep Learning*

1. Introduction

Programming Online Judges (POJs) have great importance in the industry and academia. In the industry, it allows improved skills such as logical reasoning, data structure management and the use of different algorithm paradigms for various real life problems. All these topics are an important part of interviews in Big Tech companies such as Amazon, Google, Meta and others [McDowell, 2015]. In the education sector, it has been proven that the use of POJs in Computer Science courses considerably improve student performance because it allows students to practice programming and receive instant feedback on their solutions. This helps them to learn from their mistakes, identify their weaknesses, and improve their programming skills [Wu et al., 2016]. Also, POJs can be a motivating factor for students, as it allows them to compete with their peers and see their progress in real-time. Moreover, students can earn recognition and rewards for their achievements, which can boost their confidence and encourage them to learn more.

Currently, there is a considerable number of POJs with a large number of challenges [Cruz et al., 2022, Rahman et al., 2022], which have little clarity about the difficulties (easy, medium, hard) and categories (math, graph, strings, etc) [Rahman et al., 2021]. Some examples of these problems can be seen in the POJ UVA which has more than 2 thousand challenges, and in the SPOJ judge which has more than 6 thousand programming problems [Fantozzi and Laura, 2020]. This overload of information generates frustration and abandonment in students if they do not have someone with experience to guide them [Pereira et al., 2021, Fantozzi and Laura, 2020, Yera and Martínez, 2017]. In POJs, the classical way to recommend the next challenge is by doing the most solved challenges [Pereira et al., 2021], this approach lacks customization because it will suggest the same challenges to all users, regardless of the problems they have solved [Caro-Martinez and Jimenez-Diaz, 2017].

One of the most effective and common strategies for automatically recommending is by using collaborative filtering (CF) techniques [Aljunid and Dh, 2020, Bobadilla et al., 2020]. These techniques are classified into memory-based CF and model-based CF. In memory-based approaches, recommendations are generated using the preferred information associated with users. On the other hand, model-based approaches focus on discovering intermediate knowledge, such as the rules of the association, patterns, and other ways of knowledge representation, to build a predictive model that is then used to generate final recommendations [Bhalse and Thakur, 2021, Kluver et al., 2018, Ortega et al., 2016].

A strong and scalable recommender system that is specifically designed for POJ has only seldom been the subject of investigations. When creating such systems, it is important to take into account the distinctive qualities of POJs, such as the enormous quantity of tasks, the variety of difficulty levels and categories, and the rise in online users. Although recommender systems have been thoroughly studied in many different fields, little has been done to apply these methods to the unique difficulties faced by POJs. It is necessary to take into account strategies that are appropriate for addressing the scope and complexity of these systems in order to construct an efficient recommender system for POJs. For these reasons, in this research we have three main objectives. The first objective is to propose a cloud architecture that enables us to couple a recommender system to programming judges using REST APIs. This allows us to take advantage of

cloud computing resources and ensure scalability and availability of the system. The second is to identify machine and deep learning models that provide significantly better recommendations, which are validated through statistical tests such as the Friedman test and Post-hoc tests. Our third is to measure the time of execution of these models, which helps us evaluate their efficiency and select the most suitable model for large datasets and real time applications. By achieving these aims, we provide a comprehensive solution that can help improve the user experience on the Programming Online Judges platform.

This paper's remaining sections are organized as follows: Section II describes the background, III the methodology, IV experimentation and results, and finally V conclusions and future works.

2. Background

2.1. Programming Online Judges

Programming Online Judges are web platforms that have a list of problems. These problems are solved by users sending their solutions (source code) developed in a programming language supported by the platform. The submitted source code is automatically evaluated and the verdict is given back to the user. The POJs were mainly inspired by the most prestigious contest in the world, known as ACM-ICPC [1], where different teams from around the world compete in-person. This contest lasts 5 hours and the participants try to solve between 10 to 13 programming problems, the winners being the ones who solve the most in the shortest time. POJs these days is a very effective tool for learning programming. Many of them have been established for the purpose of online learning and competitions [Intisar et al., 2019]. For example, globally, we have POJs like Peking University, Valladolid University, Timus, and Saratov State University [Yera and Martínez, 2017]. Furthermore, in Latin America there are the Caribbean Online Judge, Beecrowd, OmegaUp and finally in Peru NinjaCoding and Huahcoding are used in different universities and events [Julca-Mejia et al., 2018].

2.2. Collaborative Recommender Systems in POJ

Recommender systems (RS) provide a collection of recommended elements to alleviate the search process in an overloaded environment. In the POJ scenario, the recommender systems that have been developed are based on content and user-item interactions. In this research we focus on this matrix of interactions where if a programming challenge has been solved or not, a rating is assigned and a history of each user and item is generated, allowing us to carry out a collaborative recommendation system. The most relevant investigations found in our review of the literature are:

Caro-Martinez and Jimenez-Diaz [2017] implement a user-based approach by representing them in graphs, using different similarity functions, and eliminating all user submissions to a problem, except the one last resolved. The results indicate that the selection of the highest-performing similarity metric is crucial to achieving the best results and that user-based approaches perform better with unweighted metrics.

Yera and Martínez [2017] use a collaborative filtering recommendation approach, which is composed of three main steps: 1) The construction of the extended matrix of user

[1] https://icpc.global/

problems, 2) the preprocessing of the extended matrix of user problems for managing natural noise, and 3) the recommendation of the problems. Experimental results show that steps 1) and 2) guarantee the formation of a more accurate neighborhood positively impacting accuracy. For the evaluation of their proposal, they use a dataset from the POJ Caribbean Online Judge. For the authors to obtain the value K of neighbours, they tested different values of K where these results conclude that the precision stabilizes for $K > 130$ and tends to increase when K increases in the range $[90, 130]$.

Pereira et al. [2021] use a POJ called CodeBench from the Federal University of Amazonas in Brazil to measure the effort it took to solve a problem using the average number of attempts, the average number of code lines, average number of variables, algorithmic complexity, number of attempts and other indicators for each problem. Using cosine similarity as the distance metric, nearest neighbor analysis is used to calculate the degree of similarity between the recommended problem and the target problem. With the qualitative Kappa Cohen test, they obtained a result of 0.83, which is considered a good result. Then, through the statistical test of Bonferroni's correction, it is shown that the proposed model maximizes the positive emotional state, while minimizing frustration.

Lara-Cabrera et al. [2020] use a recursive approach to matrix factorization and deep learning. Its objective is to improve the quality of the recommendations made to the user using successive training with a recursive matrix factorization approach and deep learning. The evaluation of the model is carried out through the MAE and Precision metrics, giving better results than models such as the Probabilistic Matrix Factorization, Non-Negative Matrix Factorization and SVD++. The author concludes by commenting that his model breaks the trend of using deep learning with neural networks and applying it to matrix factorization.

3. Methodology

3.1. Materials and Methods

The datasets used in this research are CodeChef [2] and NinjaCoding [Julca-Mejia et al., 2018]. Codechef contains data between the years 2009 and 2016 with 565,027 user-item interactions, of which there are 59,322 users and 541 unique problems, and Ninjacoding contains 1,145 user-item interactions.

The methodology we used is CRISP-DM, which provides phases and establishes a set of tasks and activities for each phase for data science projects[Martínez-Plumed et al., 2019].

The experiments were initially carried out on the Google Colaboratory platform with Python 3 through Jupyter notebooks. Then the Colab notebook was transferred to a AWS SageMaker notebook to be able to make it available through Amazon AWS components, so that the architecture is robust and in a productive environment.

3.2. Recommender System

We tested 6 models such as KNN based in Cosine, Pearson, Mean Squared Difference (MSD) similarity functions and Probabilistic Matrix Factorization (SVD), Not Negative

[2]https://www.codechef.com/

Matrix Factorization (NMF) and Deep Matrix Factorization (DeepMF) evaluated through the RMSE metric.

The mentioned models work with the history of user-item interactions. These interactions are represented by a matrix M, where each cell $M[u, i]$ represents the evaluation result of the programming judge (AC, WA, TLE, CE). Inspired by [Toledo and Mota, 2014] we are modeled with the value 1 if the programming challenge was not solved or tried and 2 if the programming challenge was ever solved.

3.3. Architecture

Our proposed architecture was developed by applying the Architecture Centric Design Method (ACDM), because any change in requirements is mitigated with the early stage of rapid prototyping adoption. Therefore, it can be used to establish high fidelity estimates and track the progress of any project construction [Devadiga, 2017].

The POJ and the recommender system communicate through REST APIS, where the $/interactions$ API provides the recommendation system with the list of verdicts in the POJ and the $recommendations/ < idUser >$ API recommends the suggested items for the $idUser$ to be with the active session.

In this research, we have tested with the Amazon cloud provider, since it is important that it can be scaled horizontally and vertically, in addition to providing easily integrated components to carry out the recommendation system. This system is available through the Amazon SageMaker, Notebooks, S3, Lambda, Api Gateway and Single Sign One. We use the Api Gateway to manage the $recommendations/ < idUser >$ API with the lambda, which allows us to expose the sagemaker model via the $boto3$ library and Sagemaker allows training and the deployment of models. The data are consumed from POJ through the python $requests$ library and the test and training sets are saved in S3 files. Finally, Single Sign One allows multiple access for the researchers with a single account enabling them to develop and test the architecture using several AWS services.

4. Experimentation and Results

4.1. Models and parameterization

The models used in this research are those based on KNN and Matrix Factorization. With KNN we have Cosine, Pearson and Mean Squared Difference (MSD) similarity and the models based on Matrix Factorization are Singular Value Descomposition (SVD), Non-Negative Matrix Factorization (NMF) and DeepMF that blends matrix factorization with deep learning.

a. KNN models based

Model	Number of neighbors	Minimum number of neighbors	User based
KNN models for Codechef	30	1	True
KNN models for NinjaCoding	5	1	True

Table 1. Cosine, Pearson and MSD models parameters

b. Matrix Factorization models based

Model	Number of factors	Number of epochs	Regularizer	Learning rate
SVD for Codechef	10	20	0.02	0.005
SVD for NinjaCoding	20	50	0.1	0.02

Table 2. SVD model parameters

Model	Number of factors	Number of epochs	Regularizer	Learning rate users	Learning rate items
NMF for CodeChef	50	50	0.02	0.005	0.005
NMF for NinjaCoding	15	20	0.1	0.02	0.001

Table 3. NMF model parameters

Model	Latent dimensions	Regularizer lambda	Number of epochs	Number of layers
DeepMF for CodeChef	50	0.02	10	4
DeepMF for NinjaCoding	15	0.1	10	4

Table 4. DeepMF model parameters

4.2. Results and Discuss

The RMSE results obtained from different datasets for models Cosine, DeepMF, MSD, NMF, Pearson, and SVD have been presented in Table 5. The datasets used for evaluation include different samples of CodeChef and NinjaCoding. The results presented indicate that the DeepMF model improves its predictions as the amount of data increases. This suggests that DeepMF is a robust and reliable method for handling large datasets. Another pattern is that the Cosine, Pearson, and MSD models do not appear to differ from each other when dealing with small datasets. However, to validate if these differences are significant we need to validate through statistical tests.

Dataset	Cosine	Pearson	MSD	SVD	NMF	DeepMF
CodeChef-30k	0.4261	0.4294	0.4413	0.4123	0.4603	0,4348
CodeChef-20k	0.4242	0.4219	0.4329	0.4122	0.4588	0,4346
CodeChef-15k	0.4184	0.4166	0.4249	0.409	0.453	0,4342
CodeChef-10k	0.4222	0.4187	0.4278	0.4124	0.4591	0,4295
CodeChef-5k	0.4165	0.4175	0.4175	0.4175	0.4478	0.4399
CodeChef-4k	0.4173	0.4176	0.4191	0.4179	0.4389	0.4347
CodeChef-3k	0.4095	0.4103	0.4117	0.4064	0.4275	0.4422
CodeChef-2k	0.4063	0.4049	0.4067	0.4066	0.4138	0.4342
CodeChef-1k	0.3938	0.3969	0.3969	0.3944	0.4064	0.4319
CodeChef-500	0.4368	0.4368	0.4368	0.4342	0.4450	0.4368
CodeChef-400	0.4240	0.4240	0.4240	0.4250	0.4271	0.4641
CodeChef-300	0.4380	0.4380	0.4380	0.4353	0.4438	0.4349
CodeChef-200	0.4312	0.4312	0.4312	0.4309	0.4392	0.4932
CodeChef-100	0.4132	0.4132	0.4132	0.4166	0.4094	0.4917
NinjaCoding	0.5083	0.4827	0.4996	0.4534	0.4892	0.4947

Table 5. RMSE results of each dataset and model

According to the Shapiro-Wilks test results in Table 6, the distribution of the Table

Model	p-value
Cosine	0.00053
DeepMF	0.00017
MSD	0.00493
NMF	0.68700
Pearson	0.01150
SVD	0.43300

Table 6. Shapiro-Wilks test of RMSE results

5 data is not balanced. This is particularly evident in the values obtained for $Cosine$ and $DeepMF$, which are 0.00053 and 0.00017, respectively, indicating that these two models have the most significantly non-normal distributions. In contrast, the Shapiro Wilks test statistic for NMF is 0.68700, suggesting that its distribution is the closest to normal. The values obtained for $Pearson$ and SVD are also different, with 0.01150 and 0.43300, respectively. This indicates that the distribution of these methods is not balanced, and this is further supported by the visual evidence presented in Figure 1. It clearly shows that the data points deviate significantly from the straight line, indicating that the distribution of the data for these methods is also not normal.

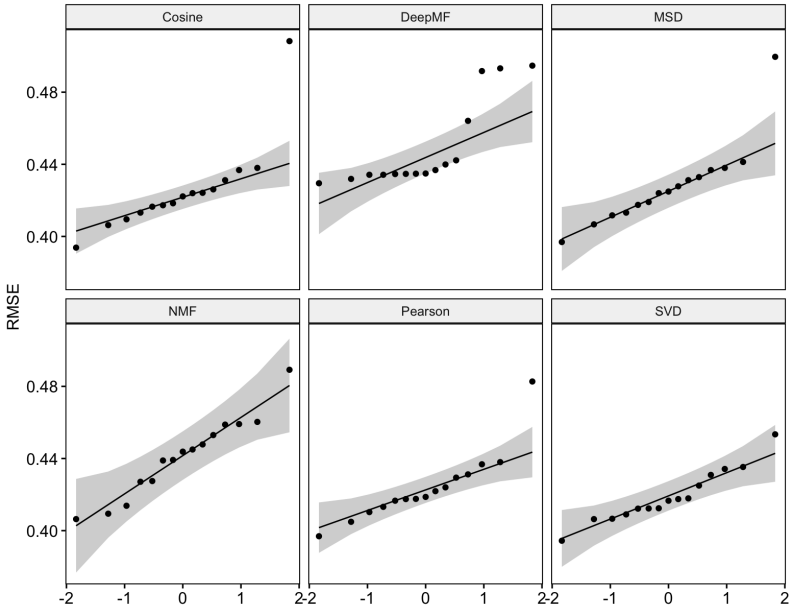

Figure 1. Q-Q plot showing the normal distribution of RMSE results

As the the Shapiro-Wilks test demonstrated that the results were not balanced, we use a non-parametric test (Friedman test) to know if there are significant differences. The result of the Friedman test showed that the p-value was equal to 0.000000275, suggesting that there is extremely strong evidence against the null hypothesis. This indicates that at least one of the model pairs differs significantly from the rest. To identify the pairs of models that differ significantly, a post-hoc Wilcoxon test with Bonferroni adjustment was executed and the results are presented in Table 7. These results show that pairs $DeepMF - SVD$, $NMF - Pearson$, and $NMF - SVD$ are significantly different, in

contrast to other pairs of models.

Models Comparison	p-value	p-value-adj	significance
Cosine - DeepMF	0.004	0.058	ns
Cosine - MSD	0.059	0.888	ns
Cosine - NMF	0.003	0.05	ns
Cosine - Pearson	0.541	1	ns
Cosine - SVD	0.088	1	ns
DeepMF - MSD	0.012	0.18	ns
DeepMF - NMF	0.639	1	ns
DeepMF - Pearson	0.001	0.021	*
DeepMF - SVD	0.000122	0.002	**
MSD - NMF	0.003	0.039	*
MSD - Pearson	0.014	0.214	ns
MSD - SVD	0.01	0.152	ns
NMF - Pearson	0.000183	0.003	**
NMF - SVD	0.000305	0.005	**
Pearson - SVD	0.024	0.357	ns

Table 7. Wilcoxon test with Bonferroni adjustment

When evaluating the efficiency of machine and deep learning models, it is important to consider not only their predictive accuracy but also their execution time. This becomes particularly relevant when dealing with large datasets or when quick responses are needed in real-time applications. For this reason, we ran 20 thousand interactions to measure the time of execution (presented in Figure 2). As the figure shows, the Matrix factorization models (SVD, NMF, DeepMF) were found to be faster than the K-Nearest Neighbors models (MSD, Cosine, Pearson). This difference in execution time can be attributed to the fact that NMF, DeepMF and SVD reduce the dimensionality of the data, making it easier and faster to find patterns and make predictions. In contrast, KNN relies on the original high-dimensional data, which can be slow to process.

Regarding the execution time of each model according to CPU and Memory, it does not differ much. Therefore, if we use $large$, $xlarge$ or $2xlarge$ for any model, the time for each is almost the same. SageMaker instance types such as $ml.t3.medium$ and $ml.t3.large$ do not support 10000 and 30000 interactions, due to their limited memory of $4GiB$ and $8GiB$. Nevertheless, the 30000 interactions go well on all models with instances larger than $ml.t3.xlarge$, which have memory more than $16GiB$.

5. Conclusions and future works

This research offers an architecture powered by cloud computing that supports machine and deep learning models, to recommend programming problems to the judges online. As a result, we suggest a cloud-based architecture to choose the best recommended algorithm with large datasets or when quick responses are needed. In addition, we explored six collaborative filtering-based algorithms for recommender systems and after conducting statistical tests and evaluating the execution time of these models, we found that the SVD model was the best-performing one in terms of both statistical significance and experimental evaluation. One of the limitations we found in this work is related to the

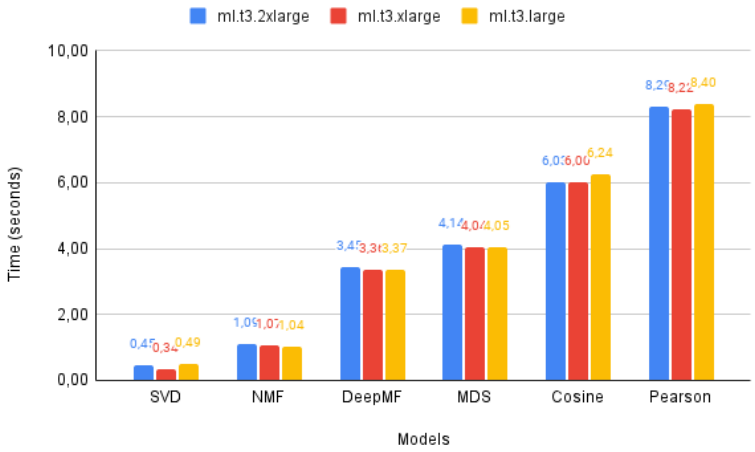

Figure 2. Time execution in seconds for each model

strategy applied by Toledo and Mota [2014], which limits us to use models based on the Cosine similarity function, because divisions by zero are not possible. Thereby, we use 1 instead of 0 and 2 for the other cases. Moreover, the $user\_id$ class in the interaction matrix must be transformed to an integer type so that models based on non-negative matrix factorization can be used. Finally, our experimental evaluation shows that more data requires more CPU and Memory capacity, which can be easily scaled across different Amazon SageMaker instances so that machine and deep learning models can be executed.

Our future work will be focused on using a content based recommendation approach in another lambda, to process the data and choose the best algorithm. Also, another direction will be to explore more models and their combination and comparison with the approaches proposed in this work.

References

M. F. Aljunid and M. Dh. An efficient deep learning approach for collaborative filtering recommender system. *Procedia Computer Science*, 171:829–836, 2020.

N. Bhalse and R. Thakur. Algorithm for movie recommendation system using collaborative filtering. *Materials Today: Proceedings*, 2021.

J. Bobadilla, F. Ortega, A. Gutiérrez, and S. Alonso. Classification-based deep neural network architecture for collaborative filtering recommender systems. 2020.

M. Caro-Martinez and G. Jimenez-Diaz. Similar users or similar items? comparing similarity-based approaches for recommender systems in online judges. In *International Conference on Case-Based Reasoning*, pages 92–107. Springer, 2017.

A. Cruz, C. S. Neto, P. Cruz, and M. Teixeira. Utilização da plataforma beecrowd de maratona de programação como estratégia para o ensino de algoritmos. In *Anais Estendidos do XXI Simpósio Brasileiro de Jogos e Entretenimento Digital*, pages 754–764, Porto Alegre, RS, Brasil, 2022. SBC. doi: 10.5753/sbgames_estendido.2022.225898. URL `https://sol.sbc.org.br/index.php/sbgames_estendido/article/view/23713`.

N. M. Devadiga. Tailoring architecture centric design method with rapid prototyping. In *2017 2nd International Conference on Communication and Electronics Systems (ICCES)*, pages 924–930. IEEE, 2017.

P. Fantozzi and L. Laura. Collaborative recommendations in online judges using autoencoder neural networks. In *International Symposium on Distributed Computing and Artificial Intelligence*, pages 113–123. Springer, 2020.

C. M. Intisar, Y. Watanobe, M. Poudel, and S. Bhalla. Classification of programming problems based on topic modeling. In *Proceedings of the 2019 7th International Conference on Information and Education Technology*, pages 275–283, 2019.

W. Julca-Mejia, H. D. Calderon-Vilca, and F. C. Cárdenas-Mariño. Evaluation of source code in acm icpc style programming and training competitions. In *Avances en Ingenieria de Software a Nivel Iberoamericano, CIbSE 2018*, 2018.

D. Kluver, M. D. Ekstrand, and J. A. Konstan. Rating-based collaborative filtering: algorithms and evaluation. *Social Information Access*, pages 344–390, 2018.

R. Lara-Cabrera, Á. González-Prieto, and F. Ortega. Deep matrix factorization approach for collaborative filtering recommender systems. *Applied Sciences*, 10(14):4926, 2020.

F. Martínez-Plumed, L. Contreras-Ochando, C. Ferri, J. Hernández-Orallo, M. Kull, N. Lachiche, M. J. Ramírez-Quintana, and P. Flach. Crisp-dm twenty years later: From data mining processes to data science trajectories. *IEEE Transactions on Knowledge and Data Engineering*, 33(8):3048–3061, 2019.

G. L. McDowell. *Cracking the coding interview: 189 programming questions and solutions*. CareerCup, LLC, 2015.

F. Ortega, A. Hernando, J. Bobadilla, and J. H. Kang. Recommending items to group of users using matrix factorization based collaborative filtering. *Information Sciences*, 345:313–324, 2016.

F. D. Pereira, H. B. Junior, L. Rodriguez, A. Toda, E. H. Oliveira, A. I. Cristea, D. B. Oliveira, L. S. Carvalho, S. C. Fonseca, A. Alamri, et al. A recommender system based on effort: Towards minimising negative affects and maximising achievement in cs1 learning. In *International Conference on Intelligent Tutoring Systems*, pages 466–480. Springer, 2021.

M. M. Rahman, Y. Watanobe, R. U. Kiran, T. C. Thang, and I. Paik. Impact of practical skills on academic performance: A data-driven analysis. *IEEE Access*, 9:139975–139993, 2021.

M. M. Rahman, Y. Watanobe, T. Matsumoto, R. U. Kiran, and K. Nakamura. Educational data mining to support programming learning using problem-solving data. *IEEE Access*, 10:26186–26202, 2022.

R. Y. Toledo and Y. C. Mota. An e-learning collaborative filtering approach to suggest problems to solve in programming online judges. *International Journal of Distance Education Technologies (IJDET)*, 12(2):51–65, 2014.

H. Wu, Y. Liu, L. Qiu, and Y. Liu. Online judge system and its applications in c language teaching. In *2016 International Symposium on Educational Technology (ISET)*, pages 57–60. IEEE, 2016.

R. Yera and L. Martínez. A recommendation approach for programming online judges supported by data preprocessing techniques. *Applied Intelligence*, 47(2):277–290, 2017.