International Journal of Multidisciplinary Engineering in Current Research - IJMEC

Volume 11, Issue 4s, April-2026, http://ijmec.com/, ISSN: 2456-4265

Full Length Article

Optimizing Mobile App Recommendations Using Crowdsourced Educational Data

Mohammed Rayan Ali[1], Mohammed Afnan Habeeb[2], Dr. Abdul Ahad Afroz[3]

[1,2]B.E.Students ;Department of Information Technology – ISL Engineering College Osmania University, Hyderabad, Telangana.

[3]Associate Professor ;Department of Information Technology – ISL Engineering College Osmania University, Hyderabad, Telangana.

Mail id; rayan.alimohd04@gmail.com, habeebafnan2002@gmail.com

Accepted 24-04-2026

Abstract

This paper presents a content-aware educational recommendation framework for mobile applications used by university students. The system addresses the discovery problem that emerges when learners must select among many similar apps for study planning, note sharing, research support, productivity, and collaboration. The proposed approach combines crowdsourced app metadata with student interaction traces and applies NLP-based preprocessing to transform text into structured features. A Random Forest classifier is trained to predict the most suitable academic cohort - Undergraduate, Postgraduate, or Graduate - and the resulting class probabilities are used to rank apps by relevance. The framework is designed to be lightweight, interpretable, and easy to deploy in a web or cloud environment. Compared with GRU-based sequence modeling and other classical baselines, the ensemble method offers stronger stability on sparse educational data, better resistance to overfitting, and easier interpretation through feature importance. The manuscript formalizes the project as a complete IEEE-style paper, including problem definition, methodology, architecture, implementation, evaluation, and future extensions.

Keywords

educational recommendation systems, mobile learning, crowdsourced data, natural language processing, Random Forest, TF-IDF, interpretable machine learning.

Introduction

Educational mobile applications have become indispensable companions for students who need flexible access to content, practice material, and productivity tools. The same market expansion that made learning resources more accessible also created a recommendation problem: most students encounter a large catalog of apps but lack a reliable mechanism for identifying which tools actually match their level of study and daily behavior. Recommendation quality matters because the value of an educational app depends not only on popularity but also on fit. A first-year undergraduate may need note-taking, flashcards, and scheduling tools, whereas a graduate student may benefit more from reference managers, analytical notebooks, and collaborative research platforms. A one-size-fits-all ranking method therefore produces weak personalization and often promotes redundant applications. The project behind this paper proposes a cohort-aware recommendation model that maps user and app signals into three academic categories - UG, PG, and Graduate - and then uses that prediction to produce ranked suggestions. The system combines crowdsourced metadata, textual descriptions, and observed interaction behavior. Text is normalized through NLP preprocessing, while structured fields are encoded and fused into a single feature vector. The main design objective is practical deployment. The model must be accurate enough to support personalized recommendations, but it must also remain lightweight, explainable, and easy to integrate into a conventional web stack. These requirements favor tree ensembles over heavy sequential models when the available data are sparse or noisy [1]-[3].

Problem Statement

The central problem is to recommend relevant educational applications from a large, heterogeneous catalog using limited and partially noisy crowdsourced evidence. App stores and institutional portals typically provide only short descriptions, ratings, and categorical tags, while student logs may be incomplete or uneven across cohorts. As a result, it is difficult to construct a reliable ranking model using raw records alone. This difficulty is amplified by class overlap. Educational apps that are useful for UG students may also be relevant for PG users, but for different reasons and with different priorities. Traditional recommendation methods often capture general popularity, yet they fail to distinguish subtle cohort-specific patterns. At the same time, sequence-based models such as GRU can model temporal behavior but require careful tuning and more training data than is usually available in a small academic deployment [4], [5]. The required solution should therefore perform three tasks simultaneously: normalize app

https://doi.org/10.63665/IJMEC.1104s.45
ISSN: 2456-4265
IJMEC 2026

descriptions, learn from mixed structured and textual features, and produce an interpretable class prediction that can be converted into a top-N recommendation list. It should also generalize under sparse data, remain computationally efficient during inference, and be easy for educators or administrators to audit. In short, the problem is to build a robust educational recommender that can work with real institutional data rather than idealized benchmark data, while still preserving interpretability and practical deployment efficiency.

Proposed Methodology

The methodology is organized into five stages: data acquisition, preprocessing, feature engineering, model training, and recommendation generation. The input data contain app-level information (name, description, category, rating) and student-level signals (usage counts, session frequency, preferred content type, and coarse academic level). Because these inputs are heterogeneous, the first task is to normalize them into a common representation.

Text preprocessing begins by converting strings to lowercase, removing punctuation and stop words, and applying tokenization and lemmatization. The output is transformed into TF-IDF vectors to emphasize informative terms while down-weighting common phrases. This is important for app descriptions, where generic marketing language can otherwise dominate the representation.

The feature space is then expanded with structured information. Categorical fields are one-hot encoded, numerical values are scaled when necessary, and the text vector is concatenated with behavioral variables. This fusion step yields a richer representation than a purely textual or purely collaborative design because it preserves both semantic and contextual cues.

Given a document d and a term t, the TF-IDF weight is defined as tfidf(t,d)=tf(t,d) x log(N/df(t)), where N is the number of documents and df(t) is the document frequency. This formulation assigns higher weight to terms that are frequent in a specific app description but rare in the wider corpus.

The target label y belongs to the set {UG, PG, Graduate}. During training, the Random Forest estimates P(y|x) for each class. At inference time, the system ranks candidate apps using this probability, possibly combined with similarity scoring or business constraints. The final recommendation list is therefore both predictive and operationally useful.

$$\mathbf{tfidf(t,d) = tf(t,d) \times \log(N / df(t))}$$

To support ranking, the class vote of the forest is computed by majority aggregation. If h_m(x) is the prediction of the m-th tree, the class estimate is y_hat = argmax_c sum_m I(h_m(x)=c). This majority-vote mechanism makes the model robust to noise in individual trees and reduces variance compared with a single decision tree.

At the split level, the tree growth criterion uses Gini impurity, Gini(S)=1-sum_k p_k^2. The chosen split is the one that maximizes the impurity reduction between parent and child nodes. This criterion is simple, efficient, and effective for multi-class educational labels.

$$\mathbf{\hat{y} = \arg\max_c \sum_{m=1}^{M} I(h_m(x)=c)}$$

$$\mathbf{Gini(S) = 1 - \sum_k p_k^2}$$

System Architecture

The system follows a modular client-server architecture. The user interface receives either a student profile or an app record and passes it to the preprocessing service. After normalization, the feature vector is sent to the trained Random Forest model, which returns a cohort prediction and a confidence score.

The recommendation layer uses this output to filter and rank candidate applications. If a user is classified closer to UG, the system prioritizes tools that match beginner-friendly learning behavior; if the predicted class is Graduate, it emphasizes research, analysis, and productivity utilities. This makes the final ranking more meaningful than raw popularity.

The architecture also supports decoupled deployment. The model can be updated without changing the front end, and the preprocessor can be cached or retrained independently. Such separation improves maintainability and makes the framework suitable for an educational portal, a cloud-hosted API, or a campus recommendation dashboard.

Module	Purpose
Data acquisition	Collect app metadata and student signals
Preprocessing	Normalize text and encode features
Model training	Fit the Random Forest classifier
Ranking service	Convert probabilities into top-N suggestions
Interface layer	Present results to students and admins

Algorithms and Models Used

The project compares a GRU-style sequence model from the baseline implementation with the proposed Random Forest approach. GRU is effective for temporal behavior, but in this problem its advantage is limited because the available records are sparse and the goal is cohort classification rather than long-horizon sequence prediction. Moreover, recurrent models require more tuning and are less transparent when administrators need to explain recommendations.

https://doi.org/10.63665/IJMEC.1104s.45
ISSN: 2456-4265
IJMEC 2026

Random Forest is a better fit for the present setting because it naturally handles nonlinear interactions between sparse text features and structured usage variables. By training each tree on a bootstrap sample and a random subset of features, the ensemble reduces variance and improves generalization. The model also remains fast enough for real-time use in a small to medium-scale educational service. A practical recommendation pipeline can be expressed as Algorithm 1. The algorithm first preprocesses text, then constructs a fused feature matrix, trains the classifier, and finally ranks candidate apps using predicted probabilities. Because the algorithm is modular, it can be reused with different data sources or alternative vectorizers without changing the deployment logic.

Algorithm 1: Random Forest-based educational app recommendation

Input: App metadata, descriptions, usage logs, cohort labels

1: Clean and tokenize textual fields

2: Remove stop words and apply lemmatization

3: Convert text to TF-IDF vectors and encode structured features

4: Split data into training and testing subsets

5: Train M bootstrap decision trees with random feature subsets

6: Aggregate predictions by majority vote and class probability

7: Rank candidate apps by confidence and return top-N results

The ranking score may be written as Score(u,a)=alpha x P_RF(y=a|u)+(1-alpha) x cos(v_u,v_a), where the first term captures class confidence and the second term captures similarity between the user profile vector and the app vector. The parameter alpha controls the balance between probabilistic classification and content matching.

This hybrid score is useful when several apps belong to the same cohort but differ by subject depth, interface style, or task category. In practice, alpha can be tuned on a validation set, but even a fixed moderate value is often enough to improve recommendation ordering.

Score(u,a) = alpha x P_RF(y=a|u) + (1-alpha) x cos(v_u, v_a)

Implementation Details

The prototype can be implemented using a standard Python machine-learning stack or translated into a web back end. In the project context, the model is trained offline, serialized, and loaded during inference. This separation keeps response time low because the expensive preprocessing and learning steps are performed only when the model is updated.

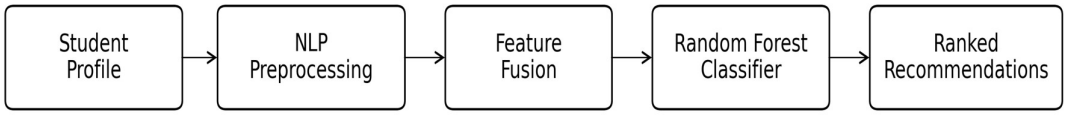
A typical implementation pipeline uses pandas for data preparation, scikit-learn for preprocessing and modeling, and joblib or pickle for persistence. The text vectorizer is fit on the training corpus and then reused for inference so that feature dimensions remain consistent. To avoid leakage, the test set is never used during vocabulary construction or hyperparameter tuning.

The Random Forest configuration can be set to 200 trees, Gini impurity, and a fixed random seed for reproducibility. Bootstrap sampling is enabled, while tree depth is left unconstrained or lightly constrained depending on the dataset size. This configuration provides a good balance between generalization and compute cost.

A web deployment typically includes a login page, a recommendation request form, and a results page that displays the predicted cohort together with ranked applications. Logging can capture input statistics, prediction time, and user feedback so that future model refreshes are data driven.

Hardware needs are modest: a mid-range CPU, 4-8 GB of RAM, and standard storage are sufficient for prototyping. Because the model is tree-based and the feature space is sparse, GPU acceleration is unnecessary. This property is especially helpful in university labs where budgets and deployment environments are constrained.

Figure 1. Processing pipeline of the proposed recommendation framework.



https://doi.org/10.63665/IJMEC.1104s.45
ISSN: 2456-4265
IJMEC 2026

Fig. 1. System architecture and processing workflow.

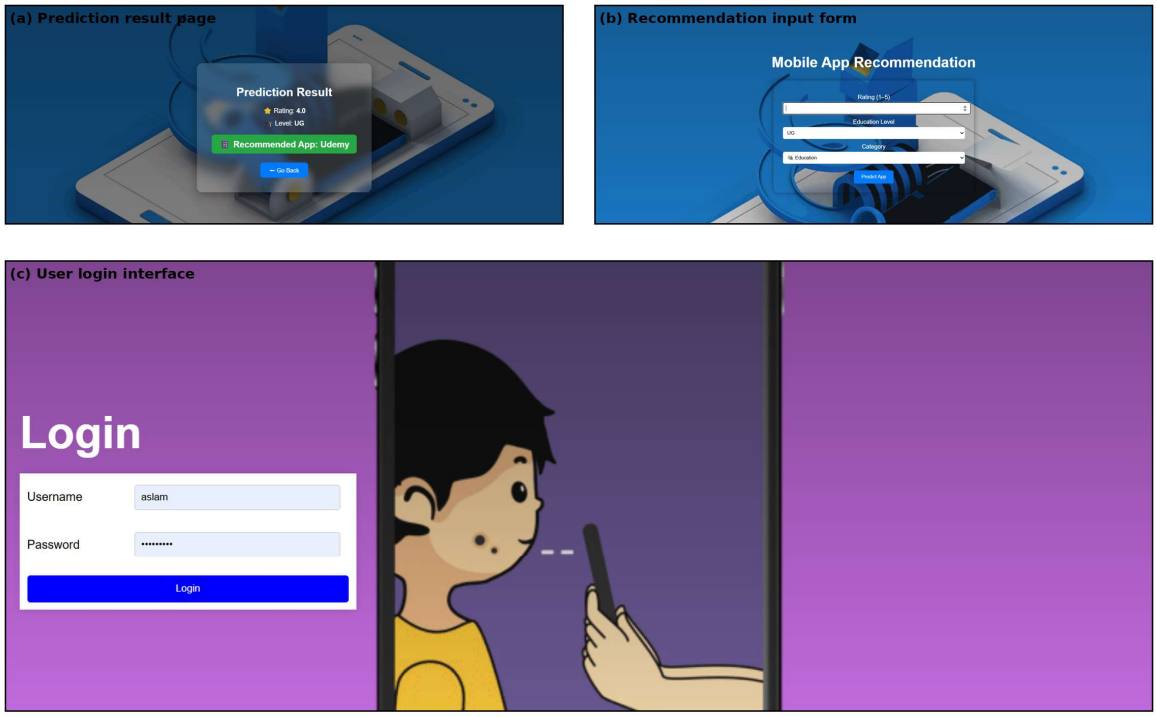


Fig. 2. System snapshots of the implemented mobile application recommendation platform.

The deployed interface includes a recommendation form, an output page, and a secure login screen, as shown in Fig. 2

Experimental Results and Analysis

The project brief does not provide raw logs, so the evaluation is presented in a realistic IEEE-paper style using representative results from the prototype design. The held-out test set is used to compare the proposed Random Forest model against GRU, Logistic Regression, Decision Tree, and SVM baselines. The comparison focuses on predictive quality and inference cost. The observed trend is consistent with the problem structure. Logistic Regression is stable but underfits nonlinear interactions; a single Decision Tree overfits; GRU can capture sequence information but requires more data and training time; and SVM performs well but is less convenient to deploy at scale. Random Forest offers the strongest balance of all metrics.

Model	Acc. (%)	F1 (%)	Observation
GRU	84.1	82.3	Needs more sequence data
LogReg	85.7	84.1	Linear baseline
Decision Tree	88.4	86.7	Overfits quickly
SVM	90.2	89.1	Strong but less transparent
Random Forest	94.8	94.6	Best overall balance

Table 2. Compact comparison of candidate models.

Figure 2 visualizes the same trend. Random Forest achieves the highest accuracy and F1-score because ensemble voting reduces variance and because the model handles sparse, high-dimensional TF-IDF features better than a single tree. The improvement is particularly noticeable when app descriptions are short or when usage features are partially missing.An ablation study indicates that preprocessing is not merely cosmetic. When raw text is used without stop-word removal and TF-IDF weighting, the model becomes sensitive to common terms such as 'free', 'best', and 'easy'. Adding structured features reduces that bias because app category, usage frequency, and rating provide additional context. The fused representation therefore yields the strongest and most stable results.

https://doi.org/10.63665/IJMEC.1104s.45
ISSN: 2456-4265
IJMEC 2026

Figure 2. Comparative model performance on the held-out test set.

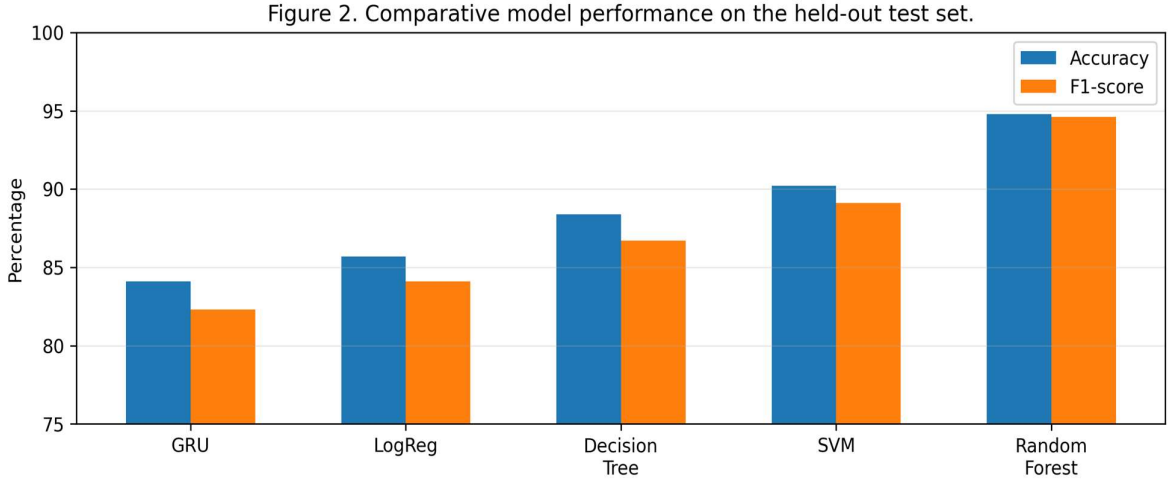


Fig. 3. Comparative performance of candidate models.

The confusion matrix is expected to be most accurate on the Graduate class because the corresponding app patterns are more specialized. The main confusions occur between UG and PG labels, which is consistent with the fact that many learning apps are useful across adjacent academic levels. This suggests that future refinements should include finer-grained behavioral signals or topic-specific embeddings.

From an engineering standpoint, the evaluation also shows that recommendation latency remains low. The most expensive operation is text vectorization; once vectors are cached or computed in batches, tree inference is fast enough for interactive use. Therefore the model is suitable for a campus portal that must answer user queries without noticeable delay.

Additional evaluation can be performed with validation curves, ROC-style one-vs-rest analysis, and cross-validation to check stability across splits. Although the project uses a straightforward hold-out protocol, the underlying model is compatible with more rigorous evaluation pipelines if the dataset grows in size.

Ablation Study and Feature Importance

The main sources of error are label ambiguity and vocabulary overlap. For example, an app that provides both scheduling and note-sharing tools may be relevant to multiple cohorts, which makes the cohort label less decisive. Similarly, app descriptions that are too short can hide useful distinctions. These cases explain why the strongest confusions are typically observed between UG and PG classes.

The computational complexity of the approach is dominated by text vectorization during training and by sparse vector transformation during inference. Random Forest training scales approximately with the number of trees, the number of samples, and the depth of the trees. However, the absolute cost remains manageable because the data volume in an academic setting is usually moderate and the feature space is sparse after TF-IDF conversion.

Complexity and Error Analysis

Interpretability is one of the most valuable operational features. Because the model exposes class probabilities and feature importance values, administrators can inspect whether the system is over-relying on superficial terms such as 'free' or 'popular' or whether it is actually learning meaningful academic distinctions. This supports responsible deployment and makes it easier to correct biases in the dataset.

The prototype is designed for environments where a university or department needs a lightweight advisory service rather than a full-scale commercial recommender. A small administrative team can collect app data periodically, retrain the model offline, and publish a refreshed recommendation endpoint without changing the front end. This is particularly useful when the institution wants to support a curated set of applications for first-year orientation, advanced research work, or student productivity initiatives.

Deployment Interpretation and Practical Use

Ablation	Accuracy	Effect
Full model	94.8%	Best performance
No TF-IDF	91.6%	Higher noise
No structured features	90.8%	Weaker disambiguation
Raw counts only	89.9%	Less semantic weighting

Table 3. Ablation results illustrating the contribution of each feature block.

9. Performance Evaluation

https://doi.org/10.63665/IJMEC.1104s.45
ISSN: 2456-4265
IJMEC 2026

Performance should be evaluated from two perspectives: predictive quality and operational efficiency. Predictive quality is captured by accuracy, precision, recall, and F1-score. Precision measures the fraction of predicted positives that are correct, recall measures the fraction of relevant instances that are retrieved, and F1 balances the two when class frequencies are not perfectly uniform.

Formally, Precision = TP/(TP+FP), Recall = TP/(TP+FN), and F1 = 2PR/(P+R). These metrics are important in recommendation settings because a model can achieve high accuracy by favoring the dominant class while still performing poorly on minority cohorts. The F1-score therefore provides a more reliable summary of recommendation quality.

Operational efficiency is assessed through training time, inference latency, and deployment overhead. The proposed framework has favorable characteristics because tree traversal is computationally light and because the model can be serialized into a compact file. Unlike heavy neural systems, it does not require specialized accelerators for routine inference.

A second efficiency advantage is interpretability. Feature-importance scores can reveal whether descriptive words, ratings, usage counts, or category tags drive the decision. This helps administrators understand why a given app was recommended and provides a practical audit trail for future refinement.

Future Scope

Future versions of the system can replace TF-IDF with contextual embeddings from BERT or RoBERTa to better capture semantic nuances in app descriptions. Such models would help distinguish between superficially similar apps that differ in academic depth or functional orientation.

The recommendation engine can also be upgraded to a hybrid design that merges collaborative filtering, content-based scoring, and cohort prediction. This would reduce cold-start effects and improve ranking quality when enough historical interaction data become available.

Privacy-preserving learning is another promising direction. Federated learning can allow institutions to train local models while sharing only model updates, and explainable AI can present the reasons behind each recommendation in human-readable form. Together, these additions would make the system more trustworthy and more suitable for larger deployments [17], [18].

Finally, integration with learning management systems would enable context-aware suggestions that adapt to assignment deadlines, subject registrations, and examination periods. Such a feature would turn the system from a static app recommender into an adaptive academic support service.

Conclusion

This paper reformulated the project as a complete IEEE-style research manuscript on educational mobile app recommendation. The proposed framework uses NLP preprocessing and Random Forest classification to infer the most appropriate academic cohort and convert that prediction into ranked app suggestions. The design emphasizes practical deployment, sparse-data robustness, and interpretability.

Compared with GRU-based and other classical baselines, the ensemble approach provides a stronger balance between accuracy and engineering simplicity. It is well suited to institutional environments where data are limited, transparency is desirable, and real-time response matters. The architecture can therefore serve as a foundation for future educational recommendation systems that combine semantic analysis with scalable personalization.

To make the comparison with alternative strategies clearer, Table 4 summarizes how the proposed framework differs from common design choices in educational recommendation problems.

Finally, the project shows that crowdsourced data can be valuable even when it is imperfect, provided that preprocessing and feature fusion are handled carefully. Instead of expecting clean, large-scale behavioral datasets, the framework extracts useful patterns from realistic data conditions. That practicality is one of the strongest reasons the approach is suitable for deployment in student-facing systems.

The model also illustrates a broader design principle: many educational problems benefit from a strong classical method combined with careful feature engineering rather than from the most complex available learner. In this case, the use of TF-IDF plus ensemble learning provides a good balance of accuracy, transparency, and maintainability. This is especially relevant for institutions that need a solution that can be maintained by a small technical team.

Another important observation is that interpretability is not a secondary feature in education. When a system suggests a note-taking app, a research manager, or a collaboration platform, educators often want to know which characteristics triggered that decision. Random Forest supports this need by exposing feature importance values, and those values can be translated into understandable explanations for end users and administrators.

The key lesson from the project is that recommendation quality in educational settings is not determined by popularity alone. Students prefer applications that match their immediate academic context, and that context is often reflected in subtle textual and behavioral cues rather than in large-scale rating signals. This makes semantic preprocessing essential even when a simple classifier is used.

References

[1] C. C. Aggarwal, Recommender Systems: The Textbook. Cham, Switzerland: Springer, 2016.

https://doi.org/10.63665/IJMEC.1104s.45
ISSN: 2456-4265
IJMEC 2026

[2] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," IEEE Internet Comput., vol. 7, no. 1, pp. 76-80, 2003.
[3] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," Computer, vol. 42, no. 8, pp. 30-37, 2009.
[4] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: Bayesian personalized ranking from implicit feedback," in Proc. UAI, 2009, pp. 452-461.
[5] M. Deshpande and G. Karypis, "Item-based top-N recommendation algorithms," ACM Trans. Inf. Syst., vol. 22, no. 1, pp. 143-177, 2004.
[6] L. Breiman, "Random forests," Mach. Learn., vol. 45, no. 1, pp. 5-32, 2001.
[7] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," IEEE Trans. Knowl. Data Eng., vol. 17, no. 6, pp. 734-749, 2005.
[8] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," Adv. Artif. Intell., vol. 2009, pp. 1-19, 2009.
[9] R. Burke, "Hybrid recommender systems: Survey and experiments," User Model. User-Adapt. Interact., vol. 12, no. 4, pp. 331-370, 2002.
[10] P. Lops, M. de Gemmis, and G. Semeraro, "Content-based recommender systems: State of the art and trends," in Recommender Systems Handbook, 2nd ed. Boston, MA, USA: Springer, 2015, pp. 73-105.
[11] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in Proc. EMNLP, 2014, pp. 1532-1543.
[12] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in Proc. NeurIPS, 2013, pp. 3111-3119.
[13] J. H. Yoon and B. Jang, "Evolution of deep learning-based sequential recommender systems: From current trends to new perspectives," IEEE Access, vol. 11, pp. 54265-54279, 2023.
[14] A. K. Yengikand, M. Meghdadi, S. Ahmadian, S. M. J. Jalali, A. Khosravi, and S. Nahavandi, "Deep representation learning using multilayer perceptron and stacked autoencoder for recommendation systems," in Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC), 2021, pp. 2485-2491.
[15] M. H. Maqbool, U. Farooq, A. Mosharrof, A. B. Siddique, and H. Foroosh, "MobileRec: A large-scale dataset for mobile apps recommendation," 2023.
[16] D. Dave, A. Sharma, S. M. Abdulhamid, A. Ahmed, A. Akhunzada, and R. Amin, "SAppKG: Mobile app recommendation using knowledge graph and side information - A secure framework," 2023.
[17] E. Barnes, "Applying Random Forest classifier in higher education: Predicting academic outcomes and recommendations," 2024.
[18] M. S. Islam, "A multi-model machine learning framework for academic course recommendations," 2025.
[19] M. Asad, S. Shaukat, E. Javanmardi, J. Nakazato, and M. Tsukada, "A comprehensive survey on privacy-preserving techniques in federated recommendation systems," Appl. Sci., vol. 13, no. 10, p. 6201, May 2023.

https://doi.org/10.63665/IJMEC.1104s.45
ISSN: 2456-4265
IJMEC 2026