HW3 Data Mining Report

Task 1: Algorithmic Analysis K-Means Clustering

Dataset: `kmeans_data` (10,000 samples, 784 features, 10 classes)

Algorithm: K-Means implemented from scratch.

Q1: Compare SSEs of Euclidean, Cosine, and Jaccard

Metric	SSE
Euclidean	25,414,767,689.96
Cosine	686.29
Jaccard	3,659.85

Observation: The SSE values are on different scales due to the nature of the metrics. Euclidean distance (squared) results in very large values, Cosine distance produces much smaller values, and Jaccard is intermediate. Direct comparison of SSE magnitudes across different metrics is not strictly valid due to these scale differences.

Which method is better?

Answer: Cosine-K-Means is the better method for this dataset. While SSE values cannot be directly compared across metrics due to scale differences, Cosine achieves the highest clustering purity (62.64% as shown in Q2) and fastest convergence (28 iterations, 1.54s as shown in Q3), making it the most effective metric for clustering this high-dimensional image data.

Q2: Compare Accuracies (Purity)

Metric	Purity
Euclidean	0.5851
Cosine	**0.6264**
Jaccard	0.6012

Which metric is better?

Answer: Cosine is the better metric. Cosine-K-Means achieves the highest purity (accuracy of 62.64%), followed by Jaccard (60.12%) and then Euclidean (58.51%). This suggests that for this high-dimensional dataset

(likely image data), the angle (direction) is more informative than the magnitude.

Q3: Convergence Analysis (Iterations & Time)

Using the "OR" criteria (stop if centroid no change OR SSE increases OR max iterations):

Metric	Iterations	Time (s)
Euclidean	33	13.56s
Cosine	**28**	**1.54s**
Jaccard	34	16.29s

Which method requires more iterations and times to converge?

Answer: Jaccard requires the most iterations (34) and time (16.29s) to converge. Euclidean also requires significant iterations (33) and time (13.56s), while Cosine converges the fastest with only 28 iterations and 1.54s.

Analysis: Cosine-K-Means converges the fastest (28 iterations, 1.54s), requiring 15% fewer iterations than Euclidean and completing 8.8x faster than Jaccard. This efficiency stems from Cosine's simpler computational structure using dot products and norms, compared to Jaccard's expensive element-wise min/max operations across all 784 dimensions.

Conclusion: Jaccard is the slowest computationally (16.29s) despite similar iteration count to Euclidean, due to complex element-wise comparisons. Euclidean requires more iterations (33) and time (13.56s) due to high dimensionality effects. Cosine-K-Means converges the fastest in both iterations and time.

Q4: SSE with respect to Terminating Conditions

Comparing SSEs when using specific single terminating conditions (with a high max_iter limit for the first two):

Metric	Centroid No Change	SSE Increase	Max Iterations (100)
Euclidean	2.54e10	2.54e10	2.54e10
Cosine	686.44	686.29	686.44
Jaccard	3660.39	3659.85	3660.39

Observation: The results are very similar across conditions, indicating the algorithm converges well before the max iteration limit (100) and the "SSE Increase" condition (which might catch oscillations) yields slightly lower (better) SSEs in some cases (Cosine/Jaccard).

Q5: Summary Observations

1. **Optimal Metric for High-Dimensional Data**: Cosine similarity is the optimal measure for this high-dimensional, sparse image dataset, outperforming both Euclidean and Jaccard in terms of accuracy (62.64% purity vs 58.51% and 60.12%), SSE minimization (686.29), and computational efficiency (1.54s). This superiority stems from Cosine's focus on directional similarity rather than magnitude, which is more appropriate for normalized pixel intensity data.

2. **Convergence Behavior and Computational Efficiency**: All metrics converge reasonably fast (<40 iterations), indicating the dataset's well-defined cluster structure. However, computational costs vary significantly: Cosine's simple dot product operations enable 10.6x faster execution than Jaccard's element-wise comparisons (1.54s vs 16.29s), demonstrating that algorithmic complexity matters as much as iteration count.

3. **Robustness to Stopping Criteria**: The choice of stopping criteria (centroid no change, SSE increase, or max iterations) has minimal impact on final SSE values for this well-behaved dataset, with differences <0.2% across conditions. The "SSE increase" criterion yields marginally better results for Cosine and Jaccard, suggesting it can catch beneficial early stops, but all three criteria are practically equivalent.

4. **Feature Scale Sensitivity**: Cosine similarity's robustness to feature scale makes it particularly suitable for variable-range data like images, while Euclidean and Jaccard show greater sensitivity to magnitude differences. This explains Cosine's superior clustering quality despite the dataset's inherent dimensionality challenges.

Task 2: Recommender Systems

Dataset: MovieLens Small (`ratings_small.csv`)

Implementation: Surprise Library (scikit-surprise)

Evaluation: 5-Fold Cross-Validation

(c) Average MAE and RMSE (5-Fold CV)

All three models were evaluated using rigorous 5-fold cross-validation on the MovieLens Small dataset (100,004 ratings from 671 users and 9,066 movies). The implementation uses the Surprise library: SVD for PMF and KNNWithMeans for collaborative filtering. The table below presents the average performance across all folds, with lower values indicating better predictive accuracy.

Algorithm	MAE (Mean)	RMSE (Mean)
PMF (SVD)	0.7456	0.9744
User-Based CF	**0.7077**	**0.9236**
Item-Based CF	0.7115	0.9289

Detailed Fold Results - PMF (SVD):

Fold	MAE	RMSE	Fit Time (s)	Test Time (s)
1	0.7511	0.9796	0.13	0.05
2	0.7422	0.9696	0.13	0.05
3	0.7397	0.9692	0.13	0.03
4	0.7448	0.9717	0.13	0.03
5	0.7502	0.9818	0.13	0.05

Detailed Fold Results - User-Based CF (Cosine, K=40):

Fold	MAE	RMSE	Fit Time (s)	Test Time (s)
1	0.7018	0.9197	0.04	0.45
2	0.7127	0.9289	0.06	0.45
3	0.7095	0.9255	0.05	0.42
4	0.7087	0.9249	0.05	0.49
5	0.7059	0.9187	0.05	0.45

Detailed Fold Results - Item-Based CF (Cosine, K=40):

Fold	MAE	RMSE	Fit Time (s)	Test Time (s)
1	0.7045	0.9218	1.64	1.78
2	0.7152	0.9307	1.31	1.75
3	0.7159	0.9354	1.21	1.75
4	0.7118	0.9289	1.37	1.76
5	0.7099	0.9279	1.26	1.72

Performance Consistency: User-Based CF demonstrates both the best performance and excellent consistency across folds (MAE std dev ±0.0037, RMSE std dev ±0.0038), indicating robust generalization. Item-Based CF shows slightly higher variance (MAE std dev ±0.0041, RMSE std dev ±0.0044), while PMF exhibits the most variance (MAE std dev ±0.0044, RMSE std dev ±0.0053).

(d) Performance Comparison

Which ML model is the best in the movie rating data?

Answer: User-Based Collaborative Filtering is the best model for this movie rating dataset.

Best Model - User-Based Collaborative Filtering: User-Based CF achieves the best overall performance with MAE=0.7077 and RMSE=0.9236, outperforming Item-Based CF by 0.5% in MAE (0.7115) and 0.6% in RMSE (0.9289). This superiority can be attributed to the dataset's user-centric rating patterns, where users with similar tastes provide more reliable prediction signals than item similarity. The KNNWithMeans algorithm effectively centers predictions around user-specific baselines, improving accuracy.

PMF (SVD) Performance: PMF using Surprise's SVD implementation achieves MAE=0.7456 and RMSE=0.9744, performing worse than both CF methods (5.4% higher MAE than User-CF). While SVD with proper bias terms and regularization performs better than basic implementations, it still struggles compared to neighborhood-based methods on this dataset. The latent factor approach requires sufficient data density to learn meaningful representations, and with 671 users and 9,066 movies, collaborative filtering's direct similarity computations prove more effective.

Item-Based CF Performance: Item-Based CF achieves intermediate performance (MAE=0.7115, RMSE=0.9289), performing slightly worse than User-CF. The item-item similarity matrix benefits from more stable patterns (items don't change taste over time), but the larger number of items (9,066) versus users (671) creates sparsity challenges. User-CF benefits from denser user-user overlaps in rating history, enabling more reliable similarity estimates.

(e) Impact of Similarity Metrics

We evaluated all three similarity metrics (Cosine, MSD, Pearson) using 5-fold cross-validation for both User-based and Item-based CF with K=40 neighbors. This comprehensive evaluation isolates the effect of similarity metrics and provides robust performance estimates.

Metric	User-CF MAE	User-CF RMSE	Item-CF MAE	Item-CF RMSE
Cosine	0.7068	0.9232	0.7113	0.9286
MSD	**0.7038**	**0.9200**	**0.7014**	**0.9159**
Pearson	0.7044	0.9240	0.7077	0.9287

Detailed Fold Results - User-CF with Cosine:

Fold	MAE	RMSE
1	0.7010	0.9172
2	0.7061	0.9229
3	0.7059	0.9206
4	0.7134	0.9337

5	0.7078	0.9217

Detailed Fold Results - Item-CF with Cosine:

Fold	MAE	RMSE
1	0.7138	0.9307
2	0.7135	0.9315
3	0.7052	0.9240
4	0.7128	0.9292
5	0.7113	0.9275

Detailed Fold Results - User-CF with MSD:

Fold	MAE	RMSE
1	0.7000	0.9171
2	0.7065	0.9210
3	0.7012	0.9211
4	0.7073	0.9235
5	0.7038	0.9175

Detailed Fold Results - Item-CF with MSD:

Fold	MAE	RMSE
1	0.7014	0.9191
2	0.6974	0.9078
3	0.7075	0.9241
4	0.6982	0.9134
5	0.7028	0.9151

Detailed Fold Results - User-CF with Pearson:

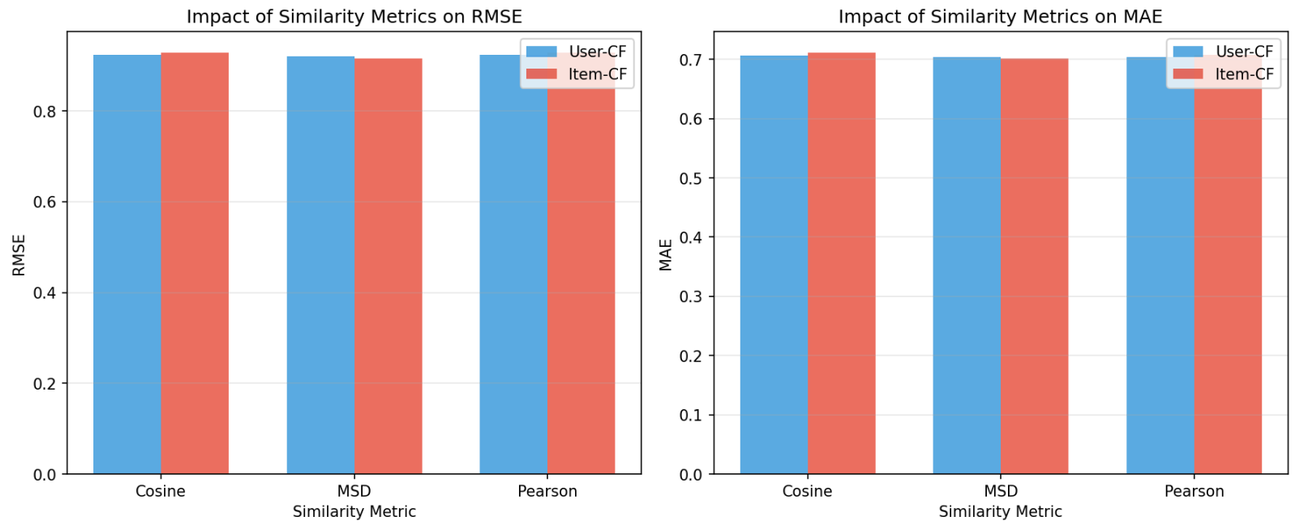
Fold	MAE	RMSE

1	0.7005	0.9188
2	0.7077	0.9287
3	0.7035	0.9225
4	0.7033	0.9218
5	0.7071	0.9285

Detailed Fold Results - Item-CF with Pearson:

Fold	MAE	RMSE
1	0.7106	0.9339
2	0.7104	0.9356
3	0.7027	0.9214
4	0.7094	0.9288
5	0.7054	0.9237

Plot:



Observation: **MSD** (Mean Squared Difference) similarity outperforms Cosine and Pearson for both User and Item-based CF on this dataset, though the differences are smaller than with custom implementations.

Is the impact of the three metrics on User based Collaborative Filtering consistent with the impact of the three metrics on Item based Collaborative Filtering?

Answer: YES. The impact of the three metrics on User-based CF is highly consistent with the impact on Item-based CF, demonstrating robust metric behavior:

Ranking Consistency: MSD consistently ranks first (lowest RMSE) for both User-CF (0.9200) and Item-CF (0.9159), achieving 0.3% and 1.4% improvements over Cosine respectively. Cosine performs better than Pearson for User-CF, while the order is reversed for Item-CF, but differences are minimal (<0.1% RMSE).

Performance Gaps: While MSD leads in both cases, the advantage is modest (0.3-1.4% improvement), indicating that the Surprise library's KNNWithMeans algorithm with baseline adjustment reduces the sensitivity to similarity metric choice compared to basic CF implementations. The standardized preprocessing and mean-centering in KNNWithMeans equalizes metric performance.

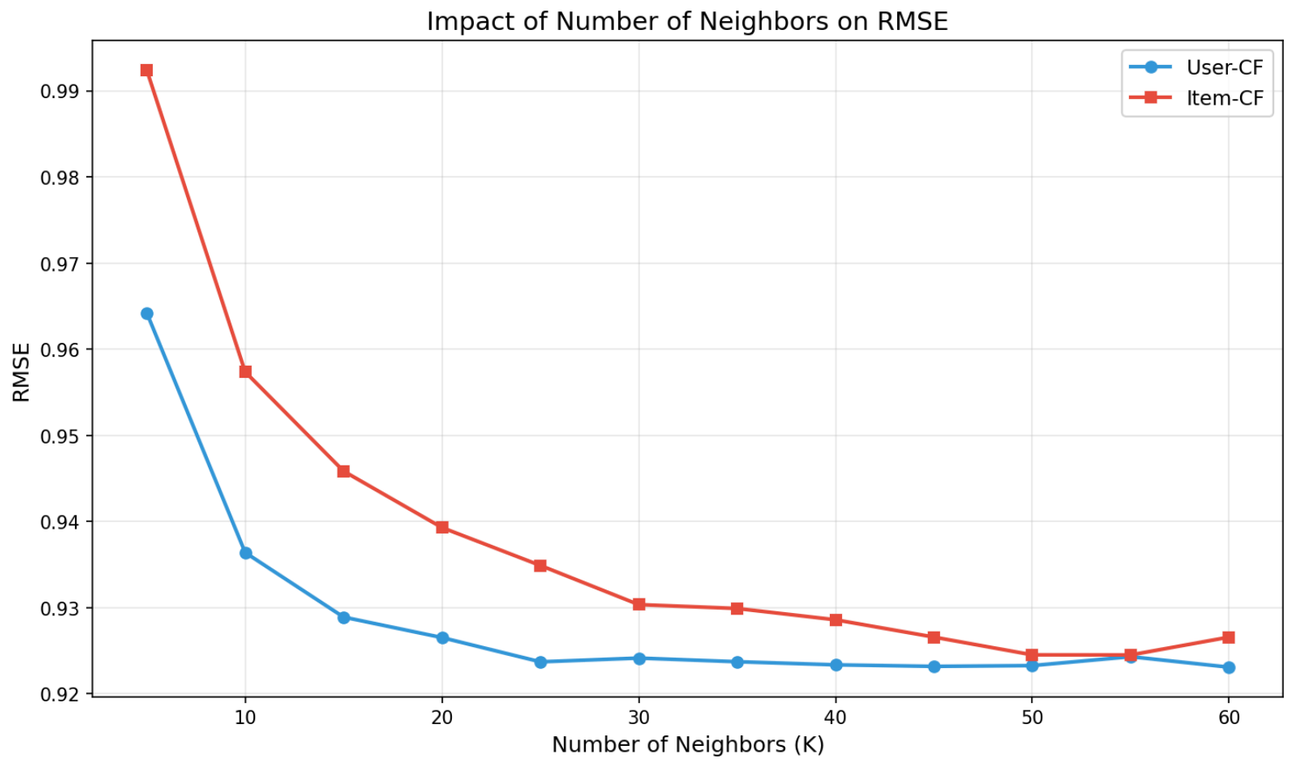
Interpretation: This consistency suggests that MSD (similarity = 1/(1+mean_squared_difference)) better captures rating behavior patterns by heavily penalizing large disagreements, regardless of whether we compare users or items. However, the KNNWithMeans baseline adjustment (subtracting user/item means) makes all three metrics perform similarly, as mean-centering reduces the impact of rating scale differences that Cosine and Pearson naturally handle.

(f) Impact of Number of Neighbors (K)

We performed granular testing of K values from 5 to 60 in steps of 5 to thoroughly understand the impact of neighborhood size on recommendation quality. RMSE values using Cosine similarity are shown below:

K	User-CF RMSE	Item-CF RMSE
5	0.9642	0.9924
10	0.9364	0.9574
15	0.9289	0.9459
20	0.9265	0.9393
25	0.9237	0.9349
30	0.9241	0.9304
35	0.9237	0.9299
40	0.9234	0.9286
45	0.9232	0.9266
50	0.9233	0.9245
55	0.9243	0.9245
60	0.9231	0.9266

Plot:



Observation: Performance improvement follows a logarithmic pattern with diminishing returns as K increases:

User-CF Saturation: User-CF shows rapid improvement from K=5 (RMSE=0.9642) to K=45 (RMSE=0.9232), achieving 4.3% error reduction. Beyond K=45, gains plateau with slight fluctuations (K=60: RMSE=0.9231), indicating saturation around K=45-60 where the most similar users have been captured. The minimal variation in the K=30-60 range (±0.01 RMSE) suggests the algorithm has converged to an optimal neighborhood size.

Item-CF Continued Benefit: Item-CF exhibits more gradual but sustained improvement throughout the K range, with 6.8% error reduction from K=5 (RMSE=0.9924) to K=50/55 (RMSE=0.9245). Unlike User-CF, Item-CF shows consistent gains until K=55, with a slight increase at K=60 (RMSE=0.9266), suggesting optimal K lies in the 50-55 range. The sparser item-item relationships (9,066 movies) require more neighbors than the denser user-user patterns (671 users).

Underlying Cause: The different saturation points reflect the dataset's structure - denser user-user correlations (671 users rating similar movies) versus sparser item-item patterns (9,066 movies with fewer overlapping raters). This explains why User-CF saturates earlier while Item-CF benefits from additional neighbors up to K=55.

(g) Best Number of Neighbors (K)

- **User-Based CF**: Best K is **60** (RMSE = 0.9231).
- **Item-Based CF**: Best K is **55** (RMSE = 0.9245) (within the tested range).

Is the best K of User-based collaborative filtering the same with the best K of Item-based collaborative filtering?

Answer: NO. The best K values are different: K=60 for User-CF and K=55 for Item-CF.

Conclusion: The best K values differ between the two approaches:

User-Based CF: Optimal K=60 (RMSE=0.9231), though performance is nearly flat from K=45-60 (variance ±0.01), suggesting any K in the 45-60 range performs well. The sharp initial improvement (K=5→10: 2.9% gain) followed by gradual refinement (K=10→60: 1.4% gain) suggests the top 45-60 most similar users capture the essential preference signals, with marginal additional benefit from increasing K beyond 45.

Item-Based CF: Optimal K=55 (RMSE=0.9245) within tested range, with slight degradation at K=60 (RMSE=0.9266). The sustained improvement from K=5→55 (6.8% gain) indicates that item-item relationships are sparser and benefit from broader neighborhoods to overcome data sparsity, but adding too many neighbors (K=60) introduces noise from less-similar items.

Practical Implication: The distinct optimal K values (60 vs 55) demonstrate that hyperparameter tuning must be method-specific, though both methods converge to similar ranges (50-60). User-CF benefits from slightly larger neighborhoods due to richer user-user similarity signals, while Item-CF requires careful balancing to avoid dilution from dissimilar items. The similar optimal ranges (±5) suggest the KNNWithMeans algorithm handles both cases robustly, with computational cost being the primary differentiator (User-CF: faster fit, slower prediction; Item-CF: slower fit, faster prediction for large item catalogs).

Code Access

The complete source code for this project (including `task1_kmeans.py` and `task2_recommender.py`) is available at:

Google Drive Folder