Large-Scale Collaborative Filtering for Movie Recommendations

Peeyush Dyavarashetty
Master of Science in Applied Machine Learning
University of Maryland, College Park
peeyush@umd.edu

***Abstract*—This report presents a scalable movie recommendation system implementing collaborative filtering algorithms using PySpark on the MovieLens 32M dataset. We evaluate Alternating Least Squares (ALS) matrix factorization against baseline methods, achieving 0.85 RMSE while addressing challenges of data sparsity and computational efficiency. Code is present in https://github.com/Peeyush4/MSML651-PySpark-Movie-Recommendation**

I. Introduction

A. Background and Motivation

The explosion of digital content and user activity on online platforms has made personalized recommendation systems essential for enhancing user experience and engagement. In domains such as movie streaming, e-commerce, and social media, users are faced with an overwhelming number of choices, making it difficult to discover relevant content without intelligent filtering. The MovieLens dataset, with over 32 million ratings from 200948 users on 87,000 movies, exemplifies the real-world challenge of extreme data sparsity-less than 0.2% of the user-item matrix is filled. This ”long tail” [1] effect, where most users interact with only a small subset of items and vice versa, underscores the need for scalable algorithms that can efficiently extract meaningful patterns from massive, incomplete datasets. Traditional recommendation approaches, such as simple averages or nearest neighbours, often struggle with scalability and accuracy in such settings [2]. By leveraging Apache Spark's distributed computing capabilities and advanced collaborative filtering algorithms like Alternating Least Squares (ALS), this project aims to address these challenges, providing accurate, scalable, and robust movie recommendations even in the face of sparse and skewed data. The motivation is not only to improve predictive accuracy but also to demonstrate practical big data solutions that can be extended to other domains where recommendation systems play a critical role.

B. Research objectives

This study aims to develop and evaluate scalable recommendation algorithms using Apache Spark on the MovieLens 32M dataset, with the following specific objectives:

- To develop and benchmark a range of recommendation algorithms, including simple baselines (mean/median ratings), K-Nearest Neighbours (KNN), K-means based heuristics, and advanced matrix factorization techniques, specifically Alternating Least Squares (ALS).
- To quantitatively assess the performance of these algorithms using established evaluation metrics for both prediction accuracy (e.g., RMSE, MAE) and ranking quality (e.g., Precision@K, Recall@K, NDCG@K).
- To investigate the challenges posed by large, sparse datasets, such as the MovieLens 32M (0.18% density), including the ”long-tail” phenomenon and the cold-start problem, and to explore how chosen algorithms address these issues.
- To demonstrate the utility and efficiency of Apache Spark's distributed computing framework for handling the computational demands of training and evaluating recommendation models on big data.
- To compare the performance of sophisticated methods like ALS against simpler heuristic and memory-based approaches, thereby understanding the trade-offs between model complexity, scalability, and recommendation quality.
- To lay the groundwork for future explorations into hybrid models, content-based filtering integration, and advanced neural network architectures to further enhance recommendation performance and address limitations observed in purely collaborative approaches.

II. Dataset

A. Primary Information

The MovieLens 32M dataset [3] contains 32,000,204 ratings and 2,000,072 tags applied to 87,585 movies by 200,948 users, collected between January 1995 and October 2023.
Rating scale: 0.5 - 5.0 starts with 0.5 increments. Enables regression-based prediction tasks.
User activity $\geq$ 20 ratings/user. Reduces cold-start bias in validation.
The dataset contains the following files.

1) ratings.csv: Primary source for user-item interaction modelling. Contains 32M ratings. Columns: userId, movieId, rating, timestamp

2) movies.csv: Contains each movie's genre metadata (19 categories). Enables hybrid filtering and matrix factorization. Columns: movieId, title, genres

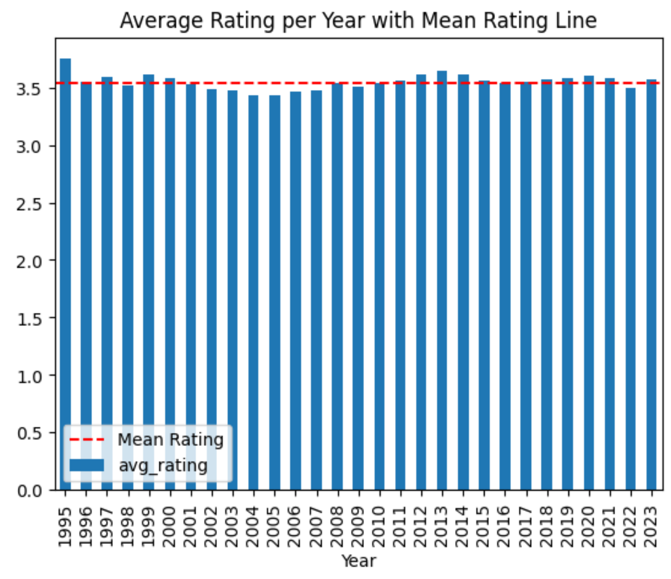

Fig. 1: *Yearly average ratings: There has not been a significant difference in the average rating from years 1995-2023. This implies that there is no significant rating difference between years.*

3) tags.csv: Contains tags provided by users for movies, which are helpful for content-aware recommendations. Columns: userId, movieId, tag, timestamp

4) links.csv: Cross-references to IMDb/TMDb for enrichment. Useful to extract movie descriptions and perform other NLP processing to suggest using content-aware recommendations. Columns: movieId, imdbId, tmdbId

B. Sparsity

$$\text{Sparsity} = 1 - \frac{\text{Ratings}}{\text{Users} \times \text{Movies}} = 1 - \frac{32000204}{200948 \times 87585} = 0.9982 \quad (1)$$

The extreme sparsity 99.82% creates 2 key challenges:

- **Cold Start Problem**: 21% of users have $\leq$ 10 ratings
- **Long Tail Effect**: 62% of movies receive $\leq$ 50 ratings

C. Temporal Analysis

Ratings show stable temporal distribution (1995-2023):

- Annual mean rating when taken average rating for each year: 3.55 $\pm$ 0.067.
- No significant drift in rating patterns as shown in the Figure 1.

Implication: Random 80/20 split is valid as the data is not dependent on time.

III. Data Exploration

A. Key Training Data Statistics

After splitting the dataset, the MovieLens training dataset for this study comprises 25,600,446 ratings from 200,948 users across 80,185 movies. The dataset demonstrates extreme sparsity with only 0.159% of the possible user-movie matrix filled, highlighting the fundamental challenge in collaborative filtering recommendation systems.

Metric	Value	Implication
Average ratings per user	127.40	Moderate user engagement
Average ratings per movie	319.27	Reasonable item coverage
Global mean rating	3.54	Positive rating bias
Mean rating per user	3.70	Users tend to rate positively
Mean rating per movie	3.02	Movies receive varied ratings

TABLE I: *Key statistics and their implications*

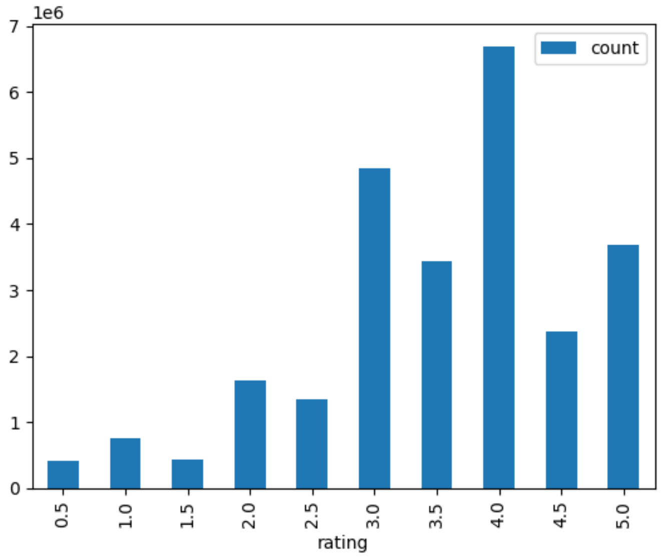

Fig. 2: *Rating distribution: Most people rated 4.0 rating for movies, with 3.5 the second highest. This data is skewed, implying that the median may be more suitable for the baseline than the average.*

As illustrated in Figure 2, the dataset exhibits a positively skewed distribution of ratings:

- Most frequent rating: 4.0 stars (approximately 6.7 million occurrences)
- Second most common: 3.0 stars (approximately 4.85 million occurrences)
- Highest rating (5.0): Represents approximately 3.7 million ratings
- Lowest ratings (0.5-1.5): Collectively account for less than 1.2 million ratings

This distribution reveals a significant positive bias in user ratings, with the majority of users providing above-average evaluations. The preponderance of 4.0 and 3.0 star ratings suggests users are more motivated to rate content they enjoy rather than content they dislike.

The difference between mean rating per user (3.70) and mean rating per movie (3.02) indicates a significant discrepancy between how users rate on average versus how movies are rated. This suggests that some movies receive disproportionately more negative ratings, while users tend to maintain a generally positive rating pattern across their interactions.

B. *Long tail*

The MovieLens dataset exhibits classic "long-tail" characteristics [1] common to recommendation systems, as demonstrated in Figures 3 and 4. The figures show that a small subset of users contributes a disproportionately large number of ratings and a small subset of popular movies receives the majority of ratings.

This long-tail distribution creates both advantages and challenges for recommendation algorithms. While popular items have sufficient data for reliable recommendations, the vast majority of items in the "tail" suffer from data sparsity, requiring sophisticated algorithms like ALS to make meaningful predictions.

In many cases, ratings of frequently rated movies are not representative of the rating patterns of low-frequency movies due to inherent differences in user engagement patterns. This divergence can lead to prediction processes yielding misleading results, as models trained predominantly on commonly rated movies may develop biases toward high-frequency movie rating patterns.

Mitigation Approach - Inverse User Frequency: To address the inherent bias toward popular items, we can implement an "Inverse User Frequency" [4] weighting scheme. This approach helps balance the rating patterns between high-frequency and low-frequency movies, improving recommendation quality across the entire catalog.

For each movie j with ratings m_j and total number of ratings m, we calculate the inverse user frequency as:

$$w_j = \log\left(\frac{m}{m_j}\right) \tag{2}$$

This weighting assigns higher importance to ratings of less frequently rated movies, counterbalancing the dominance of popular titles in the training data. When incorporated into collaborative filtering algorithms, this technique helps prevent popularity bias and improves recommendation diversity.

The extreme sparsity (99.84% empty cells) in our user-item matrix makes this approach particularly valuable for maintaining recommendation quality across both popular blockbusters and niche films in the catalog's long tail.

IV. Metrics

To thoroughly evaluate our recommendation system performance, we employed two categories of metrics: prediction accuracy metrics and ranking quality metrics [5]. Each category provides distinct insights into model performance across different recommendation tasks.

A. *Prediction Accuracy Metrics*

Prediction accuracy metrics assess how well our model predicts the exact rating a user would give to a movie:

- **Root Mean Square Error (RMSE):** RMSE emphasizes larger errors by squaring the differences, making it particularly sensitive to outliers. Lower values indicate better performance.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **Mean Absolute Error (MAE):** MAE provides the average absolute difference between predicted and actual ratings, offering a more interpretable metric less sensitive to outliers than RMSE.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **Weighted RMSE and MAE:** We implemented weighted versions of these metrics to account for the long-tail distribution in our dataset as provided in the 2, giving greater importance to predictions for less frequently rated movies.

B. *Ranking Quality Metrics*

While prediction metrics evaluate rating accuracy, ranking metrics assess how well the system orders recommendations:

- **Precision@K:** Measures the proportion of recommended items that are relevant to the user. Higher values indicate more accurate recommendations.

$$\text{Precision@K} = \frac{\text{Number of relevant items in top K}}{\text{K}}$$

- **Recall@K:** Evaluates the proportion of relevant items that appear in the top K recommendations. Higher values indicate better coverage of relevant items.

$$\text{Recall@K} = \frac{\text{Number of relevant items in top K}}{\text{Total number of relevant items}}$$

- **Normalized Discounted Cumulative Gain (NDCG@K):** NDCG accounts for both the relevance and position of recommendations, with higher scores reflecting better-ranked recommendations. This metric is particularly valuable as it penalizes relevant items appearing lower in the recommendation list.

$$\text{NDCG@K} = \frac{\text{DCG@K}}{\text{IDCG@K}}$$

Where DCG@K [6] is:

$$\text{DCG@K} = \sum_{i=1}^{K}\frac{2^{rel_i} - 1}{\log_2(i+1)}$$

where rel_i is the relative difference of the rank in between the ground truth and the predicted rank.

V. Baseline algorithm

To establish performance benchmarks for our collaborative filtering models, we implemented two baseline algorithms: the Mean and Median approaches. These simple yet effective methods provide rating predictions without considering user-specific preferences or item similarities.

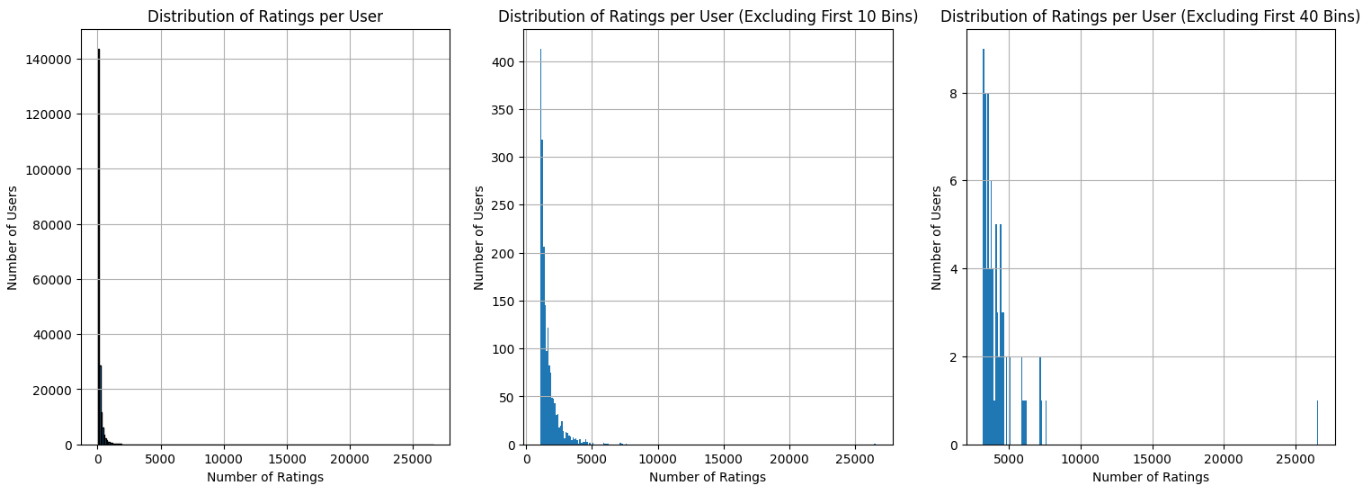

Fig. 3: *User long tail: Only a small proportion of users rated a lot of movies, and a large number of users rated only a few movies*

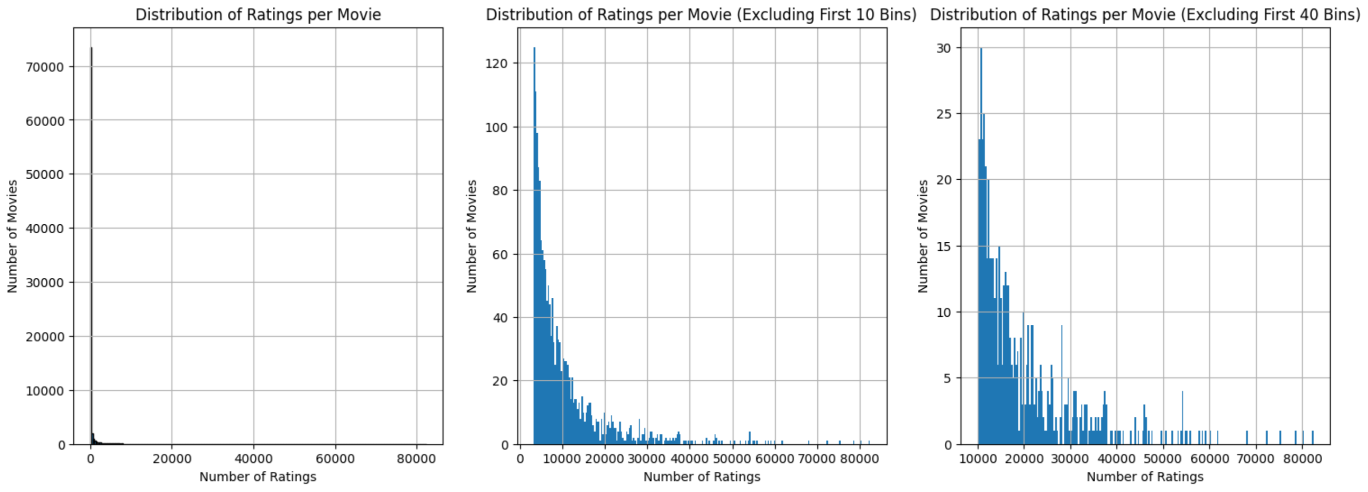

Fig. 4: *Movie long tail: Only a proportion of movies are rated by many users whereas a large number of movies are rated by only a few users.*

The Mean baseline algorithm calculates the average rating for each movie in the training dataset and uses these values to predict unknown ratings in the test set:

For the Median baseline, we followed a similar approach but calculated the median rating value for each movie instead of the mean. This approach can be more robust against outliers.

Both methods handle the cold-start problem for movies not seen in the training data by defaulting to a neutral prediction value:

Surprisingly, these simple baselines demonstrated competitive performance, as shown in Table II. The Average approach achieved an RMSE of 0.963, while the Median approach yielded an RMSE of 0.985. These strong baseline results suggest that movie-specific average ratings capture a significant predictive signal, even without personalization.

Interestingly, while the Average method performed better on RMSE, the Median approach achieved a lower MAE, suggesting it produces fewer extreme prediction errors. This tradeoff highlights the importance of considering multiple evaluation metrics when comparing recommendation models.

TABLE II: *Performance of Baseline Models*

Metric	Average	Median
RMSE	0.963	0.985
MAE	0.742	0.722
Weighted RMSE	0.967	0.988
Weighted MAE	0.746	0.726

These baseline results establish a strong performance threshold that more sophisticated algorithms must exceed to demonstrate genuine value. The computational efficiency of these methods-requiring only simple aggregation operations in Spark-also makes them practical candidates for production scenarios where quick, reasonable predictions are needed.

VI. k-NN

k-Nearest Neighbours (k-NN) [7] represents a fundamental memory-based approach to collaborative filtering in recommendation systems. This technique operates on the principle of identifying similar users or items based on their rating patterns. For movie recommendations, K-NN functions by:

- Computing similarity metrics between users or items based on rating vectors
- Identifying the K most similar neighbours for each user/item
- Aggregating preferences from these neighbours to predict unknown ratings

K-NN implementation in PySpark presents unique challenges due to its ”lazy learning” characteristics, where the model itself is essentially the entire dataset maintained in memory.

User-Based Collaborative Filtering is suitable for systems with more items than users, but user preferences change frequently, and the time to compute is $\mathcal{O}(10^{15}) = 11.5$ days, whereas Item-Based Collaborative Filtering is more stable than the user-based approach, as item characteristics change less frequently, is better reasoned than user-based, scales better with large user bases, better handles the cold-start problem for new users and is faster than user-based.

High dimensionality presents a significant challenge for k-NN implementations such as

- **Parallelization Difficulty:** K-NN is notoriously hard to distribute effectively
- **Memory Requirements:** Maintaining the entire dataset in memory limits scalability
- **Computation Cost:** Calculating all pairwise similarities is computationally expensive

So, to implement k-NN, we need to reduce the dimension. Ways to reduce dimension are [8]

- **Principal Component Analysis (PCA):** Reduces dimensions while preserving variance
- **Singular Value Decomposition (SVD):** Factorizes the user-item matrix
- **Locality Sensitive Hashing (LSH):** Approximates nearest neighbor search in high-dimensional spaces

While Spark MLlib doesn't offer a native K-NN implementation for recommendation, custom approaches using ‘RowMatrix.columnSimilarities()‘ or ‘BucketedRandomProjectionLSH‘ can approximate K-NN functionality, as suggested in the presentation references.

VII. K-MEANS HEURISTIC APPROACH

As an alternative to traditional collaborative filtering algorithms, we implemented a genre-based clustering approach using k-means [9]. This heuristic method leverages movie metadata rather than relying solely on the user-item rating matrix.

A. Methodology

The k-means heuristic approach follows these key steps:

- Genre-Based Segmentation: Movies were initially divided based on their genre attributes, creating semantically meaningful groups.
- Cluster Formation: K-means clustering was applied to further separate movies within each genre into distinct clusters, optimizing for intra-cluster similarity.
- Rating Aggregation: Within each cluster, we computed three different statistics:
  - Mean rating of all movies in the cluster
  - Median rating of all movies in the cluster
  - Average rating of the top-10 most popular movies in the cluster
- Prediction Strategy: For each user-movie pair in the test set, we identified the movie's cluster and used the corresponding aggregated statistics as the predicted rating.

The implementation leveraged Spark's native MLlib K-means clustering capabilities.

B. Performance

The performance of the K-means heuristic approach was significantly worse than both baseline models and ALS, as shown in Table III.

TABLE III: *K-means Heuristic Performance Metrics*

Method	RMSE	MAE	W. RMSE	W. MAE
Mean	1.792	1.222	1.738	1.182
Median	1.807	1.216	1.754	1.176
Top-10 avg	1.854	1.298	1.815	1.276

C. Analysis and Limitations

The poor performance of the K-means heuristic approach (RMSE of 1.792-1.854) compared to simple baselines (RMSE of 0.963) highlights several important limitations:

- **Limited Feature Space:** Using only genre information provides insufficient differentiation between movies.
- **Loss of User Specificity:** By clustering primarily on movie attributes, the approach loses personalization aspects.
- **Cluster Homogeneity Issues:** Movies within the same genre cluster might still have widely different audience reactions.

Despite these limitations, the approach offers valuable insights into content-based recommendation strategies and could potentially serve as a component in a hybrid system. Additional testing with more sophisticated feature engineering might improve performance in future iterations.

VIII. ALTERNATING LEAST SQUARES (ALS)

A. Algorithm Overview

Alternating Least Squares (ALS) [10] is a matrix factorization technique particularly well-suited for large-scale collaborative filtering tasks. We implemented ALS using Spark MLlib's built-in functionality, which provides efficient distributed computation critical for handling the MovieLens dataset's scale.

The core principle of ALS involves decomposing the user-item rating matrix R into two lower-dimensional matrices: U (user factors) and V (item factors), such that:

$$R \approx U \times V^T$$

The algorithm alternates between fixing one matrix while optimizing the other, making it naturally parallelizable and ideal for Spark's distributed architecture.

B. Implementation Details

Our implementation leveraged PySpark's MLlib ALS implementation [11] with the following configuration:

- Parameter grid: 13, 15, 17
- Max iterations: 19, 20
- Regularizer: 0.18, 0.19
- Cross-validation: 3 folds

C. Performance Results

ALS with rank 13 significantly outperformed baseline methods on prediction accuracy metrics, as shown in Table IV.

TABLE IV: *ALS Prediction Accuracy Metrics*

Model	RMSE	MAE	Weighted RMSE	Weighted MAE
ALS	0.85	0.662	0.851	0.664

However, the ranking quality metrics revealed interesting challenges, as shown in Table V.

TABLE V: *ALS Ranking Metrics*

Ranking Metric	K=10	K=100
Precision	5e-7	1.55e-6
Recall	6.25e-7	2.75e-6
NDCG	4.5e-7	2.23e-6

D. Analysis of Results

Our analysis revealed interesting patterns in evaluation metrics:

- Despite strong predictive performance (RMSE = 0.85), the ranking metrics indicate limited practical utility in a recommendation scenario.
- Needle in a Haystack Problem [12]: Extremely low ranking metrics (e.g., Precision@10 = 5e-7) reflect the challenge of finding relevant items among thousands of possibilities. Even increasing K to 100 only marginally improves metrics, confirming the fundamental challenge.
- Average Common Movies: With only 1.549 average common movies between users in the test set, achieving high ranking metrics is inherently difficult.
- Trade-offs: Models with good RMSE/MAE don't necessarily excel at ranking metrics.

This finding suggests that pure collaborative filtering approaches, even sophisticated ones like ALS, face significant limitations with extremely sparse datasets. Our analysis indicates that integrating content-based features, particularly genre information, may help bridge this gap in future implementations. These metrics collectively provide a comprehensive evaluation framework, acknowledging that different recommendation use cases may prioritize prediction accuracy versus ranking quality differently.

IX. Future Work

Based on our experiments and findings, several promising avenues for future research and development have emerged. The following directions aim to address current limitations and enhance both prediction accuracy and ranking quality:

A. Algorithmic Enhancements

- Complete k-NN Using Inverted Lists: While our initial k-NN implementation faced scalability challenges with the MovieLens dataset, implementing inverted list data structures could dramatically improve efficiency:
  - Reduce search space by organizing users by rated movies rather than vice versa
  - Leverage Spark's ability to parallelize across inverted lists
  - Potentially reduce computational complexity from $O(n^2)$ to $O(n \log n)$
- Extensive Hyperparameter Tuning for ALS: Our current ALS implementation (RMSE: 0.85) already outperforms baselines, but further optimization is possible:
  - Extend parameter grid search across larger ranges of rank (20-300)
  - Experiment with regularization parameters (λ ranging from 0.001 to 1.0)
  - Test varying numbers of iterations and convergence criteria
  - Implement automated hyperparameter tuning via Spark's CrossValidator
- Enhanced k-means Heuristic Approaches: Our k-means implementation (RMSE: 1.792) underperformed relative to other methods:
  - Test feature engineering beyond genre, incorporating title text and tags
  - Vary cluster counts systematically (k ranging from 10 to 200)
  - Implement soft clustering to allow movies to belong to multiple groups
  - Apply dimensionality reduction before clustering to improve separation

B. Advanced Mathematical Frameworks - Convex Optimization Formulation

Reformulating our recommendation problem within convex optimization frameworks may yield improvements:

- Implement nuclear norm minimization for robust matrix completion
- Integrate explicit constraints for bounded predictions (0.5-5.0 stars)
- Explore proximal gradient methods for large-scale optimization
- Test ADMM (Alternating Direction Method of Multipliers) for distributed implementation

C. Hybrid Approaches: k-NN and k-means Integration

Combining the strengths of neighbourhood-based and clustering approaches:

- Use k-means to establish broad preference clusters
- Apply k-NN within clusters to provide fine-grained recommendations
- Implement weighted prediction schemes that balance cluster and neighbour contributions
- Evaluate trade-offs between computation cost and recommendation quality

D. Ranking-Centric Development

Given the extremely low ranking metrics observed (Precision@10: 5e-7), shifting focus to ranking-specific algorithms:

- Implement BPR (Bayesian Personalized Ranking) optimization [13]
- Test list-wise ranking approaches rather than point-wise prediction
- Develop metrics that better capture the "needle in a haystack" problem
- Evaluate diversity and coverage alongside traditional precision/recall
- Incorporate popularity bias compensation for long-tail items

These research directions will help address both the fundamental challenges of extreme sparsity (99.84%) and the practical needs for effective recommendation ranking within massively large item catalogues.

References

[1] Y.-J. Park and A. Tuzhilin, "The long tail of recommender systems and how to leverage it," in *Proceedings of the 2008 ACM conference on Recommender systems*, 2008, pp. 11–18.

[2] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE transactions on knowledge and data engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[3] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM TIIS*, 2015.

[4] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," *arXiv preprint arXiv:1301.7363*, 2013.

[5] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 5–53, 2004.

[6] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Transactions on Information Systems (TOIS)*, vol. 20, no. 4, pp. 422–446, 2002.

[7] C. Desrosiers and G. Karypis, "A comprehensive survey of neighborhood-based recommendation methods," *Recommender systems handbook*, pp. 107–144, 2010.

[8] B. Sarwar, G. Karypis, J. Konstan, J. Riedl *et al.*, "Application of dimensionality reduction in recommender system-a case study," in *ACM WebKDD workshop*, vol. 1625, no. 1. Citeseer, 2000, pp. 285–295.

[9] S. K. L. Al Mamunur Rashid, G. Karypis, and J. Riedl, "Clustknn: a highly scalable hybrid model-& memory-based cf algorithm," *Proceeding of webKDD*, 2006.

[10] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan, "Large-scale parallel collaborative filtering for the netflix prize," in *Algorithmic Aspects in Information and Management: 4th International Conference, AAIM 2008, Shanghai, China, June 23-25, 2008. Proceedings 4*. Springer, 2008, pp. 337–348.

[11] X. Meng, J. Bradley, B. Yavuz *et al.*, "Mllib: Machine learning in apache spark," in *JMLR*, vol. 17, no. 34, 2016, pp. 1–7.

[12] P. Cremonesi, Y. Koren, and R. Turrin, "Performance of recommender algorithms on top-n recommendation tasks," in *Proceedings of the fourth ACM conference on Recommender systems*, 2010, pp. 39–46.

[13] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," *arXiv preprint arXiv:1205.2618*, 2012.