**Comparing Collaborative filtering-based recommender and Hybrid (collaborative plus content) recommender system**

|  |  |  |
| --- | --- | --- |
| Hetal Prajapati (C0911591)  **2024F AML 3204 2**  **Social Media Analytics**  Lambton College | Joel Laureta (C0917501)  **2024F AML 3204 2**  **Social Media Analytics**  Lambton College | Ruchita Patel (C0913017)  **2024F AML 3204 2**  **Social Media Analytics**  Lambton College |
|  | Simon Shrestha (C0913111)  **2024F AML 3204 2**  **Social Media Analytics**  Lambton College |  |

# ABSTRACT

**This report presents a comparative analysis of two prominent approaches for building recommender systems: collaborative filtering-based recommender systems and hybrid recommender systems. The first system, a collaborative filtering-based recommender, employs Matrix Factorization, a powerful technique used to uncover latent patterns in user-item interactions. This system is built using the u.data dataset, which contains user ratings for movies. In contrast, the second system combines multiple data sources to create a hybrid recommender system, aiming to improve the recommendation quality by leveraging diverse information. The hybrid system integrates movie ratings from u.data, genre information from u.item, and sentiment analysis derived from IMDB reviews. A Neural Embedding layer from the PyTorch package is utilized to combine these different data sources cohesively, enhancing the system’s ability to generate more accurate and personalized recommendations. The report evaluates both recommender systems based on their ability to generate relevant recommendations using performance metrics such as accuracy, precision, and recall. The comparative analysis aims to provide a deeper understanding of the strengths and limitations of each approach, offering insights into the potential benefits of combining multiple data sources in recommender system design. Through this evaluation, the report aims to highlight which system better meets the challenges of modern recommendation tasks, especially in terms of handling diverse data and delivering personalized experiences to users.**

# Keywords

Collaborative Filtering, Neural Networks, Deep Learning, Matrix Factorization, Implicit Feedback

# INTRODUCTION

In the digital era, where information is abundant, recommender systems play a pivotal role in managing information overload. They have become essential to online platforms like e-commerce sites, streaming services, and social media, offering personalized suggestions that enhance user engagement and experience. Collaborative filtering, a widely used approach in recommender systems, predicts user preferences by analyzing past interactions, such as clicks or ratings.

Matrix Factorization (MF) is a leading collaborative filtering technique that represents users and items as vectors in a shared latent space. These vectors model interactions through operations like the inner product. Popularized during the Netflix Prize competition, MF has since become a cornerstone of latent factor modeling in recommendations. Efforts to improve MF include integrating it with neighborhood-based techniques, incorporating item-specific content, and extending it to versatile frameworks like factorization machines. However, MF often struggles to capture complex user-item interaction patterns due to the simplicity of its underlying interaction mechanism.

To overcome these challenges, hybrid recommender systems combine multiple data sources and methodologies, leveraging the strengths of different approaches. This project focuses on creating a Hybrid Recommender System that enhances collaborative filtering by incorporating contextual data such as movie genres, user ratings, and sentiment analysis. Specifically, the system uses:

* User-movie ratings from the *u.data* dataset,
* Movie genre details from *u.item*, and
* Sentiment analysis based on IMDb movie reviews.

By integrating these diverse data sources, the system aims to deliver more accurate and personalized recommendations. Matrix Factorization remains the core algorithm for latent feature modeling, while including genre and sentiment data adds deeper contextual understanding.

This hybrid approach not only improves the accuracy of predictions but also addresses challenges like sparse data and implicit feedback. It lays the foundation for more robust and adaptable recommendation systems capable of meeting users' varied and dynamic preferences.

# RELATED STUDY

**Summaries of Papers**

1. **Matrix Factorization Techniques for Recommender Systems (Koren, 2009)**  
   This paper discusses matrix factorization techniques for collaborative filtering, emphasizing latent factors derived from observed user-item interactions. The study demonstrates how matrix factorization models outperform classical nearest-neighbor techniques by accounting for implicit feedback, temporal dynamics, and confidence levels in predictions.
2. **Factorization Machines with Applications to Recommender Systems (Rendle, 2012)**  
   Rendle introduces factorization machines (FM), a general predictor that models interactions among features. FM integrates seamlessly into collaborative filtering and hybrid approaches, handling sparse and high-dimensional datasets efficiently. The study applies FM to recommender systems, outperforming traditional CF methods.
3. **Hybrid Recommender Systems: Survey and Experiments (Burke, 2002)**  
   This paper provides a comprehensive review of hybrid recommender systems, which combine collaborative and content-based methods. Burke explores multiple hybridization strategies, including weighted, mixed, and feature-combination approaches, highlighting their advantages in reducing data sparsity and cold-start issues.
4. **AutoRec: Autoencoders Meet Collaborative Filtering (Sedhain et al., 2015)**  
   The authors propose AutoRec, an autoencoder-based collaborative filtering method. User-based AutoRec reconstructs user-item interactions, effectively capturing latent patterns in sparse data. The study shows significant improvements over matrix factorization in terms of recommendation accuracy.
5. **Neural Collaborative Filtering (He et al., 2017)**  
   He et al. develop Neural Collaborative Filtering (NCF), a deep learning framework combining matrix factorization and multi-layer perceptrons (MLPs). NCF captures non-linear user-item interactions and demonstrates enhanced performance in implicit feedback settings.
6. **Deep Content-based Music Recommendation (Van den Oord et al., 2013)**  
   This paper integrates deep learning into collaborative filtering, utilizing convolutional neural networks (CNNs) to extract music features. These features are combined with CF techniques, enhancing recommendations by leveraging both user preferences and content-based attributes.
7. **Collaborative Denoising Auto-Encoders for Top-N Recommender Systems (Wu et al., 2016)**  
   Wu et al. extend denoising autoencoders (DAEs) for collaborative filtering by adding a user-node to the input layer. This method, called Collaborative Denoising Autoencoder (CDAE), effectively incorporates implicit feedback and demonstrates robustness to noisy input data.
8. **Wide & Deep Learning for App Recommendations (Cheng et al., 2016)**  
   Cheng et al. introduce Wide & Deep learning, which combines linear and deep learning models to recommend mobile apps. The wide component memorizes feature interactions, while the deep component generalizes through non-linear transformations, applicable to hybrid systems.
9. **Cross-Domain Collaborative Filtering with Factorization Machines (Loni et al., 2014)**  
   Loni et al. extend factorization machines to address cross-domain collaborative filtering challenges. The proposed method leverages latent relationships between domains, improving recommendations for users with limited interactions in a target domain.
10. **A Hybrid Recommender System with Deep Learning for Movie Recommendation (Zhang et al., 2016)**  
    This study combines deep learning with collaborative filtering to improve movie recommendations. It utilizes a hybrid framework integrating user-item embeddings from CF with features learned by deep neural networks, outperforming traditional CF and hybrid systems.

# METHODOLOGY

## INTRODUCTION

This experiment develops and compares two recommender systems using IMDb data. The first system employs collaborative filtering, while the second integrates a hybrid approach. The comparison aims to evaluate their strengths and limitations in delivering personalized recommendations.

**Building the Collaborative Filtering-Based Recommender System**

The collaborative filtering system uses **Matrix Factorization**, which decomposes the user-item interaction matrix into latent factors for predictions.

**Steps:**

1. **Data Preprocessing**: Construct a user-item interaction matrix from IMDb data.
2. **Matrix Factorization**: Apply Singular Value Decomposition (SVD) or similar techniques.
3. **Model Training**: Minimize prediction error using an optimization algorithm.
4. **Evaluation**: Use metrics like Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE).

**Building the Hybrid Recommender System**

The hybrid system combines collaborative filtering with content-based features, using the **PyTorch Embedding Layer** to generate dense vector representations for users and items.

**Steps:**

1. **Data Representation**: Encode user and item features using IMDb metadata.
2. **Embedding Layer**: Generate dense vector representations with PyTorch.
3. **Model Architecture**: Integrate embeddings with additional neural network layers.
4. **Training**: Train the model using a balanced loss function.
5. **Evaluation**: Compare its performance with the collaborative filtering system.

This methodology outlines the creation of two systems: a collaborative filtering model with matrix factorization and a hybrid model leveraging PyTorch embeddings. Using IMDb data and advanced techniques, the experiment aims to highlight the effectiveness of these approaches.

## HYBRID RECOMMENDER SYSTEM

A **hybrid recommender system** is a combination of **content-based** and **collaborative filtering** approaches, designed to capitalize on the strengths of both techniques and overcome their limitations. The idea is to combine the personalized recommendations of collaborative filtering with the item-specific recommendations of content-based methods to provide more accurate and diverse recommendations.

1. **Content-based Filtering**: This approach recommends items based on the features of the items themselves and the preferences of the user. For example, if a user has liked action movies in the past, the system will recommend other movies from the action genre or with similar characteristics, such as the same director or actors.
2. **Collaborative Filtering**: Collaborative filtering makes recommendations based on user-item interactions and patterns of behavior across many users. It assumes that if two users have rated items similarly in the past, they will have similar preferences in the future. This method can be **user-based** (recommending items based on similar users) or **item-based** (recommending items similar to those the user has liked before).

By combining these two approaches, a hybrid recommender system can address the challenges faced by each technique:

* **Content-based filtering** can struggle with *cold start problems* (where new items or users have little data) and lacks diversity, as it often recommends similar items.
* **Collaborative filtering** can suffer from *data sparsity*, especially in systems with a large number of items or users, where there may not be enough overlapping preferences to make reliable recommendations.

**Benefits of Hybrid Recommender Systems:**

1. **Improved Accuracy**: By combining both methods, the system can make more informed predictions and offer higher-quality recommendations.
2. **Enhanced Diversity**: It can recommend a wider variety of items, balancing the personalization of collaborative filtering with the content-specific recommendations from content-based filtering.
3. **Cold Start Problem Mitigation**: New users or items can be recommended based on either content features or by leveraging the behavior of similar users.

In a **neural hybrid approach**, advanced models such as the **Neural Embedding Layer** can be used to embed both user-item interactions and item features into a continuous vector space. This allows the system to learn complex relationships and improve the representation of users and items, leading to more accurate and personalized recommendations. For instance, by training embeddings for both content features (e.g., movie genre, director) and collaborative data (e.g., user-item interaction), the system can capture a more comprehensive understanding of both users and items.

Thus, hybrid recommender systems are powerful tools for delivering personalized, relevant, and diverse content, leveraging the strengths of both content-based and collaborative filtering methods.

A diagram of a machine learning model

Description automatically generated

**Figure 1: Hybrid Recommendation System**

## DATASET

The dataset includes information such as movie titles, genres, ratings, directors, and actors. This data serves as the foundation for building two distinct recommendation systems:

1. **Collaborative Filtering-Based Recommender**: This system uses the Matrix Factorization technique, which identifies latent features from the user-movie interaction matrix. It leverages user preferences and historical interactions (like ratings) to recommend movies based on the similarity between users or movies. The dataset helps in identifying relationships between users' movie preferences and predicting the most relevant movies for them.
2. **Hybrid Recommender System**: This system combines the collaborative filtering approach with additional features (like movie metadata) to improve recommendations. The neural embedding layer from PyTorch will be used to integrate various input types (such as user interactions and movie details) to create more personalized and accurate recommendations. By combining both collaborative filtering and content-based information, the hybrid system aims to overcome the limitations of using either approach alone.

The IMDb dataset is rich in movie-related metadata, which allows for diverse types of recommendations, such as genre-based, actor-based, or director-based, while also considering user behavior. This diversity of data types enables the creation of nuanced and effective recommendation systems.

## Matrix Factorization

Matrix Factorization is a powerful technique used for dimensionality reduction and collaborative filtering in recommendation systems. It decomposes a user-item interaction matrix into two lower-dimensional matrices, enabling the prediction of missing entries in the matrix.

**Matrix Factorization Algorithm:**

The goal is to approximate the interaction matrix R (of size M×N) using two smaller matrices:

* U: User latent feature matrix (M×K).
* V: Item latent feature matrix (N×K).

Here, K is the number of latent features.

The predicted interaction R^ij​ between a user i and an item j is calculated as:

**A black text with a white background

Description automatically generated**

**This minimizes the difference between the actual matrix R and the predicted matrix R^ expressed as:**

**A math equation with black text

Description automatically generated**

**Components of the Equation:**

1. Rij​: Actual interaction value between user i and item j (e.g., ratings).
2. R^ij: Predicted interaction value based on latent factors.
3. Ui: Latent vector for user i.
4. Vj ​: Latent vector for item j.
5. lambda: Regularization parameter to prevent overfitting.
6. ∣∣U∣∣^2: Regularization term for user latent features.
7. ∣∣V∣∣^2: Regularization term for item latent features.

**A diagram of a business

Description automatically generated**

**Figure 2: Neural matrix factorization model**

## NEURAL EMBEDDING LAYER

A neural embedding layer is a type of hidden layer in neural networks designed to map discrete categorical variables (like words, items, or entities) into a continuous, dense, low-dimensional vector space. This is particularly useful in domains like Natural Language Processing (NLP), recommendation systems, and other tasks involving categorical data.

Key Concepts:

1. Input Representation:

* The input is typically a high-dimensional, sparse vector (e.g., one-hot or multi-hot encoded).
* A one-hot vector represents a single categorical entity with a vector length equal to the size of the vocabulary, having a value of 1 at the entity's index and 0 elsewhere.

1. Embedding Matrix:

* The core of the embedding layer is a learnable matrix, where rows correspond to entities (e.g., words), and columns represent latent dimensions. For a vocabulary of size V and embedding dimension d, this matrix has dimensions V×d.
* Each row in this matrix is the vector representation (embedding) of a specific input entity.

1. Feature Extraction:

* When an input vector is passed into the embedding layer, the corresponding row of the embedding matrix is retrieved, producing a dense vector of latent features.
* This vector captures semantic or relational information learned during training.

1. Training:

* Embeddings are learned through backpropagation, typically as part of a larger neural network. The embeddings adjust to minimize the task-specific loss function, making them contextually relevant.

A diagram of a system

Description automatically generated

**Figure 3: Neural Embedding Layers**

## SENTIMENT SCORE

The sentiment score calculation is a method for evaluating the emotional tone of a text, which helps in understanding how positive, negative, or neutral a review is. In the code provided, we use TextBlob, a Python library, for sentiment analysis. TextBlob provides a simple API for natural language processing (NLP) tasks, including sentiment analysis. It uses a predefined lexicon and rule-based model to compute sentiment polarity, where a score between -1 and 1 is generated:

* 0 indicates negative sentiment,
* 1 indicates positive sentiment,
* 0.5 indicates neutral sentiment.

Sentiment Score Calculation Steps:

1. TextBlob Sentiment Analysis:

* The sentiment\_analysis function takes a text (review) as input and passes it to TextBlob.
* The TextBlob(text).sentiment.polarity function returns the polarity score of the review.
* This polarity score reflects the overall sentiment of the review text: positive or negative.

1. Calculating Average Sentiment for a Movie:

* The get\_average\_sentiment\_for\_movie function retrieves the reviews for a specific movie using the get\_imdb\_data(movie\_name) function (which is assumed to fetch movie data such as ratings and reviews).
* It then iterates over each review, cleans the review text (via the clean\_text(review) function to remove unnecessary characters or formatting), and calculates the sentiment score using sentiment\_analysis.
* These sentiment scores are appended to a list, and the average sentiment score is computed by taking the sum of all sentiment scores and dividing it by the number of reviews.
* The average sentiment score provides an overall sentiment for the movie, reflecting the general mood of the reviews.

1. Return Values:

* The function returns the movie's ratings (from get\_imdb\_data) and the average sentiment score. The average sentiment score gives a general idea of how reviewers feel about the movie, with higher positive values indicating more favorable reviews and negative values indicating more criticism.

This approach helps in understanding public sentiment towards a movie or product based on user-generated content.

## ACTIVATION FUNCTION

The ReLU (Rectified Linear Unit) activation function is used after the first fully connected layer (fc1).

Here's how it works:

* ReLU Activation: The ReLU activation function is applied to the output of the first fully connected layer (fc1) using torch.relu(x). ReLU transforms all negative values to zero while leaving positive values unchanged. This introduces non-linearity into the network, which is crucial for learning complex patterns in the data.

ReLU is a popular activation function because it:

* Improves convergence: It helps the model converge faster compared to other activation functions like sigmoid or tanh because it does not suffer from vanishing gradients for positive inputs.
* Simplicity: It is computationally efficient as it involves simple thresholding (if input > 0, output = input; otherwise, output = 0).
* Avoids the vanishing gradient problem: Unlike sigmoid or tanh, ReLU does not saturate for large input values, making it suitable for deep networks.

Thus, in this model, ReLU helps in capturing the non-linear relationships between the combined features (user embeddings, movie embeddings, genres, and sentiment scores), enabling the network to make more accurate predictions.

# EXPERIMENTS

In this section, we conduct experiments with the aim of answering the following research questions:

**RQ1** Do our proposed NCF methods outperform the stateof-the-art implicit collaborative filtering methods?

**RQ2** How does our proposed optimization framework (log loss with negative sampling) work for the recommendation task?

**RQ3** Are deeper layers of hidden units helpful for learning from user–item interaction data?

In what follows, we first present the experimental settings, followed by answering the above three research questions.

## EXPERIMENT DESIGN

**OMDB API (Open Movie Database)**:

* **Purpose**: Used to fetch movie-related data such as titles, genres, descriptions, ratings, and reviews.
* **Base URL**: http://www.omdbapi.com/
* **Key Features**:
  + Fetch movie details using movie titles or IMDb IDs.
  + Provides data on movies, series, and episodes.
  + Requires an API key for access (can be obtained by signing up on the OMDB website).
* **Example Request**:  
  <http://www.omdbapi.com/?t=Inception&apikey=your_api_key>

**VaderSentiment API**:

* **Purpose**: Used to analyze the sentiment of text, especially reviews or descriptions. It outputs a sentiment score (positive, neutral, or negative).
* **Installation**: pip install vaderSentiment
* **Key Features**:
  + Sentiment scores for text, such as movie reviews or descriptions.
  + Outputs scores for positive, neutral, and negative sentiments.

**PyTorch API**:

* **Purpose**: Used for building and training neural networks, such as the hybrid recommender system with an embedding layer.
* **Installation**: pip install torch
* **Key Features**:
  + Deep learning framework for model creation, training, and optimization.
  + Supports automatic differentiation, GPU acceleration, and deployment.

## DATASET PREPARATION

To prepare the datasets for the collaborative filtering and hybrid recommender systems, the following steps were taken:

1. Data Collection:

* The IMDb dataset was scraped using the OMDB API, which provides movie details, including ratings, genres, and other relevant information.

1. Data Cleaning:

* Missing or incomplete entries were handled by either removing or filling in data where possible.
* Data types were standardized to ensure consistency across all records.

1. Feature Selection:

* Relevant features, such as movie titles, genres, ratings, and number of ratings, were selected for building the recommender systems.

1. Normalization:

* Ratings were normalized to a scale of 0 to 1 to ensure consistency for matrix factorization techniques and neural embedding layers.

1. Exploratory Data Analysis (EDA):

* An analysis of the dataset's ratings revealed the following:
  + Average IMDb Rating: 7.17
  + Highest IMDb Rating: 8.10
  + Lowest IMDb Rating: 6.20

1. Dataset Splitting:

* The dataset was split into training and test sets. The training set was used for model training, while the test set was used for evaluating the performance of both collaborative filtering and hybrid recommender systems.

1. Handling Imbalances:

* The dataset was checked for skewness in the number of ratings per movie, and measures were taken to ensure that movies with fewer ratings did not distort the model's predictions.

These steps ensured that the datasets used for training and evaluating the recommender systems were clean, consistent, and representative of the movie rating domain.

## EVALUATION METRICS

In this experiment, the following evaluation metrics are used to assess the performance of the recommender system: **Root Mean Squared Error (RMSE)**, **Mean Absolute Error (MAE)**, **Precision**, **Recall**, and **F1-Score**.

**1. Root Mean Squared Error (RMSE)**

RMSE measures the square root of the average squared differences between predicted and actual values. It gives higher weight to large errors, making it sensitive to outliers.

* **Formula**:

**A black square with a white background

Description automatically generated with medium confidence**

* **Usage**: RMSE is used to evaluate the overall accuracy of the predictions for continuous data.

**2. Mean Absolute Error (MAE)**

MAE measures the average magnitude of absolute differences between predicted and actual values. It is a simpler metric that provides the average error without squaring the differences, making it less sensitive to large errors than RMSE.

* **Formula**:

A black and white text

Description automatically generated

* **Usage**: MAE is used to assess the average prediction error in continuous data.

**3. Precision**

Precision indicates the proportion of true positive predictions out of all positive predictions. It answers the question: *Out of all the items predicted as positive, how many were actually positive?*

* **Formula**:

**A close-up of a sign

Description automatically generated**

* **Usage**: Precision is important when the cost of false positives is high.

**4. Recall**

Recall indicates the proportion of true positive predictions out of all actual positives. It answers the question: *Out of all the actual positive items, how many did the model correctly identify as positive?*

* **Formula**:

**A black text on a white background

Description automatically generated**

* **Usage**: Recall is critical when the cost of false negatives is high.

**5. F1-Score**

The F1-Score is the harmonic mean of Precision and Recall, providing a balanced measure. It is especially useful when you need to balance the trade-off between Precision and Recall.

* **Formula**:

A close-up of black text

Description automatically generated

* **Usage**: The F1-Score is useful when both Precision and Recall are important, and there is a need to balance between the two.

**Result:**

RMSE: 0.9439

MAE: 0.7572

Precision: 0.9000

Recall: 0.8436

F1-Score: 0.8708

In this experiment:

* **RMSE** (0.9439) and **MAE** (0.7572) show how well the model's predictions approximate the actual values for continuous ratings.
* **Precision** (0.9000), **Recall** (0.8436), and **F1-Score** (0.8708) indicate how well the model performs in correctly identifying positive sentiment based on a threshold rating (3). The high F1-Score reflects a good balance between precision and recall.

## RESULT AND ANALYSIS

### HYPERPARAMETER SELECTION

Hyperparameters, like the latent vector size, determine how well a collaborative filtering model can capture user-item relationships. The latent vector size represents the number of features used to encode users and items. A larger latent size can capture more complex patterns but may also lead to overfitting, while a smaller size might not capture enough information, resulting in higher RMSE.

**Matrix Factorization:**

* **Small Latent Size:** Might underfit, leading to higher RMSE.
* **Large Latent Size:** Can overfit, reducing RMSE on training data but increasing it on test data.

A graph with a line

Description automatically generated

**Figure 4: RMSE vs Latent Vector Size for Matrix Factorization**

**Embedded Layer Neural Network:**

**A graph with a line

Description automatically generated**

**Figure 5: RMSE vs Latent Vector Size for Embedded Neural Layer**

**RMSE** is used to measure prediction accuracy, with lower values indicating better performance.

By running experiments with latent vector sizes (e.g., 20, 30, 40, 50) and calculating RMSE for each, you can identify the optimal size for each model. Visualizing RMSE vs. latent size helps in selecting the best configuration.

The resulting line plots will show how RMSE varies with latent vector size for both the Matrix Factorization and Neural Network models, helping to find the best trade-off between model complexity and prediction accuracy.

### RESULTS OF RECOMMENDATION MODELS

In this section, we analyze and compare the performance metrics of the two recommendation models: the **Collaborative Filtering-based Recommender** (using Matrix Factorization) and the **Hybrid Recommender** (utilizing the Neural Embedding layer from PyTorch). The models' outputs are evaluated using the following metrics:

1. **RMSE (Root Mean Square Error):** Measures the average deviation between predicted and actual ratings. Lower RMSE indicates better accuracy.
2. **MAE (Mean Absolute Error):** Represents the mean of absolute errors between predicted and true ratings. Like RMSE, lower values denote better performance.
3. **Precision:** The proportion of recommended items in the top-N list that are relevant. Higher precision means more accurate recommendations.
4. **Recall:** The fraction of relevant items successfully retrieved in the top-N list. Higher recall suggests better coverage of relevant items.
5. **F1-Score:** A harmonic mean of precision and recall, balancing the trade-off between the two.

We evaluate Precision, Recall, and F1-Score for **Top-10** and **Top-20 recommendations**, providing insights into how each model performs at varying recommendation depths.

A graph of a bar chart

Description automatically generated with medium confidence

**Figure 6: Top-10 recommendations for Collaborative Recommendation system**

A graph of different colored bars

Description automatically generated with medium confidence

**Figure 7: Top-20 recommendations for Collaborative Recommendation system**

A graph showing a bar chart

Description automatically generated with medium confidence

**Figure 8: Top-10 and Top-20 recommendations for Hybrid Recommendation system**

# CONCLUSION AND FUTURE WORK

This report presents a detailed comparative analysis of two widely used recommender system approaches: collaborative filtering-based and hybrid recommender systems. The collaborative filtering-based system, utilizing Matrix Factorization, demonstrated its ability to uncover latent patterns in user-item interactions, showing strong performance in making accurate predictions using user ratings. However, it faces limitations in handling data sparsity and cold start problems. On the other hand, the hybrid recommender system, which integrates multiple data sources such as movie ratings, genre information, and sentiment analysis, successfully enhanced the recommendation quality. By leveraging a Neural Embedding layer, the hybrid system was able to combine these diverse data types, resulting in more personalized and relevant recommendations. The evaluation based on performance metrics like accuracy, precision, and recall revealed that the hybrid system performed better in handling diverse data and delivering tailored recommendations.

Future work could focus on the following areas:

1. **Incorporating More Data Sources**: Adding user demographics or social media data could further refine the systems' ability to capture user preferences.
2. **Advanced Hybrid Techniques**: Experimenting with different hybridization strategies, like ensemble methods or deep learning, could improve recommendation quality.
3. **Real-time Adaptation**: Integrating real-time user feedback could enhance the relevance of recommendations as preferences evolve.
4. **Explainability**: Developing techniques to explain recommendations could increase user trust and satisfaction.
5. **Scalability**: Testing with larger datasets would help assess the systems' performance and scalability in real-world environments.

These improvements could optimize the models for complex, real-world recommendation tasks.

## **Acknowledgment**

We would like to thank our instructor for their invaluable

assistance and direction during this study. In addition, we

would like to express our heartfelt gratitude to the team

members who assisted with data collecting and analysis.

# REFERENCES

1. Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction, 12*(4), 331–370. https://doi.org/10.1023/A:1021240730564
2. Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., ... & Anil, R. (2016). Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems* (pp. 7–10). https://doi.org/10.1145/2988450.2988454
3. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web* (pp. 173–182). https://doi.org/10.1145/3038912.3052569
4. Koren, Y. (2009). Matrix factorization techniques for recommender systems. *IEEE Computer, 42*(8), 30–37. https://doi.org/10.1109/MC.2009.263
5. Loni, B., Shi, Y., Larson, M., & Hanjalic, A. (2014). Cross-domain collaborative filtering with factorization machines. In *European Conference on Information Retrieval* (pp. 656–661). https://doi.org/10.1007/978-3-319-06028-6\_58
6. Rendle, S. (2012). Factorization machines with applications to recommender systems. In *Proceedings of the 2012 IEEE 12th International Conference on Data Mining* (pp. 995–1000). https://doi.org/10.1109/ICDM.2010.127
7. Sedhain, S., Menon, A. K., Sanner, S., & Xie, L. (2015). AutoRec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web* (pp. 111–112). https://doi.org/10.1145/2740908.2742726
8. Van den Oord, A., Dieleman, S., & Schrauwen, B. (2013). Deep content-based music recommendation. In *Proceedings of the 27th International Conference on Neural Information Processing Systems* (pp. 2643–2651).
9. Wu, Y., DuBois, C., Zheng, A. X., & Ester, M. (2016). Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining* (pp. 153–162). https://doi.org/10.1145/2835776.2835837
10. Zhang, S., Yao, L., Sun, A., & Tay, Y. (2016). Deep learning-based recommender system: A survey and new perspectives. *ACM Computing Surveys, 52*(1), 1–38. https://doi.org/10.1145/3285029