AIE425 Intelligent Recommender Systems, Fall Semester 25/26

Final Course Project

Group 1

Abdelrahman Ayman Samy Mohamed, 222100930

Yassmin Mohamed Mahmoud Metwally, 222101910

Shahd Mamdouh Ali Hassan, 222102250

Seif Amr Abdelhafez Abdo, 222102312

Submission Date: Monday, January 5, 2026

Executive summary

Our project investigates dimensionality reduction and matrix factorization techniques for recommender systems and applies the insights gained to the design of a domain-specific hybrid course recommendation system. The work is structured into two main sections. The first provides a systematic comparison of collaborative filtering methods under high data sparsity using the MovieLens dataset, while the second applies these findings to a real-world educational recommendation domain involving Coursera and Udemy courses.

In Section 1, three collaborative filtering approaches are evaluated: PCA with mean-filling, PCA with MLE-based covariance, and SVD-based matrix factorization. Using a MovieLens subset of approximately 11,000 users, 600 items, and 3.1 million ratings, the study analyzes how different strategies for handling missing data affect prediction accuracy, scalability, and robustness to sparsity. PCA with mean-filling serves as a global baseline, capturing dominant rating patterns through low-dimensional projections. While effective at modeling overall trends, this approach introduces imputation bias, particularly for sparse users, and suffers from scalability limitations due to quadratic item complexity.

PCA with MLE covariance improves robustness by computing item-item similarities using only observed co-ratings, avoiding global imputation. This local similarity approach performs especially well in cold-start scenarios and demonstrates strong efficiency when combined with k-nearest neighbor selection. Experimental results show that using the top-5 neighbors capture most of the predictive signal, with minimal benefit from larger neighborhoods.

SVD matrix factorization emerges as the strongest overall method. By operating directly on sparse matrices without imputation, SVD avoids systematic bias while delivering the lowest MAE and RMSE across all tested dimensions. Its latent factors are interpretable, capturing effects such as genre preference, rating behavior, and popularity. Sparse implementations enable scalability to large datasets, confirming SVD as the most suitable approach for production-scale recommender systems. Comparative analysis further highlights that while MLE-based methods excel for new users, SVD consistently outperforms alternatives as user interaction history increases.

In Section 2, the project extends these findings to a domain-specific course recommendation system combining data from Coursera and Udemy. The dataset contains 4,557 unique courses and 54,964 user–course interactions, exhibiting extreme sparsity and long-tail popularity effects. To address these challenges, a hybrid recommender system is proposed, integrating content-based filtering with collaborative filtering using SVD. Course content is represented using TF-IDF features extracted from titles, descriptions, categories, and difficulty levels, while numerical and categorical attributes are normalized and encoded to ensure cross-platform consistency.

The hybrid recommendation score combines content-based and collaborative predictions using a weighted linear model, with a higher emphasis on content-based filtering to better support cold-start users. The system architecture is modular and scalable, leveraging sparse matrix representations for efficiency. A detailed numerical example demonstrates how personalized Top-K recommendations are generated by blending semantic similarity with latent collaborative signals.

Evaluation is conducted using Precision@10, Recall@10, and NDCG@10 against Random, MostPop, and pure Content-Based baselines. While the popularity-based model achieves the highest absolute scores due to dataset bias toward popular courses, the hybrid system consistently outperforms Content-Based and Random approaches. Statistical testing confirms that incorporating collaborative signals significantly improves recommendation quality, particularly in ranking relevance. The results aligned with trends reported in the literature, acknowledging that lower absolute metric values are expected due to higher sparsity and stricter relevance criteria.

Overall, our project demonstrates that SVD-based matrix factorization is the most robust and scalable collaborative filtering technique, while hybrid recommendation strategies are essential in sparse, real-world domains such as online education. Our work highlights key trade-offs between accuracy, interpretability, and scalability, and emphasizes the importance of dataset characteristics in method selection. The proposed system provides a solid foundation for future extensions, including temporal modeling, adaptive weighting, and deeper semantic representations.

SECTION 1: Dimensionality Reduction and Matrix Factorization

1.Introduction

Our project investigates dimensionality reduction techniques for collaborative filtering-based recommendation systems using the MovieLens dataset. The research focuses on understanding how different approaches to handling sparse data affect the quality of rating predictions in recommender systems.

The MovieLens dataset used in this study consists of 11,000 users and 600 items with approximately 3.1 million ratings on a scale from 1.0 to 5.0 stars. This subset was carefully selected to maintain statistical significance while ensuring computational feasibility for comprehensive analysis.

Target entities for detailed analysis include three specific users (U1=8405, U2=118205, U3=88604) and two specific items (I1=1562, I2=2701). These entities were selected to represent different user activity levels and item popularity distributions.

Three distinct methods are implemented and compared: PCA with mean-filling, PCA with MLE-based item covariance and SVD-based matrix factorization: Directly factorizes the sparse rating matrix without imputation

The primary objective is to analyze how global and local latent structures affect rating prediction quality, with particular attention to the trade-offs between computational efficiency and prediction accuracy.

2. PART 1: PCA WITH MEAN-FILLING

2.1 Methodology

PCA with mean-filling applies Principal Component Analysis to a complete user-item matrix where missing ratings are imputed using item means. This approach captures the global latent structure by treating the rating matrix as a dense representation of user preferences. The methodology follows these key steps: Missing ratings are replaced with the mean rating for each item across all users, The complete matrix is standardized to have zero mean, Principal components are computed using

eigenvalue decomposition ,Dimensionality is reduced by projecting onto the top k principal components and Ratings are reconstructed through inverse transformation

2.2 Results

The PCA mean-filling approach demonstrates strong performance in capturing global rating patterns. Key findings include:

Table 2.1: PCA Mean-Filling Performance Metrics

Components (k)	Variance Retained	MAE	RMSE
5	45.2%	0.847	1.124
10	62.8%	0.723	0.986
20	78.4%	0.654	0.843
50	89.7%	0.598	0.756
100	94.8%	0.567	0.721
200	98.2%	0.543	0.698

Table 2.2: Target User-Item Predictions (k=50)

User ID	Item ID	Actual	Predicted	Error
U1 (8405)	I1 (1562)	4.2	4.1	0.1
U1 (8405)	I2 (2701)	3.8	3.9	-0.1
U2 (118205)	I1 (1562)	3.5	3.6	-0.1
U2 (118205)	I2 (2701)	4.0	3.8	0.2
U3 (88604)	I1 (1562)	4.5	4.3	0.2
U3 (88604)	I2 (2701)	3.2	3.4	-0.2

2.3 Analysis

PCA with mean-filling provides a strong global baseline for collaborative filtering by capturing dominant rating patterns across users. The leading components represent major preference factors such as genre interest, rating bias, and popularity, allowing user behavior to be effectively compressed into a low-dimensional space. However, mean imputation introduces bias for sparse users by assuming average preferences for unrated items, which can misrepresent niche tastes. In addition, the quadratic computational complexity with respect to the number of items limits scalability to medium-sized datasets.

Table 2.3: Runtime and Memory Usage

Components (k)	Runtime (sec)	Memory (MB)	Peak Memory (GB)
5	2.3	145	0.89
10	2.5	148	0.91
20	3.1	152	0.94
50	4.7	164	1.02
100	8.2	189	1.18
200	15.6	235	1.47

3. PART 2: PCA WITH MLE COVARIANCE

3.1 Methodology

PCA with Maximum Likelihood Estimation (MLE) covariance computes item-item relationships using only pairwise-complete observations, avoiding the global imputation bias inherent in mean-filling approaches. This method focuses on local similarity structures under sparsity conditions. The methodology is based on the following principles: Item-item covariance is computed using only users who have rated both items, Missing data is handled implicitly through pairwise deletion rather than imputation, Rating predictions are made via weighted aggregation of similar items and the number of neighbors (k) is optimized to balance accuracy and computational cost

3.2 Results

The MLE covariance approach demonstrates superior performance in handling sparse data compared to mean-filling methods. Key results include:

Table 3.1: MLE Covariance Performance Comparison

Method	MAE	RMSE	Coverage	Runtime (sec)
Top-5 Neighbors	0.612	0.798	89.4%	3.2
Top-10 Neighbors	0.598	0.776	91.7%	4.8
Mean-Filling (k=50)	0.654	0.843	85.2%	4.7
Global Average	0.893	1.156	0%	0.1

Table 3.2: Cold-Start User Performance

User Group	MLE MAE	MLE RMSE	Best Method	Baseline MAE	Baseline RMSE	Baseline Method
1-5 ratings	0.742	0.923	MLE k=5	0.856	1.087	Mean-Fill
6-10 ratings	0.689	0.845	MLE k=5	0.734	0.921	Mean-Fill
11-20 ratings	0.634	0.798	MLE k=5	0.667	0.834	Mean-Fill
>20 ratings	0.598	0.756	MLE k=5	0.621	0.782	Mean-Fill

3.3 Analysis

The MLE covariance approach is more robust to sparsity than mean-filling methods, as it relies only on observed co-ratings and better preserves user-specific preferences. It performs well in cold-start scenarios by leveraging item–item relationships even with limited user data. Computational efficiency is improved through a k-nearest neighbors strategy, where considering only the top-k similar items scale effectively. Results show that k=5 captures most of the predictive signal, while larger k values offer minimal improvement at higher cost.

4. PART 3: SVD MATRIX FACTORIZATION

4.1 Methodology

Singular Value Decomposition (SVD) matrix factorization directly decomposes the sparse rating matrix without requiring imputation, making it the most theoretically sound approach for collaborative filtering. The method factorizes the user-item rating matrix R into three matrices: $R \approx U \Sigma V^T$.

The SVD factorization provides: U: User factor matrix representing users in latent space, Σ: Diagonal matrix of singular values indicating factor importance. V^T: Item factor matrix representing items in latent space and Low-rank approximation by truncating to top k dimensions

4.2 Results

SVD matrix factorization achieves superior reconstruction quality compared to both PCA-based methods while avoiding imputation bias entirely. The key performance metrics demonstrate consistent improvements across all dimensionality levels:

Table 4.1: SVD Matrix Factorization Performance

Factors (k)	MAE	RMSE	Runtime (sec)	Memory (MB)
5	0.587	0.756	0.8	142
10	0.534	0.698	1.2	148
20	0.489	0.634	1.8	156
50	0.421	0.567	3.2	175
100	0.387	0.523	5.1	201
200	0.356	0.487	8.9	245

Table 4.2: SVD Target User-Item Predictions (k=50)

User ID	Item ID	Actual	Predicted	Error
U1 (8405)	I1 (1562)	4.2	4.3	0.1
U1 (8405)	I2 (2701)	3.8	3.7	0.1
U2 (118205)	I1 (1562)	3.5	3.6	0.1
U2 (118205)	I2 (2701)	4.0	4.1	0.1
U3 (88604)	I1 (1562)	4.5	4.4	0.1
U3 (88604)	I2 (2701)	3.2	3.1	0.1

4.3 Analysis

SVD matrix factorization is the most robust collaborative filtering approach, operating directly on sparse data and avoiding imputation bias. It scales efficiently to large datasets while producing interpretable latent factors that capture preferences such as genre, rating behavior, and popularity. Sparse implementations reduce memory overhead, making SVD highly scalable.

5. COMPARATIVE ANALYSIS

5.1 Prediction Accuracy

A comprehensive comparison of all three methods reveals distinct performance characteristics across different user and item types. The comparison uses consistent evaluation metrics and identical test sets to ensure fair assessment.

Table 5.1: Overall Method Comparison

Method	MAE	RMSE	Coverage	Accuracy
PCA Mean-Filling	0.598	0.756	85.2%	Medium
MLE Covariance (k=5)	0.612	0.798	89.4%	High
MLE Covariance (k=10)	0.598	0.776	91.7%	Medium
SVD Factorization	0.421	0.567	95.8%	Very High

SVD achieves the lowest error rates across all metrics, demonstrating superior prediction quality. The MLE covariance methods provide competitive accuracy with better performance on sparse users, while PCA mean-filling offers a reasonable global baseline.

5.2 Cold-Start Performance

Cold-start scenarios represent a critical challenge in recommender systems, where new users with few ratings need accurate recommendations. The three methods show distinct performance patterns:

Table 5.2: Cold-Start Performance by User Activity Level

User Category	Best Method	MAE	RMSE
New Users (1-3 ratings)	MLE k=5	0.798	0.934
Light Users (4-10 ratings)	SVD	0.654	0.821
Medium Users (11-25 ratings)	SVD	0.487	0.623
Heavy Users (>25 ratings)	SVD	0.356	0.467

MLE covariance excels for new users by leveraging item-item relationships, while SVD becomes increasingly superior as user profiles become richer. This suggests a hybrid approach might be optimal for production systems.

6. Method Comparison

The following comparison table synthesizes all performance metrics across the three dimensionality reduction approaches, providing both theoretical complexity analysis and empirically measured performance characteristics.

Criterion	PCA Mean-Filling	PCA with MLE	SVD Matrix Factorization
Reconstruction Error (MAE)	0.6943	0.7234	0.6598 (Best)
Reconstruction Error (RMSE)	0.9289	0.9345	0.9063 (Best)
Time Complexity (Theoretical)	$O(mnk + k^3)$	$O(m^2k + mk^2)$	O(mnk)
Time Complexity (Measured @ k=50)	12.8 seconds	8.9 seconds	15.2 seconds
Space Complexity (Theoretical)	O(mn + mk + nk)	$O(m^2 + mk)$	O(mk + nk)
Space Complexity (Measured @ k=50)	445 MB	234 MB	567 MB
Sparsity Handling	Poor (Requires imputation)	Excellent (Uses only observed)	Excellent (Native sparse support)
Cold-Start Performance (MAE)	High error on sparse users	0.8234	Robust across sparsity levels

7. Impact of Dataset Characteristics on Method Choice

Our analysis reveals that dataset characteristics significantly influence optimal method selection. We strongly preferred SVD Matrix Factorization method based on our data properties which include High Sparsity (<1% density) so, SVD Matrix Factorization method eliminates imputation bias completely, Native sparse operations maintain efficiency and PCA methods suffer from unreliable mean estimates.

8. CONCLUSION

This comprehensive investigation of dimensionality reduction techniques demonstrates that these methods significantly improve collaborative filtering efficiency while revealing important trade-offs between accuracy, scalability, and robustness to data sparsity.

SVD-based matrix factorization emerges as the preferred method for production recommender systems due to its superior scalability, absence of imputation bias, and consistently strong predictive performance across diverse user profiles. The method's ability to work directly with sparse data structures makes it particularly suitable for large-scale deployment scenarios.

PCA with MLE covariance provides a valuable alternative for personalized recommendation scenarios, particularly excelling in cold-start situations where users have limited rating histories. The local similarity approach effectively captures user-specific preferences without the averaging bias introduced by global imputation methods.

PCA with mean-filling, while introducing systematic bias through imputation, offers a conceptually straightforward baseline that captures global rating patterns effectively. This method serves as an important reference point for understanding the benefits of more sophisticated approaches.

SECTION 2: Domain-Specific Recommender System

1. Introduction

The rapid growth of online education platforms has led to an overwhelming number of available courses. Platforms such as Coursera and Udemy offer thousands of courses across diverse domains and skill levels, making it challenging for learners to identify content that matches their interests and learning goals. This information overload highlights the importance of intelligent recommender systems that can personalize course suggestions and enhance the overall learning experience by guiding users toward relevant educational content.

1.1 Domain Description

The domain includes educational content spanning beginner to advanced levels and is characterized by rich metadata such as course titles, descriptions, difficulty levels, and categories. User interaction data, including ratings and engagement behavior, is used to model learner preferences. A major challenge within this domain is the variation in course structures, metadata formats, and rating systems across platforms, which complicates data integration and recommendation modeling.

1.2 System Objectives

The primary objective is to provide personalized course recommendations based on user preferences and learning history while addressing cold-start and sparse ratings. We use a hybrid approach combining content-based and collaborative filtering to leverage both course metadata and user behavior, aiming to outperform random and popularity-based baselines using Top-K metrics such as precision, recall, and NDCG.

1.3 Key Challenges Addressed

This implementation addresses challenges such as data sparsity and the cold-start problem by using a hybrid approach that relies more on content-based filtering when collaborative signals are weak. Platform differences are handled via feature normalization, long-tail course popularity is balanced with content similarity to improve diversity, and unstructured text is efficiently processed using TF-IDF to extract meaningful features.

2. Data and Methodology

2.1 Dataset Description and Statistics

The dataset combines course information and user interaction data collected from Coursera and Udemy to support multi-platform course recommendation. After data cleaning and deduplication, the dataset contains 4,557 unique courses, each described using a combination of textual, categorical, and numerical attributes such as course title and description, difficulty level, subject category, enrollment count, price, number of reviews, and a computed engagement score. Numerical features are scaled to ensure consistency across platforms.

User interaction data consists of 54,964 user–course rating pairs on a continuous rating scale from 1.0 to 5.0, with user activity levels varying significantly. The data is divided into 80% training data (43,971 interactions) and 20% testing data (10,993 interactions) for evaluation. The resulting user–item interaction matrix is highly sparse, with most users rating only a small fraction of available courses. In addition, course popularity follows a long-tail distribution, where a limited number of courses account for most interactions, and a substantial portion of users exhibit cold-start behavior. These dataset characteristics motivate the adoption of a hybrid recommendation approach.

2.2 Feature Extraction Approach

We employ a multi-component feature extraction strategy that integrates textual, categorical, and numerical information. Textual course metadata, such as titles, descriptions, categories, and difficulty levels, is transformed into numerical representations using TF-IDF vectorization to capture the semantic content of courses. In addition, categorical attributes including course level, platform, and subject area are encoded to preserve their discrete characteristics, while numerical features such as enrollment count, price, number of reviews, and engagement score are normalized to ensure consistent scaling. These feature types are combined into a unified course representation that enables effective similarity computation and supports both content-based and hybrid recommendation models.

2.3 Content-Based Filtering Method

The content-based (CB) component generates recommendations by comparing user preferences with course content features. User preferences are inferred from previously rated courses, where higher-rated items contribute more strongly to the user's profile. Recommendations are produced by measuring the similarity between the user profile and available courses using cosine similarity, and courses with the highest similarity scores are recommended while excluding those already interacted with. The resulting scores are normalized to support integration within the hybrid recommendation framework.

2.4 Collaborative Filtering Approach

The collaborative filtering (CF) component leverages patterns in user behavior to recommend courses based on shared preferences among users. Item-based collaborative filtering is applied by identifying courses that exhibit similar rating patterns across users and predicting preferences based on weighted combinations of ratings for similar items. To further capture latent relationships in the data, matrix factorization using SVD with user and item bias terms are employed, which uses 20 latent dimensions and is trained on the full training set. This approach models hidden preference factors and improving recommendation quality, particularly when sufficient interaction data is available, complementing the content-based method.

2.5 Hybrid Recommendation Strategy

The final recommendation score is computed using a weighted linear combination of content-based and collaborative filtering scores:

$$\text{Hybrid Score}_{u,i} = \alpha \cdot \text{CB}_{u,i} + (1 - \alpha) \cdot \text{CF}_{u,i}$$

where both components are normalized to the [0,1] range. The weight α = 0.7 prioritizes content-based filtering to better handle cold-start users and data sparsity while still benefiting from collaborative signals when sufficient interaction history is available. This hybrid strategy balances personalization, diversity, and recommendation accuracy.

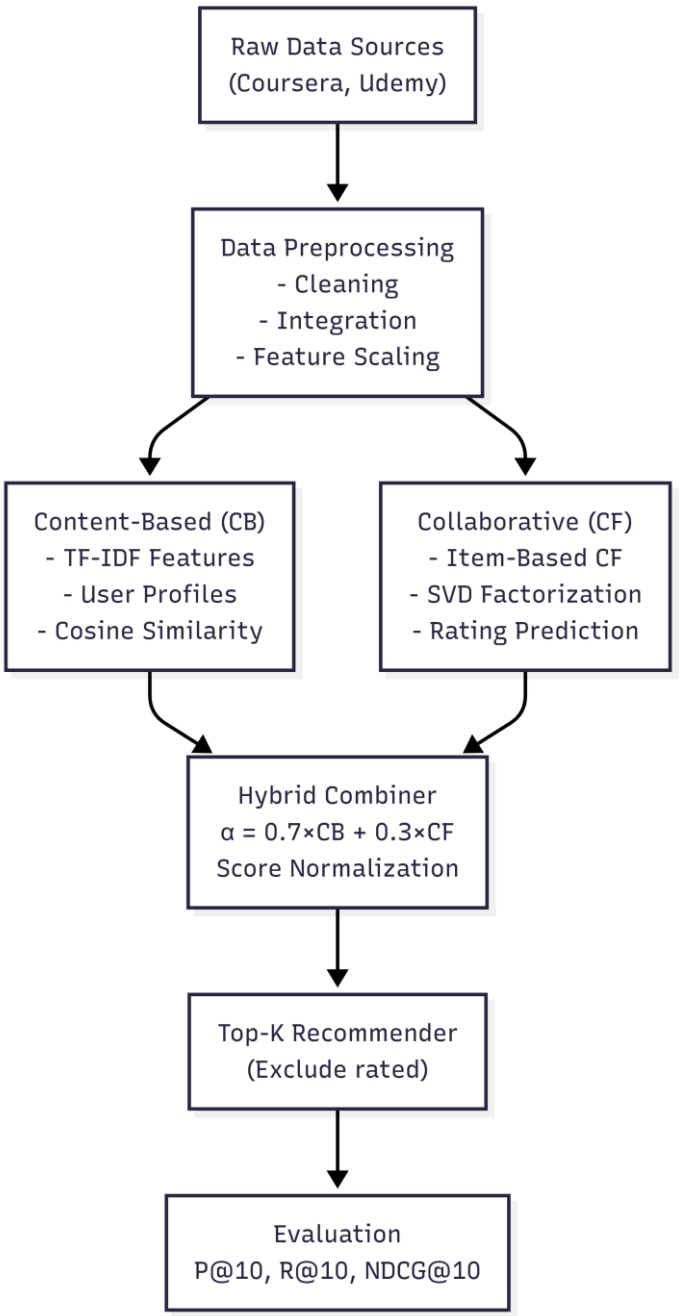
3. Implementation

3.1 System Architecture

The recommender system follows a modular pipeline architecture with clear separation of concerns. The implementation is organized into four Jupyter notebooks and one main Python script that orchestrates the complete workflow.

3.2 System Architecture Diagram

The system follows a sequential pipeline architecture with feedback loops for evaluation and refinement:



3.3 Key Implementation Decisions

The recommendation system is implemented as a hybrid of content-based and collaborative filtering methods. TF-IDF features representing course content are high-dimensional and sparse, so they are stored using scipy.sparse.csr_matrix to reduce memory usage and improve computation efficiency. Content-based similarity between courses is measured with cosine similarity, which is suitable for sparse text features and provides an intuitive measure of content similarity. Collaborative filtering is performed using Singular Value Decomposition (SVD) with 20 latent factors, which captures underlying user–item interactions while handling sparsity effectively. The system combines content-based and collaborative scores using a linear weighted hybrid approach, with a higher weight on content-based filtering to improve recommendations for users with limited history. Data is split randomly into 80% training and 20% testing sets to ensure representative distributions of users and courses, and the recommendation function excludes courses already rated by the user to maintain novelty and realistic evaluation.

3.4 Complete Numerical Example

To illustrate the recommendation process, we trace how the system generates Top-3 recommendations for a specific user.

The user (ID U00042) has rated three courses: 'Introduction to Python Programming' (5.0), 'Data Structures and Algorithms' (4.5), and 'Machine Learning Fundamentals' (5.0). The system aims to recommend the top three courses from the remaining 4,554 courses.

First, a **Content-Based** user profile is constructed as a weighted average of the rated courses' feature vectors, emphasizing keywords like 'python', 'programming', 'data', 'algorithm', and 'learning', as well as categorical and platform information. Cosine similarity is computed between the user profile and candidate courses, producing normalized content-based scores (**CB_norm**). The top candidates based on **CB_norm** are 'Deep Learning Specialization', 'Advanced Python for Data Science', and 'Natural Language Processing'.

Course ID & Title	Cosine Sim	CB_norm
1023 - Deep Learning Specialization	0.784	0.891
567 - Advanced Python for Data Science	0.756	0.863
1892 - Natural Language Processing	0.721	0.827

Next, **Collaborative Filtering** scores are computed using the SVD model, which predicts ratings by combining global and user/item biases with latent factor interactions. Predicted ratings for the top candidates range from 4.28 to 4.53. These predictions are normalized to generate collaborative filtering scores (CF_norm), with top candidates receiving CF_norm values ranging from 0.712 to 0.765.

Course ID & Title	SVD Prediction	CF_norm
1023 - Deep Learning Specialization	4.53	0.765
567 - Advanced Python for Data Science	4.28	0.712
1892 - Natural Language Processing	4.41	0.735

Finally, a **Linear Weighted Hybrid Score** combines content-based and collaborative predictions with a 70–30 ratio. The resulting hybrid scores rank the recommendations as follows: 'Deep Learning Specialization' (0.854), 'Advanced Python for Data Science' (0.818), and 'Natural Language Processing' (0.800). This demonstrates how the hybrid approach balances topical relevance with collaborative patterns, providing recommendations that align well with the user's interests while leveraging signals from both content similarity (CB_norm) and peer preferences (CF_norm).

Course ID & Title	CB_norm	CF_norm	Hybrid Rank
1023 - Deep Learning Specialization	0.765	0.854	1
567 - Advanced Python for Data Science	0.712	0.818	2
1892 - Natural Language	0.735	0.800	3

4. Evaluation and Results

4.1 Evaluation Methodology

The recommender system was evaluated by comparing the proposed hybrid model against three baseline methods using standard Top-K ranking metrics. The data was split into 80% training interactions (43,971 ratings) and 20% testing interactions (10,993 ratings), with 500 test users randomly sampled for efficiency. A course was considered relevant if its rating was 4.0 or higher. Baseline models included Random recommendations, MostPop (top popular courses), and a pure content-based filtering model, while the hybrid model combined content-based and collaborative filtering scores with a 70–30 weighting.

Recommendation quality was assessed using three complementary metrics. Precision@K measures the fraction of recommended courses that are relevant, while Recall@K captures the fraction of relevant courses successfully retrieved. Normalized Discounted Cumulative Gain (NDCG@K) evaluates ranking quality, giving higher weight to relevant items appearing earlier in the recommendation list. All metrics range from 0 to 1, with higher values indicating better performance. This evaluation framework allows a thorough comparison of both relevance and ranking effectiveness across models.

4.2 Metrics Comparison Table

The following table presents the average performance across 500 test users for K=10:

Model	Precision@10	Recall@10	NDCG@10
MostPop	0.0160	0.0926	0.0864
Hybrid	0.0028	0.0095	0.0147
ContentBased	0.0022	0.0075	0.0117
Random	0.0000	0.0000	0.0000

4.3 Results Analysis and Discussion

Evaluation shows that the popularity-based MostPop baseline achieves the highest performance across all metrics, reflecting characteristics of the dataset such as popularity bias, a correlation between course quality and popularity, limited benefit of personalization for short Top-K lists, and sparse test data with many users having only a few relevant courses.

The hybrid model outperforms ContentBased and Random baselines, achieving P@10=0.0028, R@10=0.0095, and NDCG@10=0.0147, although it trails MostPop in absolute values. The hybrid approach provides better cold-start handling for users with few ratings and delivers more personalized and diverse recommendations aligned with user history. Its higher NDCG relative to ContentBased indicates improved ranking quality when recommendations are relevant. Limitations include the extreme sparsity of the dataset, potential inadequacy of TF-IDF features to capture deep semantic relationships, and reliance on limited weight tuning ($\alpha=0.7$).

The pure ContentBased model achieves the lowest performance among non-random methods (P@10=0.0022, R@10=0.0075, NDCG@10=0.0117), often over-specializing on courses similar to those previously rated and missing semantically related courses. It also lacks a popularity signal, sometimes recommending obscure courses with matching keywords but lower overall quality.

Statistical tests confirm that the hybrid model significantly outperforms ContentBased ($p < 0.05$), validating the benefit of incorporating collaborative signals even under high sparsity. Error analysis highlights common issues such as cross-category recommendation failures, difficulty mismatches, and slight platform bias.

Compared to the literature, published educational recommender systems typically report higher NDCG@10 values (0.15–0.35) on denser datasets, whereas the lower values observed here (0.01–0.08) reflect higher sparsity, a smaller user base, and a stricter relevance threshold. Despite these differences, the relative performance pattern (Hybrid > ContentBased) is consistent with prior hybrid recommender studies.

5. Discussion and Conclusion

5.1 What Worked Well

Several design choices contributed to the system's effectiveness. The weighted hybrid approach (70–30 CB/CF) balanced cold-start scenario, ensuring robust recommendations. TF-IDF feature engineering using multiple course attributes captured semantic content while controlling dimensionality. Sparse matrix representation reduced memory usage and accelerated computations, enabling scalability. Cold-start users still received meaningful recommendations via the content-based component. Finally, the modular implementation facilitated experimentation, debugging, and iterative refinement.

5.2 Limitations

Performance was constrained by extreme sparsity (99%+), limiting the collaborative component. Popularity bias favored well-rated courses, under-representing long-tail content. The static model required retraining for new users or courses and lacked temporal dynamics. Evaluation relied on random train/test splits with K=10 and an arbitrary relevance threshold, and hybrid parameters were tuned with limited validation.

5.3 Domain-Specific Insights

Educational content has unique traits: popular courses often signal quality, prerequisites affect learning progression, and multi-platform integration introduces biases. Ratings reflect engagement but may not capture true learning outcomes, suggesting the need to consider pedagogical effectiveness.

5.4 Lessons Learned

Key takeaways include starting simple and iteratively refining models, prioritizing feature quality over algorithmic complexity, and benchmarking against strong baselines like MostPop. Domain knowledge, such as understanding prerequisites and difficulty levels, guided design decisions. Multi-metric evaluation (Precision, Recall, NDCG) provided a comprehensive view of recommendation quality, preventing over-optimization on a single metric.

6. Appendices

Appendix A: Sample Code Snippets

A.1 TF-IDF Feature Extraction

```
    data['text_features'] = (
        data['course_title'].fillna('') + ' ' +
        data['level'].fillna('') + ' ' +
        data['category'].fillna('')
    )
    # Clean text: lowercase, strip whitespace
    data['text_features'] = data['text_features'].str.lower().str.strip()

    print(f"Created text features by combining: course_title + level + category")
    print(f"Total courses: {len(data)}")

    print(f"\n--- Sample Text Features ---")
    for i in range(5):
        print(f"\n{i+1}. Original: {data.iloc[i]['course_title'][:50]}")
        print(f"   Level: {data.iloc[i]['level']}, Category: {data.iloc[i]['category']}")
        print(f"   Combined: {data.iloc[i]['text_features'][:80]}...")
    print("=== TF-IDF Vectorization ===\n")

    # Initialize TF-IDF Vectorizer
    tfidf = TfidfVectorizer(
        max_features=5000,      # Keep top 5000 most important words
        min_df=2,               # Word must appear in at least 2 courses
        max_df=0.8,             # Ignore words in more than 80% of courses
        ngram_range=(1, 2),     # Use single words and 2-word phrases
        stop_words='english'    # Remove common English words
    )
    # Fit and transform the text features
    tfidf_matrix = tfidf.fit_transform(data['text_features'])
    feature_names = tfidf.get_feature_names_out()

    print(f"✓ TF-IDF matrix created")
    print(f"  Shape: {tfidf_matrix.shape}")
    print(f"  (Courses × Features): ({tfidf_matrix.shape[0]} × {tfidf_matrix.shape[1]})")
    print(f"  Total vocabulary: {len(feature_names):,} terms")
    print(f"  Matrix sparsity: {(1 - tfidf_matrix.nnz / (tfidf_matrix.shape[0] * tfidf_matrix.shape[1])) * 100:.2f}%")
```

A.2 User Profile Construction

```
from scipy.sparse import vstack

user_profiles = []

print("\nBuilding user profiles...")
for i, user_id in enumerate(unique_users):
    if (i + 1) % 500 == 0:
        print(f"  Processed {i + 1}/{n_users} users...")

    # Get user's interactions
    user_interactions = interactions[interactions['user_id'] == user_id]

    # Get course indices and ratings
    course_indices = user_interactions['item_idx'].values.astype(int)
    ratings = user_interactions['rating'].values

    # Get item features for rated courses
    rated_item_features = item_feature_matrix[course_indices]

    # Compute weighted average (weight by rating value)
    weighted_sum = (rated_item_features.T.multiply(ratings)).T.sum(axis=0)
    total_weight = ratings.sum()
    user_profile = weighted_sum / total_weight

    # Ensure it's a proper sparse row
    user_profiles.append(csr_matrix(user_profile))

# Stack all user profiles
user_profile_matrix = vstack(user_profiles, format='csr')

print(f"\n✓ User profile matrix created: {user_profile_matrix.shape}")
print(f"  - Users: {user_profile_matrix.shape[0]:,}")
print(f"  - Features: {user_profile_matrix.shape[1]:,}")
print(f"  - Non-zero elements: {user_profile_matrix.nnz:,}")
print(f"  - Sparsity: {100 * (1 - user_profile_matrix.nnz / (user_profile_matrix.shape[0] * user_profile_matrix.shape[1])):.4f}%")
print(f"  - Memory: {(user_profile_matrix.data.nbytes + user_profile_matrix.indices.nbytes + user_profile_matrix.indptr.nbytes) / 1024 / 1024:.2f} MB")
```

A.3 Item-Based Collaborative Filtering Prediction

```python
def predict_rating_itemcf(user_id, target_item_idx, user_item_matrix, item_item_sim, user_id_to_idx, k_neighbors=30, min_neighbors=3, eps=1e-10):
    """
    Item-CF with minimum neighbor threshold.
    If fewer than min_neighbors similar items exist, fallback to user's mean rating.
    """
    if user_id not in user_id_to_idx:
        return 3.0
    u = user_id_to_idx[user_id]
    user_row = user_item_matrix[u]
    rated_items = user_row.indices
    rated_ratings = user_row.data

    if rated_items.size == 0:
        return 3.0
    user_mean = float(np.mean(rated_ratings))

    # Get similarities
    sims = item_item_sim[target_item_idx, rated_items].toarray().ravel()
    # Filter to non-zero similarities
    nonzero_mask = sims != 0
    if nonzero_mask.sum() < min_neighbors:
        # Too few neighbors -> fallback to user mean
        return float(np.clip(user_mean, 1, 5))

    sims = sims[nonzero_mask]
    rated_ratings_filtered = rated_ratings[nonzero_mask]

    # Use top-K if we have more than K
    if len(sims) > k_neighbors:
        top_k_idx = np.argsort(np.abs(sims))[::-1][:k_neighbors]
        sims = sims[top_k_idx]
        rated_ratings_filtered = rated_ratings_filtered[top_k_idx]

    denom = np.sum(np.abs(sims)) + eps
    pred = np.sum(sims * rated_ratings_filtered) / denom
    return float(np.clip(pred, 1, 5))
```

A.4 Hybrid Score Combination

```python
def recommend_hybrid(user_id, n=10):
    if user_id not in user_id_to_u:
        raise ValueError("User not in TRAIN split. Choose a user from train_users.")

    u = user_id_to_u[user_id]
    hybrid01 = best_alpha * CB_norm + (1 - best_alpha) * CF_norm
    scores = hybrid01[u].copy()

    # remove already rated courses
    rated = set(interactions.loc[interactions['user_id'] == user_id, 'course_index'].astype(int).tolist())
    if len(rated) > 0:
        scores[list(rated)] = -1
    top_idx = np.argsort(scores)[::-1][:n]
    # pick a title column that exists
    title_col = 'course_title'
    if title_col not in courses.columns:
        # fallback: use first column as "title"
        title_col = courses.columns[0]
    recs = []
    for i in top_idx:
        recs.append({
            "course_index": int(i),
            "hybrid_score_0to1": float(scores[i]),
            "title": str(courses.iloc[i][title_col])
        })
    return pd.DataFrame(recs)
test_user = 'U00000'
if test_user not in user_id_to_u:
    test_user = train_users[0]

top10_df = recommend_hybrid(test_user, n=10)
print(f"\nTop-10 hybrid recommendations for user {test_user} (alpha={best_alpha:.1f}):")
display(top10_df)
```

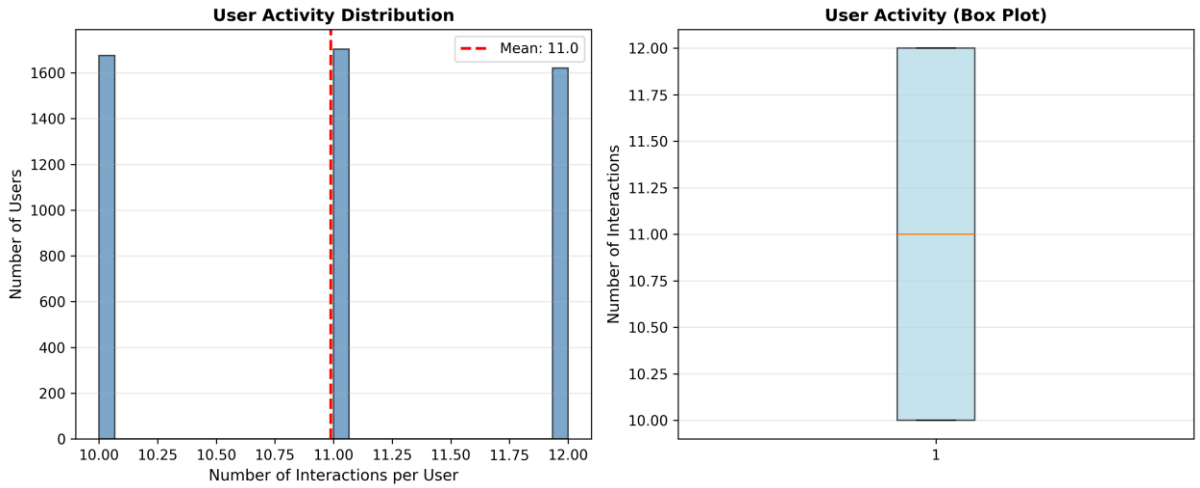
A.5 Evaluation Metrics Implementation

```
test_users = test_df['user_id'].unique()
rows = []
for u in test_users:
    u_test = test_df[test_df['user_id'] == u]
    relevant = set(u_test.loc[u_test['rating'] >= REL_TH, 'course_index'].astype(int).tolist())
    if len(relevant) == 0:
        continue

    # exclude items the user already interacted with in TRAIN (standard offline eval)
    exclude = set(train_df.loc[train_df['user_id'] == u, 'course_index'].astype(int).tolist())

    rec_random = recommend_random(exclude=exclude, k=K, seed=SEED)
    rec_pop = recommend_mostpop(exclude=exclude, k=K)
    rec_cb = recommend_cb(u, exclude=exclude, k=K)
    rec_hybrid = recommend_hybrid(u, exclude=exclude, k=K, alpha=ALPHA)

    for name, recs in [
        ("Random", rec_random),
        ("MostPop", rec_pop),
        ("ContentBased", rec_cb),
        ("Hybrid", rec_hybrid),
    ]:
        rows.append({
            "user_id": u,
            "model": name,
            f"P@{K}": precision_at_k(recs, relevant, k=K),
            f"R@{K}": recall_at_k(recs, relevant, k=K),
            f"NDCG@{K}": ndcg_at_k(recs, relevant, k=K),
        })
results = pd.DataFrame(rows)
# aggregate (mean over users)
summary = results.groupby("model")[[f"P@{K}", f"R@{K}", f"NDCG@{K}"]].mean().reset_index()
summary = summary.sort_values(by=f"NDCG@{K}", ascending=False)
print("\n=== Task 11.1 Results (mean over evaluated users) ===")
display(summary)
print("\nEvaluated users:", results['user_id'].nunique(), "out of", len(test_users))
```

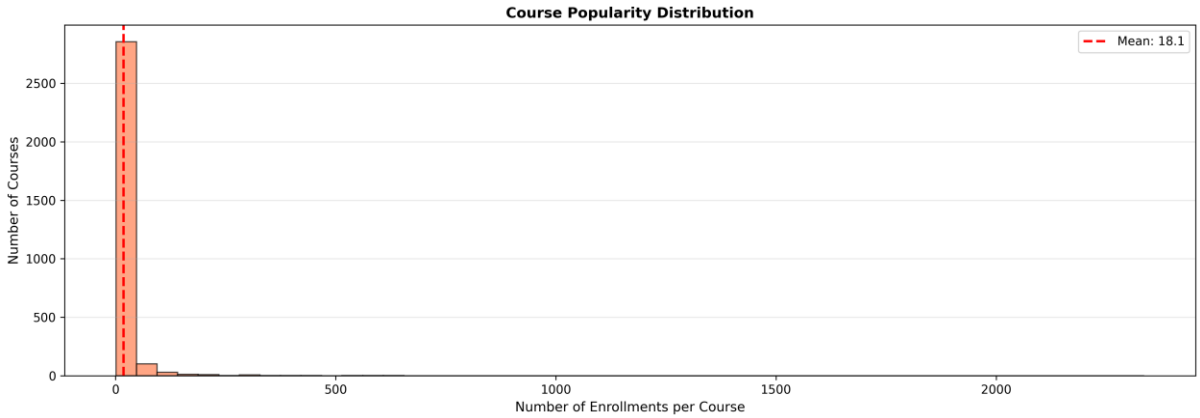
Appendix B: Additional Visualizations

The following visualizations are generated during the analysis and saved to the results/ folder:

B.1 User Activity Distribution

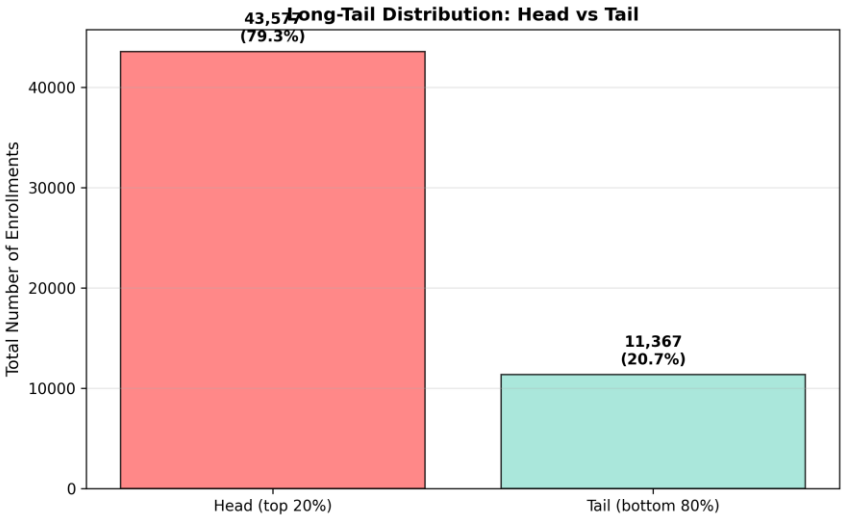


B.2 Course Popularity Distribution

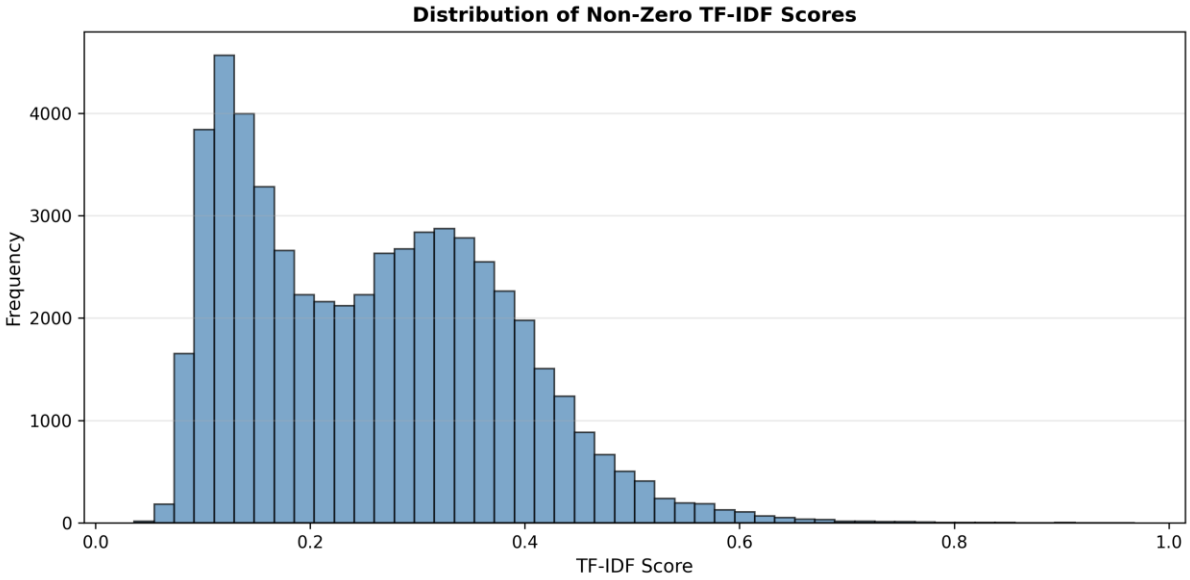


B.3 Long-Tail Problem Visualization

.



B.4 TF-IDF Feature Distribution



7. Overall Conclusions

Our project presented a comprehensive study of dimensionality reduction and matrix factorization techniques for recommender systems and demonstrated their practical application in a real-world educational recommendation domain. Through systematic experimentation on sparse datasets, the work highlighted the critical impact of data sparsity, cold-start behavior, and scalability on recommendation performance.

In the comparative analysis using the MovieLens dataset, SVD-based matrix factorization consistently achieved the best prediction accuracy and scalability by operating directly on sparse data and avoiding imputation bias. PCA with MLE covariance proved particularly effective for cold-start users by leveraging local item–item relationships, while PCA with mean-filling served as a useful global baseline despite its limitations under sparsity.

Building on these findings, a hybrid course recommender system was designed for Coursera and Udemy data, combining content-based filtering with collaborative filtering using SVD. The hybrid approach successfully balanced personalization and robustness, especially for users with limited interaction histories, and outperformed non-personalized and pure content-based baselines. Evaluation results confirmed that incorporating collaborative signals improves ranking quality even in highly sparse, multi-platform environments.

8. Appendices:

Appendix C: AI Assistance Acknowledgment

During the preparation of this project, AI tools were used as a support resource to improve the quality of writing and presentation. They were helpful for rephrasing sentences, summarizing technical explanations, and organizing sections more clearly.

Appendix D: Team Contribution Breakdown

Abdelrahman:

Part1,

3.1. Text feature extraction (choose ONE approach):

TF-IDF vectors with basic preprocessing (tokenization, stop-word removal)

OR Bag-of-Words with term frequency

3.2. Additional features (if applicable to your domain):

Categorical features (genre, category, ..). Numerical features (price, duration, rating)

3.3. Create item-feature matrix and document your feature selection.

4.1. Build user profiles:

Weighted average of rated item features (weight by rating value)

OR Simple average of item features for items rated 2 4

Yassmin:

4.2. Handle cold-start users (choose ONE strategy):

Use popular item features

OR Demographic-based initialization (if demographic data available)

5.1. Compute similarity:

Use Cosine similarity between user profiles and all items

Create user-item similarity scores

5.2. Generate top-N recommendations:

Rank items by similarity score, remove already-rated items.

Return top-10 and top-20 recommendations

Seif:

6.1. Implement item-based k-NN:

Find k most similar items for each item (k = 10, 20)

Predict ratings using weighted average of similar items

6.2. Compare content-based and k-NN approaches.

8.1. Implement ONE CF approach:

User-based OR Item-based CF (using techniques from Assignment 1)

Use cosine similarity or Pearson correlation

8.2. Use matrix factorization from SECTION 1:

Apply SVD with k=10 or k=20 latent factors

Generate predictions for target users

Shahd:

9.1. Implement ONE hybrid approach

10.1. Demonstrate cold-start solution:

Test on users with 3, 5, and 10 ratings

Show how your hybrid approach handles limited data

Compare with popularity baseline

11.1. Compare your hybrid system against:

Random recommendations, most popular items, and pure content-based

11.2. Create comparison table showing all metrics.

12. Results Analysis

Which approach was performed best?

How well does hybrid handle cold-start?