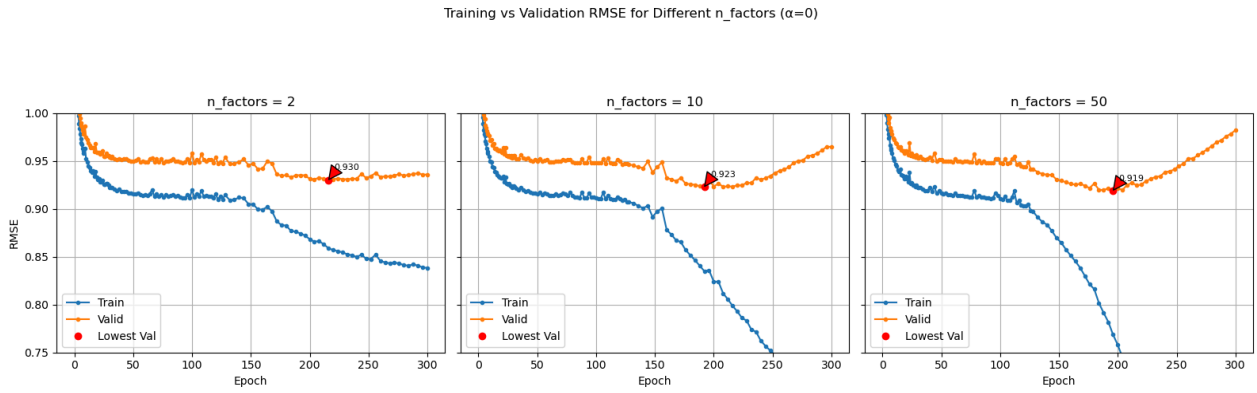
Movie Recommendations Project

Edward Lai

Figure 1a:



Caption 1a:

(i) What happens as K increases in terms of under- or over- fitting?

As K increases, the model starts overfitting because the training RMSE values get lower at larger K values. Meanwhile, the validation RMSE is higher at larger K values, indicating a loss of generalization on unknown data.

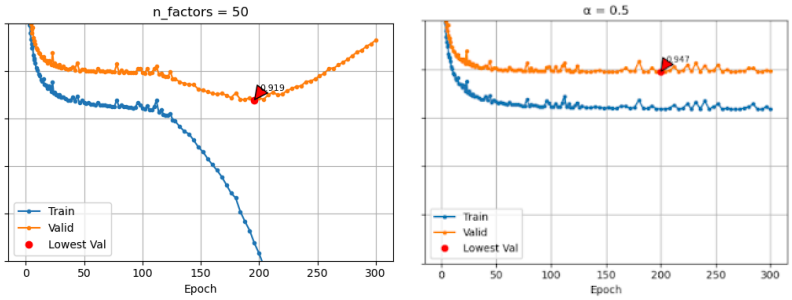
(ii) How does the 'best' validation-set performance change from K=2 to 10 to 50?

The “best” validation-set performance decreases slightly as we increase the K value. At K=2, the lowest validation RMSE is 0.930, at K=10 the lowest validation RMSE is 0.923, and at K=50 the lowest validation RMSE is 0.919. This makes sense because increasing K should make the model more accurate at some point compared to models with smaller K values before beginning to overfit as training goes on.

(iii) What step size? Why did you pick that value?

We chose step size 0.8 because it was at this step size that the graphs started to look more stable and not diverge. We started by testing larger step size values such as 5.0 and 2.0 which showed divergence, then incrementally decreased the step size to get to a point where there was no divergence and the graph did not have many sharp peaks/valleys, which happened around 0.8.

Figure 1b:



Caption 1b:

(i) What value of alpha? How did you select it? (you should try several values)

We picked an alpha value of 0.5. We selected it by generating plots for several different alpha values: [0.0, 0.1, 0.3, 0.5, 1.0], using different step sizes as well. We observed that at alphas 0.0, 0.1, and 0.3 the training RMSE was still decreasing as training went on which meant that the model was still overfitting. Alpha 0.5 was the point at which the training RMSE plateaued and wasn't decreasing as training went on, so the model didn't overfit.

(ii) What value of step size did you pick?

We picked a step size of 0.8. We selected it by generating plots for several different step size values: [0.7, 0.8, 0.9, 1.0]. We observed different combinations of alpha and step size pairs and saw that at alpha=0.5 the model wasn't overfitting, and step size 0.8 was when the model started looking more stable without divergence and large peaks/valleys in the graph (compared to the larger step sizes of 0.9 and 1.0).

(iii) Did you get better validation-set error with this alpha than you did with the K=50, alpha=0 result in 3a?

The lowest validation RMSE value with alpha=0.0 (0.919) is still lower than the lowest validation RMSE value with alpha=0.5 (0.947). However, as training goes on the validation RMSE with alpha=0.0 gets worse due to overfitting while validation RMSE with alpha=0.5 does not decrease and stays relatively the same. This means that the final validation error is better with alpha=0.5 than alpha=0.0, meaning that model has better generalization.

Table 1c:

K	Alpha	Best RSME	Best MAE
2	0	0.930	0.731
10	0	0.923	0.724
50	0	0.919	0.725
50	0.5	0.947	0.746

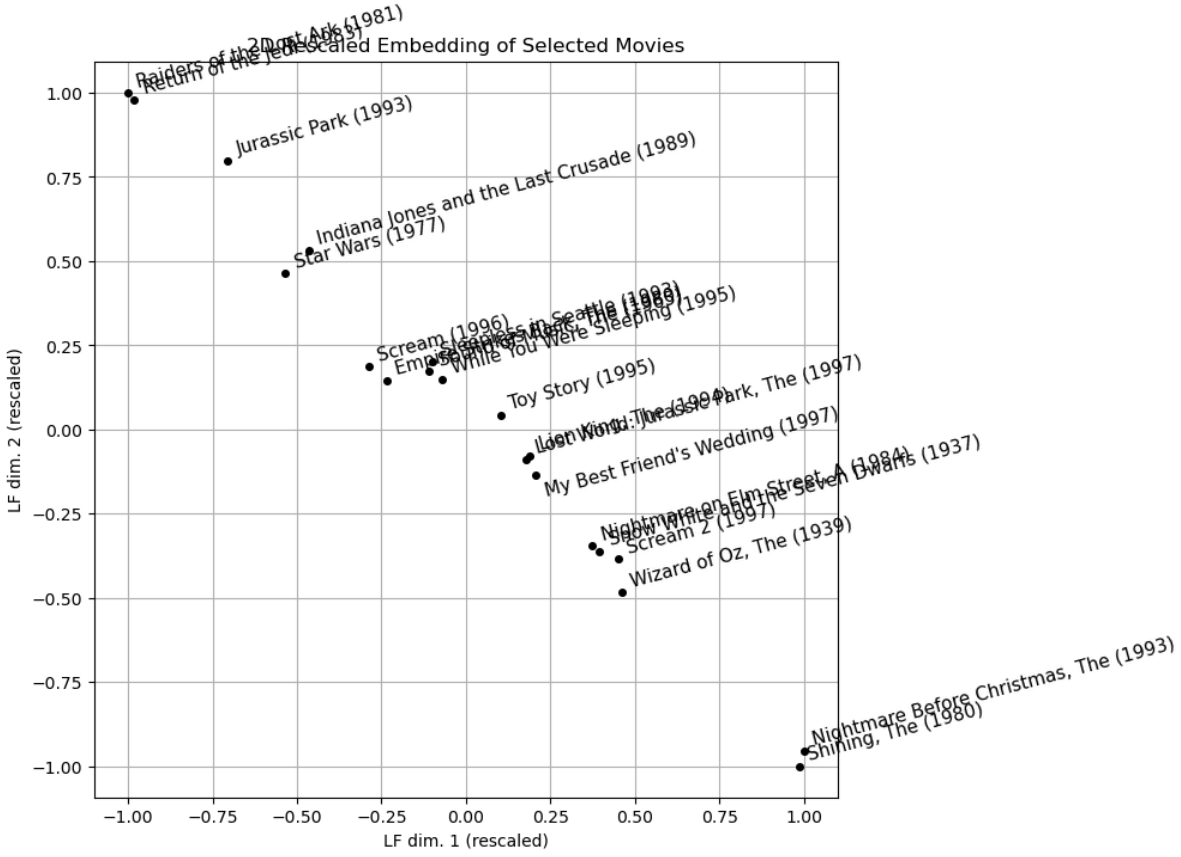
Caption 1c:

(i) Focusing on RMSE, how many factors K do you recommend? (ii) Does model ranking change if you were to use MAE instead of RMSE?

K=50 because it has the lowest RMSE value of 0.919 when alpha=0.

Yes, the model ranking changes to K = 10 being the best model when using MAE instead of RMSE because that model has the lowest MAE of 0.724.

Figure 1d:



Caption 1d: Do you notice any interpretable trends? What makes sense? What does not make sense to you?

One trend is that movies of similar genres seem to be grouped together. For example, in the upper left corner there are many action/adventure movies such as *Raiders of the Lost Ark, Return of the Jedi, Jurassic Park, Indiana Jones,* and *Star Wars.* In the bottom right corner, there are some spooky movies like *Nightmare Before Christmas* and *The Shining.* The middle contains several romance and comedy movies, and children's movies like *Toy Story, Lion King, etc.* are also close together. It makes sense that the movies would be grouped by genre because they would share similar traits and patterns that the model can pick up on, and also users who like one movie in a genre are likely to rate others similarly, which encourages clustering.

However there are some outliers that go against that trend, such as the *Scream* movies and *A Nightmare on Elm Street* which are scattered around the middle of the graph. Another thing that's strange is that movies in the same franchise aren't that close together, like the *Indiana Jones, Star Wars, Jurassic Park,* and *Scream* movies.

2a. For this project, we used the SVD++ algorithm from the Surprise Library to predict user ratings. SVD++ works by extending SVD, which uses matrix factorization to decompose the original matrix into three matrices to reduce dimensions and make the data easier to interpret. SVD++ takes into account implicit ratings by incorporating which items a user interacts with. Because the MovieLens dataset is relatively sparse, SVD++ is a good choice because it can incorporate both explicit ratings and implicit feedback from which movies users rated/interacted with. To optimize model performance, we performed hyperparameter tuning using RandomizedSearchCV over the following parameters:

- n_factors (number of hidden factors) = [150, 175, 200, 225, 250]
- lr_all (learning rate for all parameters) = [0.010, 0.012, 0.013, 0.014, 0.015, 0.016]
- reg_all (regularization strength for all parameters) = [0.05, 0.07, 0.08, 0.1, 0.12]

We also performed 50 iterations using random combinations of the hyperparameters, ran RandomizedSearchCV with all cores to speed up the search, set random state = 42 for debugging purposes, and used a 5-fold cross-validation strategy to ensure reliable performance estimates and reduce any risk of overfitting. For a 5-fold CV, each fold size is approximately 20,000 ratings (100,000 divided by 5) with the training set for each fold as 80,000 ratings and the validation set for each fold as 20,000 ratings. Each fold is split evenly at random. Our evaluation metric was MAE, which measures how closely predicted ratings match actual ratings on average. After retrieving best hyperparameters from RandomizedSearchCV, we retrained the final model on the full ratings masked leaderboard set. All predictions were clipped to valid rating range [1, 5] and additionally we rounded the predicted ratings to the nearest integer after clipping. We saw an improved leaderboard MAE performance with rounding, going against our initial intuition to include many decimal points for prediction precision. The improvement from the rounding to nearest integer is likely due to the true ratings also being integers, so rounding would prevent unnecessary penalty from small prediction errors.

Open-source tools used include:

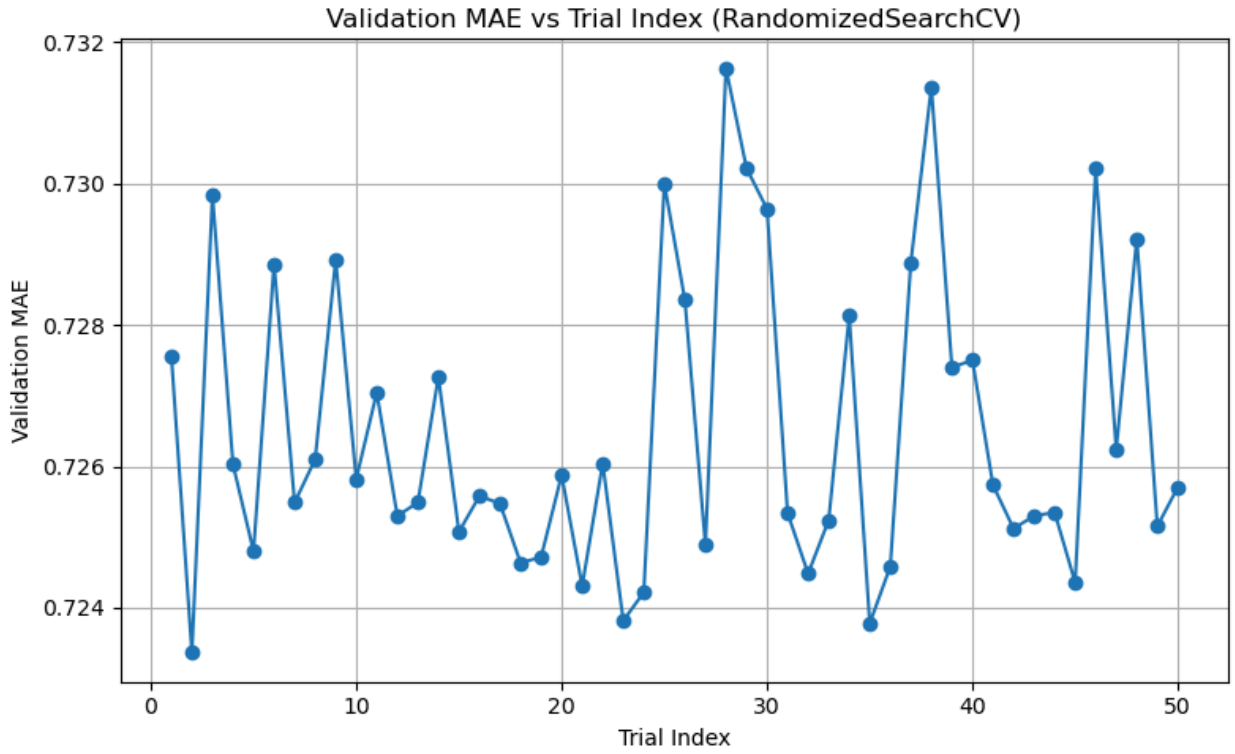
- Surprise (SVD++ model, hyperparameter tuning)
- Pandas and NumPy for data manipulation
- Matplotlib for visualization

References:

https://surprise.readthedocs.io/en/stable/matrix_factorization.html

https://en.wikipedia.org/wiki/Singular_value_decomposition

2b.



Caption: Each point corresponds to a different combination of hyperparameters (n_factors, lr_all, reg_all). This figure shows that different configurations can significantly impact valid MAE. Additionally, this reflects the importance of hyperparameter tuning and shows no signs of overfitting or underfitting during model selection. The selection returned n_factors=150, lr_all=0.012, reg_all=0.07 with the lowest MAE value of 0.7233. The MAE varies between approximately 0.723 and 0.732 across trials, showing no underfitting because some hyperparameter combinations do perform well and better than others. There is no indication of overfitting, as the validation MAE remains stable across trials without extreme fluctuations or outliers with very low values.

2c.

(i) Discuss how your leaderboard number compares to performances on the provided test split of the dev set.

The SVD++ model achieved a dev set test MAE of 0.4899 but the leaderboard MAE was 0.668 which is a noticeable increase. This gap suggests some degree of overfitting to the dev set. This could be due to the leaderboard having noisier data than our development test set. Another reason for this could be that the leaderboard contains users or items that were not well-represented in the development set, making predictions harder.

(ii) Compare contrast Problem 1 and Problem 2 solutions

Problem 1 focused on matrix factorization methods which optimizes mainly the number of latent factors k. In contrast, problem 2 incorporated implicit feedback and was fine tuned over multiple hyperparameters (latent dimensions, learning rate, and regularization strength) with the addition of cross-validation to ensure reliability of results. As a result, problem 2 solution generalized much better and achieved a lower MAE both in development testing set and on the leaderboard.

MAE Performance Table

Model	**Dev Set Test MAE**	**Leaderboard MAE**
SVD++	0.490	0.6684
K=10 MF	0.724	N / A
GradientBoosting	0.533	N / A
XGBoost	0.512	N / A

2d. 1 paragraph discussing the overall pros and cons of your current approach. What other kinds of recommendation problems would it work well for? What are its limitations? What kinds of data/problems would this approach not work well for?

The SVD++ model worked well because it used both explicit ratings and implicit user-item interactions. This led to making it very effective for sparse, user-item matrix recommendation problems like MovieLens. The main strength of SVD++ lies in its ability to generalize reasonably well to unseen data when properly tuned. However, the biggest con for SVD++ was its expensive computation. It took around 20 minutes of compute time to complete RandomizedSearchCV and achieve the lowest validation MAE. Furthermore, one limitation of SVD++ is that it is slow to train. This is amplified when we increase the number of hyperparameter trials or increase the size of the data set. Using this approach, this model assumes that user preferences can be captured through linear interactions of latent factors; this may not be well-suited for very complex datasets. Additionally, this model would not work as well for problems where users or items have no interactions.

1 paragraph reflecting on this project: What are the key takeaway lessons you learned?

My biggest key takeaway lessons that I learned is how critical hyperparameter tuning, cross-validation, and model selection is to achieving a strong generalization in machine learning. With countless terrible leaderboard submissions, this project also helped highlight that a good dev set performance does not always guarantee leaderboard success. I can aim for better generalization by being wary about the risks of overfitting and having thoughtful considerations on how to handle model complexity. I also learned some very cool preprocessing steps techniques such as clipping and rounding predictions that can have a huge impact on certain evaluation metrics. Overall, this project helped me understand recommendation system pipelines, the common pitfalls in modelling, and several practical evaluation techniques.