Unified Workflow for Training and Offline Evaluation of Recommendation Systems

Veljko Lončarević[1], Aleksa Iričanin[2], Stefan Ćirković[3], Vanja Luković[4]
University of Kragujevac, Faculty of Technical Sciences
Čačak, Serbia
Email: [1] veljkoloncarevicharry@gmail.com
ORCID: [1] [0009-0007-4296-2709], [2] [0009-0006-8145-403X],
[3] [0009-0004-6775-1543], [4] [0000-0002-1887-6102]

***Abstract*— This paper proposes a streamlined, unified workflow for training and offline evaluation, designed to standardize the process and enhance reproducibility across various recommendation models. The workflow integrates key components, including data preprocessing, model training, hyperparameter tuning, and metric computation, into a cohesive framework. To demonstrate its applicability, the MovieLens 25M dataset is utilized as a benchmark, and both the k-Nearest Neighbor and Matrix Factorization are implemented as recommendation models. The proposed workflow systematically evaluates the models using accuracy-based and error-based metrics for comparison. By providing a unified framework, this research aims to facilitate better comparisons across recommendation algorithms, promote transparency in evaluation, and accelerate innovation in the field.**

Keywords*— *recommendation systems, offline evaluation, unified workflow, evaluation metrics, reproducibility

I. Introduction

Recommendation systems have become an integral part of modern digital platforms, enhancing user experiences by delivering personalized content in various domains such as e-commerce, streaming services, and social networks. Offline evaluation techniques, which leverage historical user interaction data to measure algorithmic effectiveness without requiring real-time user feedback, are particularly valuable in scenarios where online testing is impractical or undesired. Online testing, such as A/B testing, requires deploying algorithms in a live environment, which can be resource-intensive, time-consuming, or risky, especially if the system underperforms and negatively impacts the user experience [1]. In contrast, offline evaluation provides a controlled and efficient way to benchmark algorithms by simulating their performance on previously collected data, making it an essential step in developing and validating recommendation systems before real-world deployment. While offline evaluation provides a cost-effective and scalable alternative to online testing, its effectiveness depends heavily on the rigor and consistency of the evaluation process.

Despite the widespread adoption of offline evaluation, the lack of standardized workflows poses significant challenges for practitioners and researchers. Inconsistent data preprocessing practices, the variety of evaluation metrics, and fragmented experimental designs often hinder reproducibility and comparability across studies. These issues become particularly pronounced when evaluating different recommendation algorithms, such as collaborative filtering, content-based methods, or hybrid approaches, on large-scale datasets [2].

To address these challenges, this paper introduces a unified workflow for offline evaluation of recommendation systems. The workflow consolidates key stages of the evaluation process, including data preparation, model training, hyperparameter optimization, prediction generation, and metric computation, into a cohesive and systematic framework. By streamlining these steps, the proposed workflow ensures consistency, enhances reproducibility, and provides a robust foundation for benchmarking various recommendation algorithms.

To demonstrate the utility of this unified workflow, it is applied to the well-known MovieLens dataset [3], using the k-Nearest Neighbor and Matrix Factorization algorithms as a case study. The algorithms' performance is evaluated using a combination of accuracy-based and error-based metrics, showcasing the workflow's adaptability to different evaluation criteria. The findings underscore the strengths and limitations of this workflow on a large-scale dataset, while also highlighting the broader implications of adopting a standardized workflow for offline evaluation.

II. Methodology

The proposed workflow is designed to streamline the training and evaluation process, ensure consistency across experiments, and facilitate reproducibility. The workflow consists of four key components: data preparation, model training, hyperparameter optimization, and metric computation, as shown on Fig. 1. This section outlines each component in detail and describes their role in the unified framework.

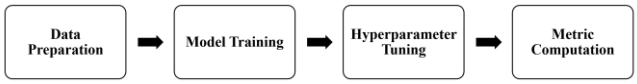

Fig. 1. Main methodology components

A. Data Preparation

Data preparation is the first step in the workflow, making sure the dataset is properly structured, and cleaned to facilitate reliable training and evaluation of recommendation models. This step involves several distinct processes, each of which addresses a specific aspect of data quality and suitability.

The first process is **Data Cleaning – e**nsuring the dataset is free of errors and inconsistencies is vital for accurate model training and evaluation [4]. It includes the following sub-processes:

- **Removing Duplicates –** Duplicate entries can distort evaluation metrics and unfairly favor certain

recommendations. Duplicate interactions are identified and removed to maintain data integrity.

- **Handling Missing Values –** Missing data can introduce bias or reduce model accuracy. Techniques like mean imputation (for numerical features), mode imputation (for categorical features), or collaborative filtering can be used to estimate missing values.
- **Addressing Inconsistencies –** Erroneous values (e.g., invalid timestamps or out-of-range ratings) are identified and corrected or removed. For example, ratings outside a predefined scale (e.g., 1 to 5) might be flagged for correction.

The order of aforementioned subprocesses is as shown in Fig. 2.

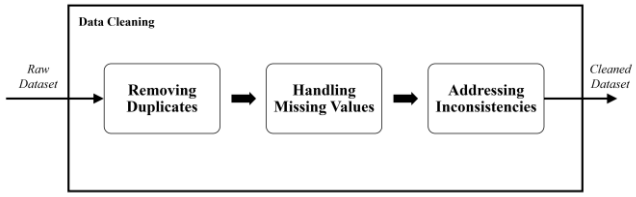

Fig. 2. Order of subprocesses during data cleaning

The second process of data preparation is **Feature Engineering**, which enhances the dataset by generating additional variables that provide more context or predictive power for the recommendation algorithm. As part of the workflow, it is suggested to first consider the following types of features:

- **User Features** – Incorporating user demographics (e.g., age, gender, location) can help the model capture user-specific preferences.
- **Item Features** – Attributes like genre, category, or price of an item can enrich the data and improve recommendation quality.
- **Contextual Features** – Features such as timestamps, device type, or session information can provide additional context to better model user behavior.
- **Interaction Features** – Deriving features from historical interactions, such as the average rating a user gives or the popularity of an item, can reveal trends or biases in the data.

The final process of data preparation is **Data Splitting,** where the dataset is split into three parts – training, validation and test. Dividing the dataset into distinct subsets for training, validation, and testing is crucial to evaluate model performance effectively. For datasets involving time-sensitive interactions (e.g., movie ratings or purchases over time), a temporal split is recommended. Here, interactions are ordered chronologically, and the most recent interactions are allocated to the validation and test sets. This approach simulates real-world scenarios where future interactions are predicted based on past behavior. For datasets without a temporal component or when interactions are static, a random split can be employed. This method randomly assigns data points to the training, validation, and test sets, ensuring a balanced distribution of users and items across subsets. Common splitting ratios are 80% for training, 10% for validation, and 10% for testing [5], though these can be adjusted based on dataset size and application requirements.

B. Model Training

The model training step involves creating a model that can accurately predict user preferences or recommend relevant items. This phase translates raw input data into a trained model capable of generating recommendations by learning patterns, relationships, and preferences within the data. The processes in this step are not necessarily sequential and allow for combinations of different categories. Below is presented a detailed breakdown of the key processes involved in the model training step, as shown on Fig. 3.

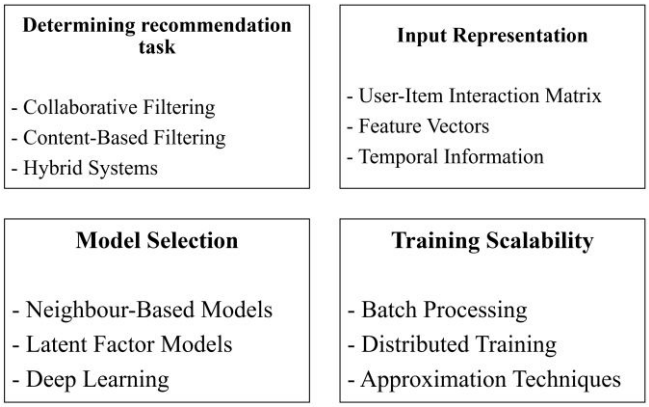

Fig. 3. Model training processes

Before training the model, the first process – **determining recommendation task** – requires to determine the type of recommendation system being developed. The task can generally fall into one of the following categories [6]:

- **Collaborative Filtering** – Uses user-item interaction data to recommend items based on similarities between users or items.
- **Content-Based Filtering –** Recommends items similar to what a user has interacted with, based on features of the items (e.g., genres, descriptions).
- **Hybrid Systems –** Combines collaborative filtering and content-based methods for improved accuracy and diversity.

The second process, **Input Representation**, consists of choosing the best way to prepare the dataset for model training. Preparing the data for the model involves converting dataset from the previous step containing preprocessed user-item interactions into a structured format suitable for training. Some of the most commonly used techniques for input representation are as follows:

- **User-Item Interaction Matrix –** The core input for collaborative filtering models, representing user preferences as a sparse matrix (rows as users, columns as items, and values as interactions such as ratings or clicks).
- **Feature Vectors –** For content-based or hybrid models, user and item features are encoded as vectors, often including metadata such as demographics, item attributes, or contextual information.

- **Temporal Information –** Time-based data can be included to capture evolving user preferences or item popularity.

Choosing a model architecture is closely connected to the type of a recommendation task at hand, as well as the amount of data available. Most commonly used choices include:

- **Neighbor-based Models** – Simpler models like k-Nearest Neighbors (kNN), which recommend based on the most similar users or items.
- **Latent Factor Models** – Techniques like Matrix Factorization (e.g., Singular Value Decomposition or Alternating Least Squares) that decompose the interaction matrix into latent features for users and items.
- **Deep Learning Models** – Neural networks such as autoencoders, recurrent neural networks (RNNs), or transformers are used for complex tasks involving high-dimensional data or sequential patterns.

Training recommendation models on large datasets is often computationally expensive, therefore various techniques have been invented in order to improve **scalability** of the model training, and some of the most commonly used are:

- **Batch Processing** – Dividing the data into smaller subsets (batches) for processing.
- **Distributed Training** – Leveraging distributed systems (e.g., Apache Spark, TensorFlow distributed) to parallelize computations.
- **Approximation Techniques** – Reducing computation complexity, such as using approximate nearest neighbor (ANN) methods for large-scale kNN.

C. Hyperparameter Optimization

Hyperparameter optimization is the step in which the best combination of hyperparameters is selected for a machine learning model to maximize its performance on a given task. Unlike model parameters (e.g., weights in a neural network) that are learned during training, hyperparameters are set before training begins and influence how the model learns or performs. Examples of hyperparameters include learning rate in gradient descent algorithms, regularization strength (e.g., L1 or L2 penalty), number of latent factors in matrix factorization, number of neighbors in k-Nearest Neighbors, batch size or number of epochs in deep learning. Proper hyperparameter optimization can significantly improve a model's accuracy, generalizability, and efficiency. Tuning hyperparameters can minimize training loss and improve metrics on validation or test datasets. Properly chosen hyperparameters help balance bias and variance, ensuring the model generalizes well to unseen data. Optimized hyperparameters can reduce computational time and resource usage by avoiding poorly performing configurations. Different datasets and tasks require different hyperparameter configurations for optimal results. Hyperparameter optimization is carried out in four processes outlined in Fig. 4.

Identifying the hyperparameters for optimization is the first process in hyperparameter optimization. This involves determining which settings of the model or training process should be tuned to improve performance. Different models have distinct hyperparameters. For example, kNNs have the number of neighbors (k) and a distance metric (e.g., Euclidean, cosine), while neural networks have learning rate, batch size, number of layers, number of units per layer, and dropout rate. It is also necessary to focus on hyperparameters that significantly affect performance. For example, in neural networks, the learning rate is usually critical. It is also advised to avoid optimizing too many hyperparameters simultaneously, as it can increase computational cost and complexity.

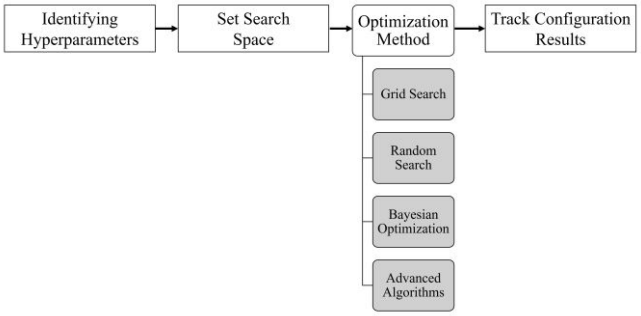

Fig. 4. Hyperparameter optimization processes

The search space defines the range of possible values for each hyperparameter. Properly defining this space is essential to ensure that the optimization algorithm can explore meaningful configurations. Different hyperparameters may accept different types of values, including continuous, discrete and categorical values. It is advised to base the range on prior knowledge of the problem and the algorithm. Best practices include using domain expertise to narrow down search ranges and avoid wasteful exploration, and starting with a broader range for initial searches and refining based on initial results.

Selecting an optimization method depends on the size of the search space, available computational resources, and the importance of hyperparameter interactions. Common optimization methods include the following:

- **Grid Search** - Systematically evaluates all possible combinations of hyperparameters in the defined search space. Best for small and discrete search spaces but computationally expensive for large ones.
- **Random Search** - Samples random combinations from the search space. Efficient for high-dimensional search spaces, as it covers a broader area compared to grid search.
- **Bayesian Optimization** - Models the objective function probabilistically and uses prior evaluations to inform future exploration. Efficient for expensive-to-train models but computationally complex.
- **Advanced Algorithms** – e.g., **Hyperband** combines random search with early stopping, which is efficient for scenarios with limited computational resources. **Evolutionary algorithms** mimic natural selection by evolving hyperparameter configurations over multiple generations, which is suitable for very complex or non-differentiable search spaces [7].

The process of systematically **tracking and recording the results of different hyperparameter configurations** is important to analyze and compare performance of different hyperparameter configurations. It is necessary to track hyperparameter settings, including the values of each

hyperparameter for every run (e.g., learning rate, batch size, number of latent factors); performance metrics - key evaluation metrics like accuracy, precision, recall, RMSE, MAE, or NDCG for each configuration; training statistics - metrics like training time, validation loss, and resource utilization (e.g., memory or GPU usage); and metadata, including details such as dataset version, runtime environment, and seed values for reproducibility. Automatic logging tools are available for this purpose - it is advised to use tools or frameworks designed for experiment tracking, such as MLflow.

D. Metric Computation

The final component of the proposed workflow focuses on evaluating the model's performance. This phase ensures a comprehensive understanding of how well the recommendation system meets its objectives by employing various evaluation metrics. These metrics allow researchers and practitioners to assess different aspects of the model, from accuracy to broader impacts like user experience. The workflow supports the following categories of evaluation (Fig. 5):

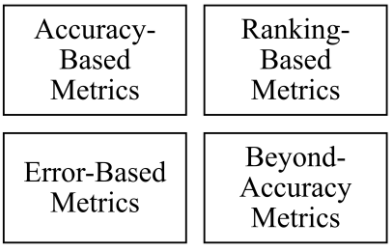

Fig. 5. Types of metrics for model evaluation

- **Accuracy-Based Metrics** – Accuracy-based metrics measure how effectively the recommendation system identifies relevant items for users. These metrics are particularly useful when the goal is to optimize for the correct identification of items within the top-k recommendations. The most commonly used metrics in this category are Precision@k, Recall@k, F1@k.
- **Ranking-Based Metrics** – Ranking-based metrics evaluate the quality of the order in which recommendations are presented. These metrics ensure that not only are the correct items included in the recommendations, but that they are prioritized effectively. These metrics include Normalized Discounted Cumulative Gain (NDCG) which measures the usefulness of recommendations based on their rank position; Hit Rate (HR@k) which measures the proportion of users for whom at least one relevant item appears in the top-k recommendations.
- **Error-Based Metrics** – Error-based metrics assess the accuracy of the predicted ratings compared to the true ratings. These metrics are especially important for models designed to predict explicit ratings, such as star ratings for movies. Most commonly used metrics in this category include Root Mean Square Error (RMSE) and Mean Absolute Error (MAE).
- **Beyond-Accuracy Metrics** – Accuracy is not always sufficient to evaluate recommendation systems comprehensively. Beyond-accuracy metrics consider aspects that improve the overall user experience and system utility [8]. These metrics are Diversity, which evaluates how varied the recommendations are. High diversity ensures that users are exposed to a broader range of items, reducing redundancy; Serendipity, which measures the ability of the system to suggest items that are both relevant and surprising to the user; Coverage, which evaluates the proportion of the total item catalog that is recommended across all users; Novelty, which focuses on recommending items that are less popular or less well-known, helping users discover new content.

III. The Experiment

We conducted experiments on the large-scale MovieLens dataset, which consists of 32 million user ratings for movies. Our objective was to evaluate the performance of two recommendation approaches: K-Nearest Neighbors and Matrix Factorization. The experiments were implemented in Python using Jupyter Notebook and libraries such as pandas, numpy, scikit-surprise and scikit-learn.

A. Data Preparation

The dataset was analyzed to ensure quality, revealing no duplicates or missing values. It was then transformed into a user-item rating matrix, where rows represented users, columns represented movies, and cells contained ratings (Fig. 6). Missing values were filled with zeros for KNN, while MF naturally handled sparsity. A temporal split strategy was applied: 80% of the oldest ratings were allocated for training, with 10% each for validation and testing, simulating real-world recommendation scenarios.

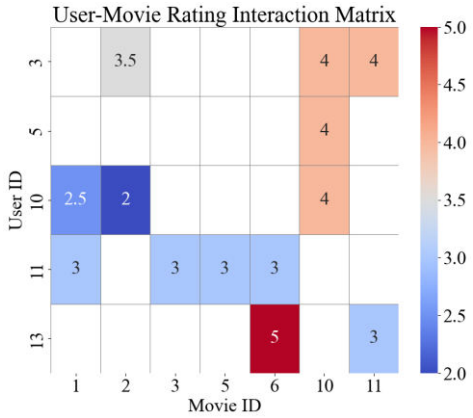

Fig. 6. User-rating interaction matrix sample

B. Model Training

KNN was employed using item-based collaborative filtering. Similarity between items was computed using cosine similarity, and predictions were made by averaging the ratings of the k most similar items. The initial k value was set to 10, with tuning performed during optimization.

MF decomposes the user-item matrix into latent factors, capturing hidden relationships. We implemented Singular Value Decomposition (SVD), which factorizes the matrix into user and item embeddings. Predictions were generated by computing the dot product of these embeddings.

Both models had problems handling large-scale MovieLens dataset (which contains over 32 million rows) without any optimizations. To handle the large MovieLens dataset, KNN required optimization techniques – batch processing combined with parallelization and the use of

KNNWithMeans from scikit-surprise. Batch processing allowed the model to manage memory usage by processing smaller subsets of the data at a time, while parallelization accelerated computation by distributing the workload across multiple processors. KNNWithMeans further improved performance by using user-based mean ratings to handle large sparse datasets efficiently. On the other hand, Matrix Factorization SVD was optimized using mini-batches with Stochastic Gradient Descent (SGD), allowing the model to incrementally update latent factors based on small batches of data, thus reducing memory requirements and speeding up convergence without needing to process the entire matrix at once.

C. *Hyperparameter Optimization*

Grid Search was used to tune hyperparameters for both models:

- **KNN**: The optimal number of neighbors (k) was selected from {10, 15, ..., 50}, tested with different distance metrics ('euclidean', 'manhattan', 'minkowski') and weighting schemes ('uniform', 'distance').
- **MF**: The number of latent factors was varied in {10, 20, ..., 100}, with different regularization values to prevent overfitting.

Optimization was guided by RMSE. Grid Search was performed to optimize hyperparameters for both KNN and MF models. For KNN the best configuration was determined to be k=50 and MSD similarity. For MF, the search identified n_factors=100 and reg_all=0.1 as optimal. The results of this parameter tuning are summarized in Fig. 7.

Model	Best Parameters
KNN	k = 50, sim_options = {name: 'msd', user_based: False}
MF (SVD)	n_factors = 100, reg_all = 0.1

Fig. 7. Grid search results

D. *Metric Computation*

Once the hyperparameters were optimized using the training and validation subsets, the models were retrained using the entire training subset (80% of the data) along with the selected hyperparameters. This allowed the models to learn from as much data as possible before final evaluation. After retraining, the models were evaluated on the test subset (10% of the data) to assess its final performance. Performance metrics (RMSE, Precision@k, MAE, F1@k) were calculated on the test set. This test set was kept separate during the optimization process, ensuring that the models' evaluation on this set reflected their ability to generalize to entirely unseen data. After calculating performance metrics, the results were stored into a separate csv file, allowing comparison between two models (Fig. 8).

Model	RMSE	MAE	Precision@k	F1@k
KNN	0.9334	0.711455	0.65168	0.63922
SVD	0.90202	0.696733	0.65664	0.66404

Fig. 8. Model performance comparison

The evaluation results of the KNN and SVD models show differing strengths depending on the metric. The KNN model has a higher RMSE and MAE, indicating that it might produce larger errors in prediction. However, its Precision@k (0.6517) and F1@k (0.6392) are lower compared to the SVD model, which performs slightly better in these metrics (Precision@k of 0.6566 and F1@k of 0.6640). This suggests that while KNN may be less accurate overall, it might offer better recommendation relevance for specific users, especially when the focus is on ranking precision and balance between precision and recall. On the other hand, SVD, with better RMSE and MAE, might be preferable when prediction accuracy is more critical. The choice between the models should depend on the specific use case, such as prioritizing accurate predictions (SVD) or higher relevance in recommendations (KNN).

During the model evaluation, the Python logging library is employed to track and store detailed information about the progress and results of the model tuning, listing dataset version, timestamp, model version and performance metrics, which allow for thorough analysis of previous and current versions of the model (Example given on Fig. 9.).

Dataset Version	Timestamp	Model Version	RMSE	Precision @k	MAE	F1@k	Coverage
1	28-01-25 10:15	1.1	0.85	0.75	0.7	0.78	0.9
1	28-01-25 11:30	1.2	0.8	0.78	0.65	0.8	0.92
1	28-01-25 12:45	1.3	0.88	0.72	0.74	0.76	0.89
1	28-01-25 14:00	2.1	0.82	0.77	0.68	0.79	0.91
1	28-01-25 15:15	2.2	0.79	0.8	0.64	0.82	0.93
1	28-01-25 16:30	2.3	0.83	0.76	0.69	0.78	0.9
1	28-01-25 17:45	2.4	0.81	0.79	0.66	0.81	0.94
1	28-01-25 19:00	2.5	0.84	0.74	0.71	0.77	0.88
1	28-01-25 20:15	2.6	0.87	0.71	0.73	0.75	0.92
1	28-01-25 21:30	3	0.8	0.79	0.67	0.8	0.91

Fig. 9. Model version analysis

This enables a clear comparison of current and past models, facilitating informed decisions about the best-performing model for deployment.

IV. Similar Research

Our work proposes a unified workflow for training and offline evaluation of recommendation systems, focusing on reproducibility and standardized evaluation. This aligns with two key frameworks in the field: FEVR [9] and Elliot [10].

The FEVR framework organizes evaluation processes by categorizing facets like evaluation goals, methods, and metrics. While similar in its emphasis on comprehensive evaluation, our work provides a more practical, executable framework, integrating data preprocessing, model training, and metric computation into a unified process, making it easier to compare recommendation algorithms.

The Elliot framework standardizes the entire evaluation pipeline, including hyperparameter optimization and statistical analysis. While Elliot offers more flexibility with a wide range of data processing strategies and metrics, our approach focuses on simplicity, providing a streamlined and easily reproducible workflow.

While both FEVR and Elliot contribute significantly to the evaluation landscape, our research provides a focused, user and beginner-friendly workflow that prioritizes clarity, ease of use, and reproducibility.

V. Conclusion

The workflow employed in this experiment significantly streamlined the process of training and offline evaluation of a

recommendation system, transforming it into a more efficient, systematic, and transparent procedure. By adhering to a well-defined sequence of four core steps—Data Preparation, Model Training, Hyperparameter Optimization, and Metric Computation—we were able to methodically optimize two models for performance and rigorously evaluate their predictive capabilities. Each step played a pivotal role in ensuring models' effectiveness, from preprocessing the data to the final assessment of their performance on unseen test data.

Together, these four steps created a comprehensive and structured framework for training and offline evaluation of a recommendation system. By following this workflow, we were able to optimize the models for performance, ensure that they were robustly evaluated, and produce a recommendation system that is capable of making reliable predictions. The methodology is not only valuable for this specific experiment but also serves as a solid foundation for future research in recommendation systems, providing a clear, reproducible process for building models and evaluating their effectiveness.

ACKNOWLEDGMENT

This study was supported by the Ministry of Science, Technological Development and Innovation of the Republic of Serbia, and these results are parts of Grant No. 451-03-66 / 2024-03 / 200132 with the University of Kragujevac - Faculty of Technical Sciences Čačak.

REFERENCES

[1] J. Beel and S. Langer, A Comparison of Offline Evaluations, Online Evaluations, and User Studies in the Context of Research-Paper Recommender Systems, Research and Advanced Technology for Digital Libraries. TPDL 2015. Lecture Notes in Computer Science, vol. 9316, pp. 153–168, 2015. doi: 10.1007/978-3-319-24592-8_12.

[2] S. Khusro, Z. Ali, and I. Ullah, "Recommender Systems: Issues, Challenges, and Research Opportunities," Information Science and Applications (ICISA) 2016. Lecture Notes in Electrical Engineering, vol. 376, pp. 1179–1189, 2016. doi: 10.1007/978-981-10-0557-2_112.

[3] F. Maxwell Harper and J. A. Konstan, "The MovieLens Datasets: History and Context," ACM Transactions on Interactive Intelligent Systems (TiiS), vol. 5, no. 4, pp. 1–19, Dec. 2015. doi: 10.1145/2827872.

[4] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang, "Data Cleaning: Overview and Emerging Challenges," Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16), pp. 2201–2206, 2016. doi: 10.1145/2882903.2912574.

[5] R. R. Picard and K. N. Berk, "Data Splitting," The American Statistician, vol. 44, no. 2, pp. 140–147, May 1990. doi: 10.1080/00031305.1990.10475704.

[6] S. Gupta and A. Maithani, "A Literature Review on Recommendation Systems," International Research Journal of Engineering and Technology (IRJET), vol. 7, no. 9, pp. 634–637, Sep. 2020. [Online]. Available: https://www.irjet.net/archives/V7/i9/IRJET-V7I9633.pdf.

[7] L. Cui, P. Ou, X. Fu, Z. Wen, and N. Lu, "A novel multi-objective evolutionary algorithm for recommendation systems," Journal of Parallel and Distributed Computing, vol. 103, pp. 53–63, 2017. doi: 10.1016/j.jpdc.2016.10.014.

[8] M. Ge, C. Delgado-Battenfeld, and D. Jannach, "Beyond accuracy: evaluating recommender systems by coverage and serendipity," in Proceedings of the Fourth ACM Conference on Recommender Systems, Barcelona, Spain, 2010, pp. 257–260. doi: 10.1145/1864708.1864761.

[9] E. Zangerle and C. Bauer, "Evaluating recommender systems: Survey and framework," ACM Comput. Surv., vol. 55, no. 8, pp. 170:1–170:38, Dec. 2022, doi: 10.1145/3556536. [Online]. Available: https://doi.org/10.1145/3556536.

[10] V. W. Anelli, A. Bellogín, A. Ferrara, D. Malitesta, F. A. Merra, C. Pomo, F. M. Donini, and T. Di Noia, "Elliot: a comprehensive and rigorous framework for reproducible recommender systems evaluation," Proc. 44th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval, 2021, pp. 1–10, arXiv:2103.02590.