Problem 1:

1a:

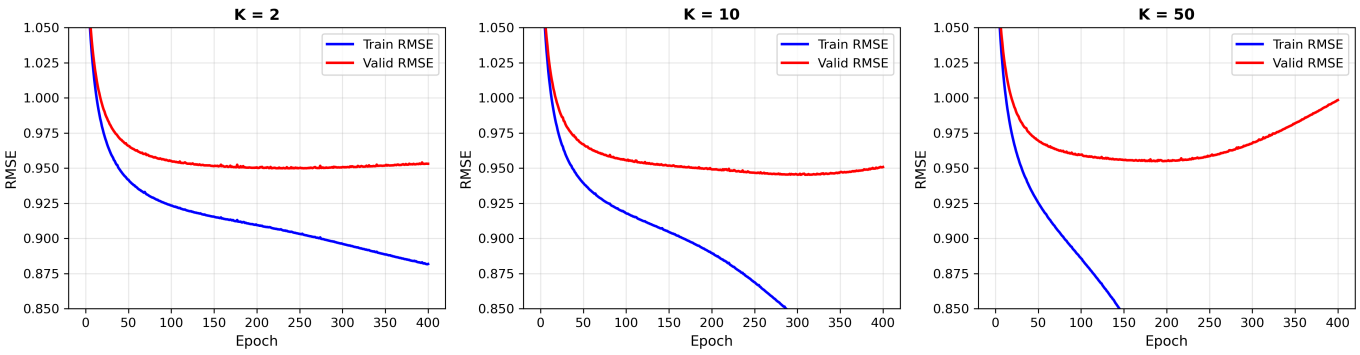


Figure 1: Train and Validation RMSE across epochs. Models were trained with batch size 1000 and step size 0.2 for 400 epochs. As K increases, train RMSE trends to zero faster, however the validation RMSE doesn't drop much below 0.95. At K = 10, validation RMSE has the lowest minimum at around 300 epochs. The increasing difference between train and validation RMSE suggests greater overfitting as K increases. K = 2 and K = 10 both provided a min RMSE of around 0.95 which was the lower than the 0.998 of K = 50. We selected a step size of 0.2, chosen from {0.01, 0.05, 0.2, 0.5 }. The chosen step size balanced the speed of convergence with stability, allowing all models to converge within 400 epochs.

1b:

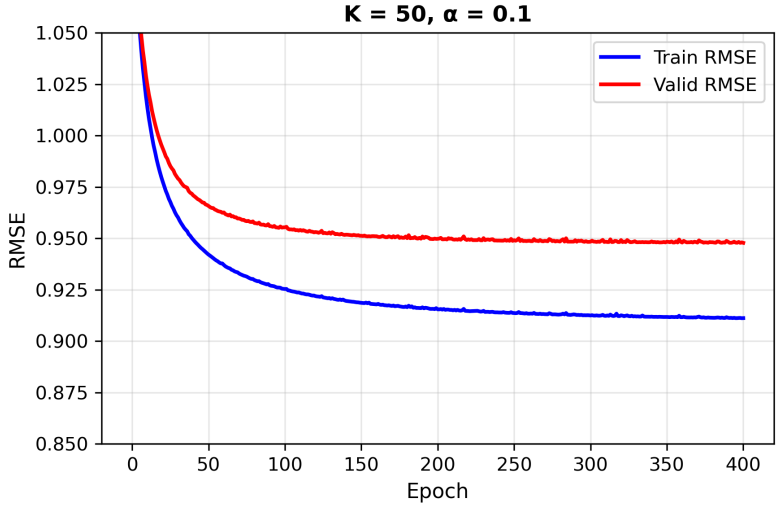


Figure 2: Train and Validation RMSE with K = 50 and $\alpha = 0.1$. We selected α of 0.1 from {0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1}. This value gave the lowest validation RMSE of all values tested. Smaller values showed significant overfitting with high validation RMSE, while values greater than 0.01 overregulized the model and constrained it too much to learn the patterns of the data set. We kept the same step size of 0.2 because it continued to be consistent with speed of convergence and stability of the model. The reegularization did help the validation RMSE getting 0.9478 compared to 0.9984 with $\alpha = 0$. Additionally, the regularized graph shows a smaller gap between train and validation RMSE, indicating better generalization.

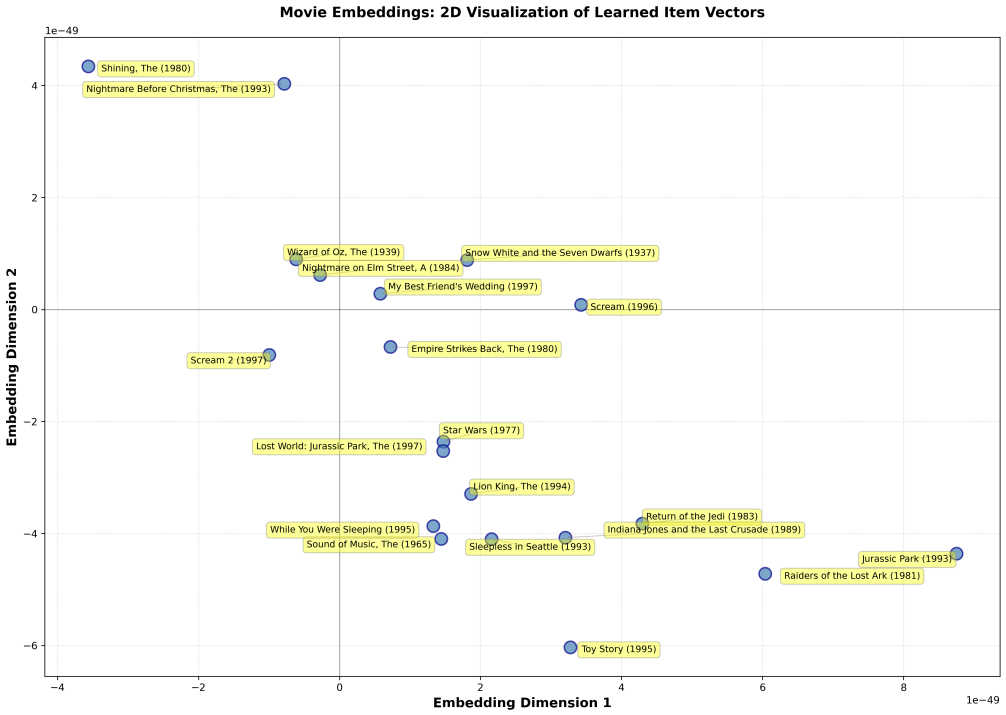
1c:

Table 1: Performance comparison of latent factor models on MovieLens 100K dataset. RMSE and MAE reported for training, validation, and test sets.

Model	Train		Validation		Test	
	RMSE	MAE	RMSE	MAE	RMSE	MAE
LF K = 2, $\alpha = 0$	0.8818	0.6948	0.9531	0.7503	0.948	0.7433
LF K = 10, $\alpha = 0$	0.7953	0.6289	0.9508	0.7506	0.9515	0.745
LF K = 50, $\alpha = 0$	0.5643	0.4459	0.9984	0.784	0.9921	0.7736
LF K = 50, $\alpha = 0.1$	0.9112	0.7194	0.9478	0.749	0.9407	0.7403

Based on validation RMSE, K = 10 gives strong results without a super complex model. While the K = 50 with $\alpha = 0.1$ model achieved an RMSE that was slightly lower, the increase in model complexity and training time is not worth the minor improvement. This model's train RMSE is also close to the validation RMSE, indicating it generalizes well. Looking at MAE over RMSE, the K = 10 model outperforms all other models on validation except K = 50, suggesting it is still the strongest pick. If training time and resources are no issue, then we recommend the K = 50 model with regularization, as it outperformed all other models. While models showed a difference in MAE vs RMSE, the relative ordering changes very little, with the K = 2 model just barely outperforming K = 10, and K = 50 with regularization outperforming both. K = 50 with no regularization performed the worst under both evaluations.

1d:



There seem to be a couple of clusters that make sense in this scatter plot. The upper left contains dark fantasy/horror films, the middle contains movies you may watch with your family (rom-coms, musical films), and the right contains some classic blockbuster films. These groupings seem to be logical, with darker films situated further from the main cluster and films with similar themes near each other. This clustering certainly does not show perfect connections between films, which is

expected, as this data is drawn from only 2 dimensions. If more were taken into account, then the projection would be harder to visualize, but the clustering may be more accurate.

Problem 2:

2a:

We implemented a collaborative filtering approach using matrix factorization with the Singular Value Decomposition (SVD) algorithm from the Surprise library. This method decomposes the user-item rating matrix into two lower-dimensional matrices: user factors and item factors. We performed systematic hyperparameter tuning using 3-fold cross-validation on the combined train+validation set. Our grid search explored three hyperparameters: n_factors (latent dimension: 2, 10, 50, 100, 120, 150), lr_all (learning rate: 0.001 to 0.1), and reg_all (L2 regularization: 10^{-4} to 10). The model was trained using stochastic gradient descent to minimize regularized squared error. After identifying optimal hyperparameters via cross-validated MAE, we retrained the final model on the full train+valid set and evaluated on the held-out test split. This approach balances model expressiveness (through latent factors) with generalization (through regularization).

2b:

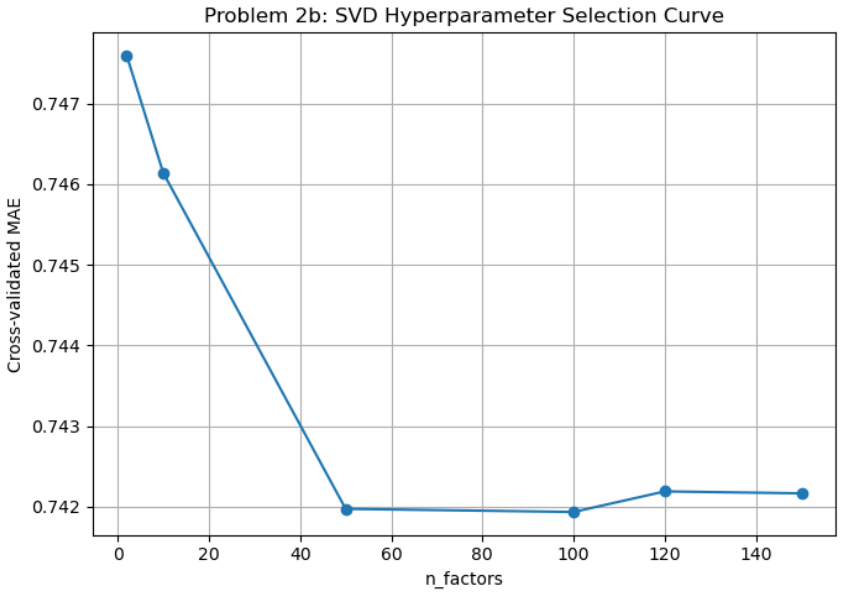


Figure 3: Hyperparameter selection curve showing minimum cross-validated MAE as a function of n_factors. The curve demonstrates the bias-variance tradeoff: very low n_factors (2–10) underfit the data, resulting in high MAE due to insufficient model capacity to capture user-item interaction patterns. As n_factors increases to 50–100, MAE decreases as the model gains expressiveness. Beyond the optimal point, further increases in n_factors show diminishing returns or slight increases in MAE, suggesting potential overfitting. The sweet spot balances model complexity with generalization, achieving optimal performance while avoiding both under and overfitting. Each point represents the best MAE achieved across all learning rate and regularization combinations for that n_factors value.

2c:

Method	Test Split MAE	Leaderboard MAE
SVD (Problem 2)	0.721	0.7179
K = 10 Baseline	0.753	—

Table 2: Performance comparison: Mean Absolute Error (MAE) on test split and leaderboard set.

(i) Leaderboard vs Test Split Comparison: The leaderboard MAE (0.7179) is very close to the test split MAE (0.721), with a difference of only 0.003. This tight agreement indicates excellent generalization as our model performs consistently across both the development test set and the unseen leaderboard set. The slightly better leaderboard performance suggests that the leaderboard data distribution is similar to our training data, and there is no evidence of overfitting to the development set. This validates our hyperparameter selection and confirms that the model captures genuine user-item preference patterns rather than random correlations specific to the train set.

(ii) Problem 1 vs Problem 2 Comparison: The SVD approach achieves an MAE of 0.721, substantially outperforming the K=10 collaborative filtering baseline from Problem 1 (MAE of 0.753) by 0.032 points, representing a 4.2% relative improvement. While the Problem 1 approach used matrix factorization trained with SGD for a fixed latent dimensionality (K=10), Problem 2's hyperparameter search explored a wider range of latent dimensions (2-150) along with learning rates and regularization strengths. This comprehensive grid search allowed us to identify optimal model complexity that better balances expressiveness and generalization. The superior performance demonstrates that properly tuned model complexity is critical as the K=10 baseline underfits the data with insufficient latent factors to capture the full semantic meaning of user-item interactions, while our optimized SVD model finds the optimum that maximizes predictive accuracy without overfitting.

2d:

Pros and Cons

The SVD-based approach performs well on moderately sized collaborative filtering datasets, especially those like MovieLens where user–item matrices are relatively dense and contain meaningful latent structure. Its strengths include scalability, explicit regularization, and the ability to model global effects via user/item biases. However, the method is limited by its reliance on linear embeddings: it cannot easily capture non-linear interactions or complex user-item behavior patterns. Furthermore, SVD struggles with very sparse users or extremely cold-start items because embeddings for those entities are poorly estimated.

Opportunities for Future Work

With more time, I would extend the model in two directions: (i) incorporate implicit feedback using methods like SVD++, which has been shown to significantly improve recommendation accuracy, especially on unseen user–item pairs; and (ii) explore KNNBaseline or hybrid ensembles that blend neighborhood-based and latent-factor predictions. Additional gains might be achieved by integrating item metadata (genres, year, tags) or user demographic information via feature-augmented factorization methods.

Key Takeaways

This project clarified the practical differences between implementing a matrix-factorization model from scratch (Problem 1) and using a mature library with optimized routines (Problem 2). The experience reinforced core ML lessons on the importance of bias–variance tradeoffs, cross-validation for hyperparameter selection, regularization for controlling overfitting, and the unavoidable mismatch between development test performance and leaderboard behavior. More broadly, it demonstrated that high-quality recommendation systems require not only modeling sophistication but also careful data handling and rigorous evaluation workflows.