Recommendation Systems

Jonathan Hu [1]

Abstract

This paper presents a movie recommendation system using the MovieLens dataset, which learns user and item embeddings, models user and item biases, and incorporates movie features to improve recommendations quality.
Code: github.com/hxxjon002/AML-Project

1. Introduction

Recommendation systems are a core component of modern online platforms, enhancing user experience, driving engagement, and increasing revenue by delivering personalized content. This paper aims to develops and evaluates a movie recommendation system using the MovieLens dataset. We begin with a baseline model that accounts for user and item biases, and then extend it by incorporating trait vectors for both user and items, as well as additional features such as movie genres to capture item similarities. We also address practical challenges, including the cold-start problem for new users and items, the identification of polarizing movies, and the integration of implicit feedback and A/B testing.

2. Background

Early recommendation systems were largely based on generic metrics such as popularity and lacked personalization. Many employed user-based collaborative filtering, which recommended items based on similarities between users. In the late 1990s, Greg Linden at Amazon developed item to item collaborative filtering, which calculated similarities between items instead. This approach allowed the most expensive computations to be done offline and scaled efficiently to extremely large datasets. A/B testing revealed that this method significantly increased cross-purchasing, establishing recommendation systems as not just a user experience enhancement but also a revenue driver.

In 2006, Netflix initiated a global competition with the goal of achieving a 10% improvement in RMSE for their recommendation algorithm. With a prize of USD 1 million, the competition galvanized teams around the world to engage in deep research into recommendation systems. The final winner emerged in 2009, BellKor's Pragmatic Chaos, a collaboration of several top performing teams. Their solution combined over 100 different models, and their results cemented Alternating Least Squares and Stochastic Gradient Descent as dominant paradigms for collaborative filtering, due to ability to represent both users and items in a shared low dimensional latent space, capturing complex interaction patterns efficiently.

These approaches were soon adopted as industry standards. The Xbox 360 platform, for example, built a recommendation system that split its work into two parts. The offline component handled computationally heavy tasks such as processing large volumes of user interaction data, training the model, and precomputing recommendations. The online component delivered these recommendations in real time to users, blending them with up-to-date contextual information and applying business rules such as age restrictions or promotional content. This offline–online architecture allowed Xbox to combine the accuracy of complex models with the speed and responsiveness required for a live, global service.

3. Dataset and Pre-processing

The ratings data was first imported and organized using Python dictionaries and lists. Each unique user ID and movie ID was mapped to an internal index, and each user–item–rating triplets were collected into per-user and per-item lists, effectively building two sparse matrix simultaneously. While straightforward to construct, this representation was inefficient for large scale data because it relied on nested Python lists and repeated dictionary lookups. To improve efficiency, the data was restructured into a single NumPy array with a pointer array that records the start and end positions for each user or item, allowing all of a user's or item's ratings to be retrieved much more efficiently.

The distribution of movie ratings are shown in Figure 1. The majority of ratings fall between 4 and 5 stars,as users tend to watch and rate movies they already expect to enjoy. Consequently, highly rated movies are more likely to be recommended to, and attract additional viewers, reinforcing

[1]University of Stellenbosch, Cape Town, South Africa. Correspondence to: Jonathan Hu <29601304@sun.ac.za>.

their popularity and further increasing the volume of ratings.

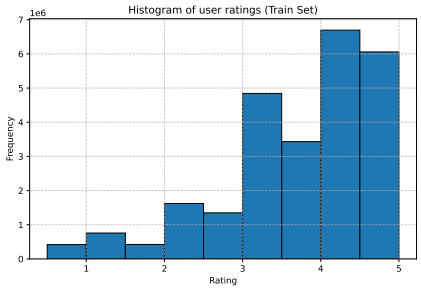

Figure 1. Rating Distribution

Figure 2 illustrates the power law distributions in the data. Both user rating activity and movie popularity exhibit heavy tailed behavior consistent with a power law. The majority of users provide only a handful of ratings, while a small group of individuals contribute thousands. Similarly, most movies receive very few ratings, whereas a limited number of blockbuster titles receive large number of ratings.

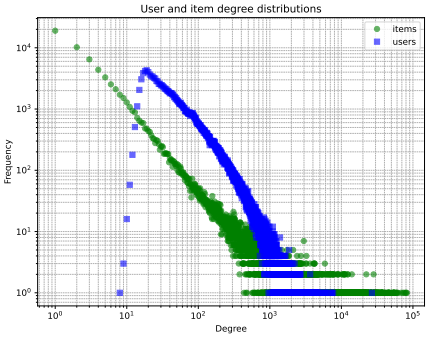

Figure 2. Power Law of the training set

4. Results

All models were trained using a 80/20 train/test split to allow for validation of generalization and hyperparameter tuning. The final outputs are based on the retrained model with the entire 32M data set to maximize the use of available data. All models utilizes the alternating least square algorithm, in which all components are updated iteratively.

The hyperparameters used to train the models are listed in table 1. Although the use of Optuna found sets of parameters that returned a lower test RMSE, the outputs of embedding plots and recommendation were not optimal, as higher dimensions of K and higher values of τ lead to recommendations being concentrated on a small set of blockbuster movies, and lacked personalization. Ultimately, the simpler hyperparameter setting offered the best trade-off between recommendation quality and training time.

Table 1. Hyperparameters used for model training

PARAMETER	VALUE
γ	0.1
λ	0.05
τ	0.1

4.1. ALS with bias only

The first model we consider is a bias-only baseline, trained to identify deviations in the data that are not explained by user item interactions. This model captures two types of biases:

User Bias (b^u) - reflects the tendency of certain users to consistently provide ratings that are higher or lower than the global average. For example, some users may be generally more generous in their evaluations, while others are more critical.

Item bias (b^i) - represents the overall popularity or unpopularity of a particular movie. Highly popular blockbuster films typically receive higher ratings across the board, whereas niche or less successful titles have much lower overall ratings.

The model predicts rating from a specific user m for a specific movie n as:

$$r_{mn} = b^u_m + b^i_n$$

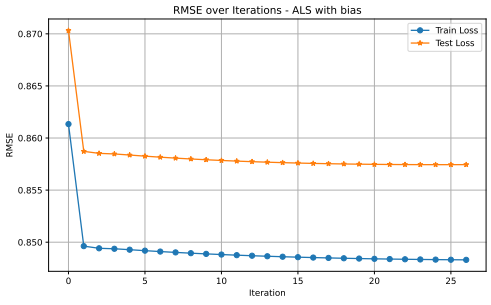

Figure 3. RMSE of ALS model with bias only

4.2. ALS with bias and embeddings

The second model includes biases and latent embeddings for both users and items.

Each user m is represented by a latent vector u_m, which captures the user's preferences. If a user consistently rates kids

movie highly, the model would move the user embedding closer to the region in which the kids genre and represented in the latent space. Similarly, if the user dislike horror movies, the model will shift the user embedding away from that region. Through training, the user embedding captures the user's preference in hidden dimension.

Each item n is represented by a latent vector v_n, which captures underlying characteristics of the movie. During training, the model will encode information such as genre, production year or casts into the item vector. As a result, items with similar properties or common liked by the same users will be positioned close to one another in the latent space.

The latent dimension K determines how many hidden factors the model can learn from the data. vectors. Choosing an appropriate value of K is important to capture model complexity while avoiding overfitting. From experiment, low values of K (5 - 8) was not enough to capture the hidden complexity in user preferences and item characteristics. Too high values of K (15 - 20) resulted in overfitting and significantly increased training time. The best value that balanced results accuracy and training efficiency was K = 10

Latent embeddings allow the model to discover interaction between any users and movies by taking the dot product $u_m^\top v_n$. The updates for user bias, item bias, user and item embeddings are shown in technical appendix A.2.

The model predicts rating as:

$$r_{mn} = b_m^u + b_n^i + u_m^\top v_n$$

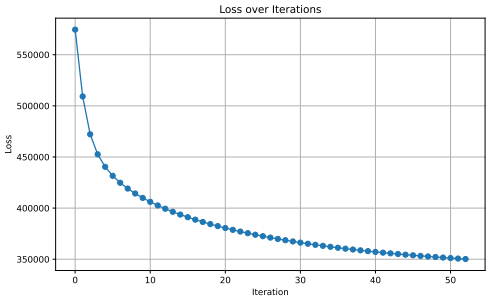

Figure 4. Train loss of model with bias and embeddings

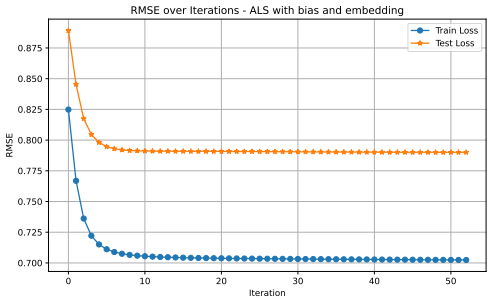

Figure 5. RMSE of ALS model with bias and embedding

Figure 6 shows the 2D visualization of the movie embeddings, where movies from various genres are highlighted. It can be seen that similar movies (Those in the same genre, although genre information has not been included yet) are encoded close to each other in the latent space, while movies with very different genres (such as RomCom vs Horror) and encoded opposite to each other.

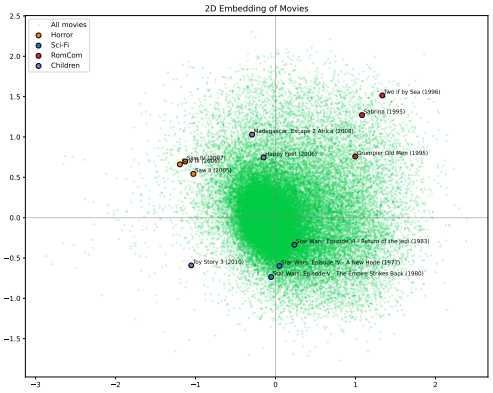

Figure 6. 2D embedding of the item vectors

4.3. ALS with bias, embeddings and features

The last model includes biases, latent embeddings and item side features such as genre information. Movie genre embeddings are encoded in the same latent space. The addition of features allow every movie to not only be represented by its own latent embedding but also by additional feature embeddings that capture its associated metadata. When there are limited ratings data available, the model can still approximate a movie's embedding by its genre, effectively addressing the cold-start problem. The updates for user bias, item bias and user embeddings remain the same, the derivation for the item embedding and feature embedding are shown in technical appendix A.3.

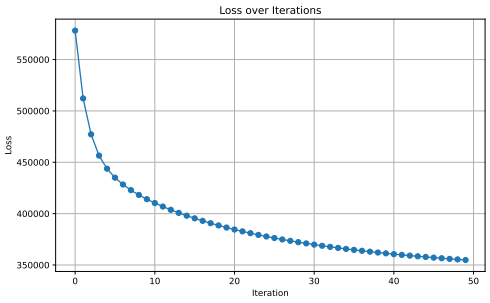

Figure 7. Train loss of model with bias, embeddings and features

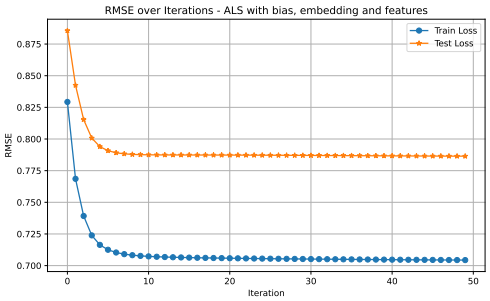

Figure 8. RMSE of ALS model with bias,embedding and features

Figure 9 illustrates the 2D projection of the trained feature embeddings. Common genres across main stream movies are positioned close to the center, In contrast, more distinct genres such as Horror, Animation, and Drama are pushed further out into their own regions.

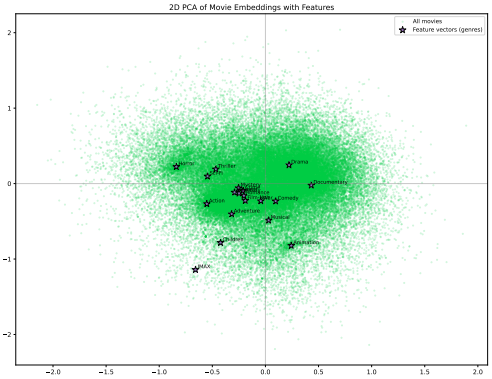

Figure 9. 2D embedding of the feature vectors

Figure 10 illustrates the movie embeddings with the addition of features. Toy Story 3, which was previous embedded far away from other children's movies is now located much closer in the embedding space. Similarly, the Star Wars films and the Saw franchises are also encoded much more tightly.

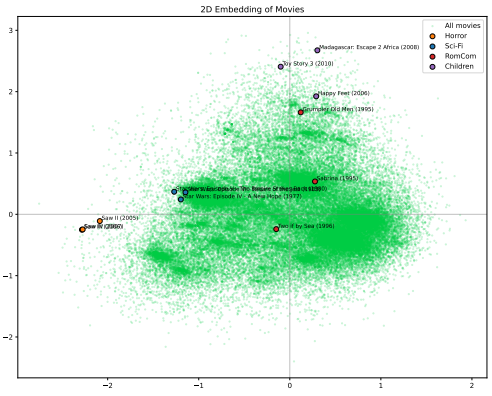

Figure 10. 2D embedding of the feature vectors

4.4. Recommendation

To test the model, we construct a dummy user to simulate a new viewer with no history or preference. A maximum score is then given to any movie in order for the model build the user vector by iteratively updating it against the fixed item embeddings and biases learned during training.

Some examples of the output are shown below:

Table 2. Top 5 Recommendations for a *Lord of the Rings* Fan

RANK	MOVIE
1	LORD OF THE RINGS: RETURN OF THE KING
2	LORD OF THE RINGS: TWO TOWERS
3	HOBBIT: DESOLATION OF SMAUG
4	STAR WARS: EPISODE III – REVENGE OF THE …
5	HOBBIT: BATTLE OF THE FIVE ARMIES

Table 3. Top 5 Recommendations for a *Cars* Fan

RANK	MOVIE
1	CARS 2
2	SHREK THE HALLS
3	ICE AGE: DAWN OF THE DINOSAURS
4	ICE AGE 4: CONTINENTAL DRIFT
5	SHREK FOREVER AFTER

Table 4. Top 5 Recommendations for a *Music in the Air* Fan

RANK	MOVIE
1	BEACHES
2	CAN'T STOP THE MUSIC
3	LITTLE INDIAN, BIG CITY
4	THE HAPPIEST MILLIONAIRE
5	TROOP BEVERLY HILLS

Music in the Air (1934) is a Romance Comedy movie with musicals, because it is a cold start movie, the system would recommend movies based on the genres. As shown by the results, the recommended movies are in the comedy, musical, or drama genres.

4.5. Polarizing Items

Blockbuster movies are often consistently well rated by the majority of viewers, while certain other films are broadly and uniformly disliked. In contrast, some movies receive sharply divided opinions: some viewers love them, while others strongly dislike them. These are referred to as polarizing movies.

A polarizing movie is characterized by a high variance in its ratings distribution. In the embedding space, such items are associated with long trait vectors V. This is because the divergence in ratings cannot be explained by the item bias alone, the model must stretch the item vector far into opposing directions to separate users into the "liked" and "disliked" regions of the latent space.

Polarizing movies are therefore highly informative for recommendation systems. Once the system observes whether a viewer liked or disliked such a movie, it can quickly narrow down their preferences and locate them within the embedding space. This strong discriminative power is not possible with blockbuster movies, which, despite their popularity, provide weaker signals since most viewers tend to agree on their quality.

Table 5 and 6 lists the most polarizing movies and their respective trait vector norms from the MovieLens dataset.

Table 5. Top 10 Most Polarizing Movies

RANK	MOVIE
1	TWILIGHT SAGA: BREAKING DAWN – PART 1
2	TWILIGHT SAGA: BREAKING DAWN – PART 2
3	TWILIGHT SAGA: ECLIPSE
4	TWILIGHT SAGA: NEW MOON
5	TWILIGHT
6	SAW IV
7	THE STUPIDS
8	SAW III
9	THE ROOM
10	THE BLAIR WITCH PROJECT

Table 6. Embedding Norms for Top 10 Most Polarizing Movies

RANK	EMBEDDING NORM
1	**5.7035**
2	**5.6654**
3	**5.6213**
4	**5.5320**
5	**5.2780**
6	**4.5922**
7	**4.5799**
8	**4.5675**
9	**4.4974**
10	**4.4616**

4.6. Implicit feedback

Explicit feedback such as user ratings is not always available. To address this, we also implement an implicit feedback system. In this setting, user preferences are inferred from observed behavior rather than explicit scores. The dataset is transformed from the 32M MovieLens dataset by treating all ratings of 4 and above as positive signals, while all other ratings (and non-ratings) are treated as negative signals.

The main task in implicit system is ranking items, instead of predicting the exact rating. Specifically, the system should rank items that the user has interacted with higher than those they did not.

This is done through Bayesian Personalized Ranking,where the model samples a pair of positive item i and negative item j for each user and calculates the predicted score as

$$x_{ui} = u^\top v_i, x_{uj} = u^\top vj$$

where the model should satisfy $x_{ui} > x_{uj}$.

The training objective is to build the user and item embeddings with updates through stochastic gradient descent:

$$\Theta \leftarrow \Theta + \alpha \left(\frac{e^{-\hat{x}_{uij}}}{1 + e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{x}_{uij} + \lambda_\Theta \Theta \right)$$

where $\Theta = \{U, V\}$ are the user and item embeddings.

We evaluate the performance of the implicit models through precision, recall and normalized discounted cumulative gain. As the restuls shows, as K increases, there is a decrese in precision but increase in recall and NDCG.

Table 7. Evaluation Metrics at different K

K	PRECISION	RECALL	NDCG
10	0.1254	0.1231	0.1653
20	0.0977	0.1828	0.1728
50	0.0684	0.2975	0.2060

4.7. A/B Testing

To evaluate whether a change in the recommendation model produces a material difference in recommendation quality, we conducted an A/B testing experiment using simulated dummy users. We created 100 dummy users, each assigned a preference for a specific movie genre. For each user, we simulated ratings for 50 movies belonging to their preferred genre, assuming that these movies received high ratings (ratings of 4 and above). The users are randomly split into group A and B and trained using two separate models.

The predicted score for a movie is computed as:

$$r_{mn} = v^\top u + (weight) b_n^i$$

where weight = 0.03 if model = A, weight = 0.2 if model = B. This set up tests whether changing the weights of the item bias leads to recommendations being better aligned with the user's preference. To measure the impact of model change, we evaluate the genre recall, specifically, how many of the top 5 recommended movies are in the user's preferred genre.

Table 7 illustrates the results of the A/B test.

Table 8. A/B testing of item bias

MODEL	BIAS WEIGHT	RECALL
A	0.05	0.2731
B	0.2	0.2958

The result showed that increasing the weight of the item bias from 0.05 to 0.2 lead to a 8% increase in genre recall.

References

Bottou, L., Peters, J., Quiñonero-Candela, J., Charles, D. X., Chickering, D. M., Portugaly, E., Ray, D., Simard, P., and Snelson, E. Counterfactual reasoning and learning systems: The example of computational advertising. *Journal of Machine Learning Research*, 14(11):3207–3260, 2013.

Dacrema, M. F., Cremonesi, P., and Jannach, D. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pp. 101–109, 2019.

Dror, G., Koenigstein, N., and Koren, Y. Yahoo! music recommendations: modeling music ratings with temporal dynamics and item taxonomy. In *Proceedings of the fifth ACM conference on Recommender systems*, pp. 165–172, 2011.

Hu, Y., Koren, Y., and Volinsky, C. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pp. 263–272. IEEE, 2008.

Koenigstein, N., Dror, G., and Koren, Y. The xbox recommender system. In *Proceedings of the sixth ACM conference on Recommender systems*, pp. 281–284, 2012.

Kohavi, R. and Longbotham, R. Seven pitfalls to avoid when running controlled experiments on the web. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1105–1114, 2009.

Kohavi, R., Longbotham, R., Sommerfield, D., and Henne, R. M. Practical guide to controlled experiments on the web: listen to your customers not to the hippo. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 959–967, 2007.

Kohavi, R., Longbotham, R., Sommerfield, D., and Henne, R. M. Controlled experiments on the web: survey and practical guide. *Data Mining and Knowledge Discovery*, 18:140–181, 2009.

Kohavi, R., Crook, T., and Longbotham, R. Unexpected results in online controlled experiments. *ACM SIGKDD Explorations Newsletter*, 12(2):31–40, 2010.

Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer*, 42(8): 30–37, 2009.

Marlin, B. Collaborative filtering and the missing at random assumption. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pp. 267–275, 2004.

Paquet, U. and Koenigstein, N. One-class collaborative filtering with random graphs. In *Proceedings of the 22nd international conference on World Wide Web*, pp. 999–1008, 2013.

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pp. 452–461, 2009.

Shi, Y., Karatzoglou, A., Baltrunas, L., Larson, M., Oliver, N., and Hanjalic, A. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. *Proceedings of the sixth ACM conference on Recommender systems*, pp. 139–146, 2012.

Stern, D., Herbrich, R., and Graepel, T. Matchbox: large scale bayesian recommendations. In *Proceedings of the 18th International Conference on World Wide Web*, pp. 111–120, 2009.

Takács, G. and Tikk, D. Alternating least squares for personalized ranking. In *Proceedings of the sixth ACM conference on Recommender systems*, pp. 83–90, 2012.

A. Technical Appendix

A.1. ALS with bias only

Loss function:

$$\mathcal{L} = -\frac{\lambda}{2}\sum_{m,n}\left(r_{mn} - b_m^{(u)} - b_n^{(i)}\right)^2 - \frac{\gamma}{2}\left(b_m^{(u)}\right)^2 - \frac{\gamma}{2}\left(b_n^{(i)}\right)^2$$

The updates for user bias:

$$b_m^{(u)} = \frac{\lambda \sum_{n\in\Omega(m)}\left(r_{mn} - b_n^{(i)}\right)}{\lambda|\Omega(m)| + \gamma}$$

The updates for item bias:

$$b_n^{(i)} = \frac{\lambda \sum_{m\in\Omega(n)}\left(r_{mn} - b_m^{(u)}\right)}{\lambda|\Omega(n)| + \gamma}$$

A.2. ALS with item and user embeddings

Loss function:

$$\mathcal{L} = -\frac{\lambda}{2}\sum_{m,n}\left(r_{mn} - u_m^\top v_n - b_m^{(u)} - b_n^{(i)}\right)^2 - \frac{\tau}{2}\sum_m u_m^\top u_m - \frac{\tau}{2}\sum_n v_n^\top v_n - \frac{\gamma}{2}\left(b_m^{(u)}\right)^2 - \frac{\gamma}{2}\left(b_n^{(i)}\right)^2$$

The updates for user bias:

$$b_m^{(u)} = \frac{\lambda \sum_{n\in\Omega(m)}\left(r_{mn} - u_m^\top v_n - b_n^{(i)}\right)}{\lambda|\Omega(m)| + \gamma}$$

The updates for item bias:

$$b_n^{(i)} = \frac{\lambda \sum_{m\in\Omega(n)}\left(r_{mn} - u_m^\top v_n - b_m^{(u)}\right)}{\lambda|\Omega(n)| + \gamma}$$

The updates for user embedding U:

$$u_m = \frac{\lambda \sum_{n\in\Omega(m)} v_n(r_{mn} - b_m^{(u)} - b_n^{(i)})}{\lambda \sum_{n\in\Omega(m)} v_n v_n^\top + \tau I}$$

The updates for item embedding V:

$$v_n = \frac{\lambda \sum_{m\in\Omega(n)} u_m(r_{mn} - b_m^{(u)} - b_n^{(i)})}{\lambda \sum_{m\in\Omega(n)} u_m u_m^\top + \tau I}$$

A.3. ALS with bias, embeddings and features

Loss function:

$$\mathcal{L} = -\frac{\lambda}{2}\sum_{m=1}^{M}\sum_{n\in\Omega(m)}\left(r_{mn} - \left(u_m^\top v_n + b_m^{(u)} + b_n^{(i)}\right)\right)^2 - \frac{\tau}{2}\sum_{n=1}^{N}\left\|v_n - \frac{1}{\sqrt{F_n}}\sum_{\ell\in\text{features}(n)} f_\ell\right\|^2 - \frac{\tau}{2}\sum_{m=1}^{M} u_m^\top u_m - \frac{\tau}{2}\sum_{\ell=1}^{L} f_\ell^\top f_\ell - \frac{\gamma}{2}\sum_{m=1}^{M}\left(b_m^{(u)}\right)^2 - \frac{\gamma}{2}\sum_{n=1}^{N}\left(b_n^{(i)}\right)^2$$

Update for item vector:

$$\mathcal{L} = -\frac{\lambda}{2}\sum_{m\in\Omega(n)}\left(r_{mn} - \left(u_m^\top v_n + b_m^{(u)} + b_n^{(i)}\right)\right)^2 - \frac{\tau}{2}\sum_{n=1}^{N}\left\|v_n - \frac{1}{\sqrt{F_n}}\sum_{\ell\in\text{features}(n)} f_\ell\right\|^2 + \ldots$$

$$\frac{\partial\mathcal{L}}{\partial v_n} = \lambda\sum_{m\in\Omega(n)}\left(r_{mn} - \left(u_m^\top v_n + b_m^{(u)} + b_n^{(i)}\right)\right)u_m - \tau\left(v_n - \frac{1}{\sqrt{F_n}}\sum_{\ell\in\text{features}(n)} f_\ell\right)$$

$$0 = \lambda\sum_{m\in\Omega(n)}\left(r_{mn} - b_m^{(u)} - b_n^{(i)}\right)u_m - \lambda\sum_{m\in\Omega(n)}\left(u_m u_m^\top\right)v_n - \tau v_n + \frac{\tau}{\sqrt{F_n}}\sum_{\ell\in\text{features}(n)} f_\ell$$

$$v_n = \frac{\left(\lambda\sum_{m\in\Omega(n)}\left(r_{mn} - b_m^{(u)} - b_n^{(i)}\right)u_m + \frac{\tau}{\sqrt{F_n}}\sum_{\ell\in\text{features}(n)} f_\ell\right)}{\left(\lambda\sum_{m\in\Omega(n)} u_m u_m^\top + \tau I\right)}$$

Update for feature vector:

$$\mathcal{L} = -\frac{\tau}{2}\sum_{n=1}^{N}\left\|v_n - \frac{1}{\sqrt{F_n}}\sum_{\ell\in\text{features}(n)} f_\ell\right\|^2 - \frac{\tau}{2}\sum_{\ell=1}^{L} f_\ell^\top f_\ell$$

$$\frac{\partial\mathcal{L}}{\partial f_\ell} = \tau\sum_{n:\ell\in\text{features}(n)}\frac{1}{\sqrt{F_n}}\left(v_n - \frac{1}{\sqrt{F_n}}\sum_{j\in\text{features}(n)} f_j\right) - \tau f_\ell$$

$$0 = \sum_{n:\ell\in\text{features}(n)}\frac{1}{\sqrt{F_n}}\left(v_n - \frac{1}{\sqrt{F_n}}\sum_{j\in\text{features}(n)} f_j\right) - f_\ell$$

$$f_\ell = \sum_{n:\ell\in\text{features}(n)}\frac{1}{\sqrt{F_n}}v_n - \sum_{n:\ell\in\text{features}(n)}\frac{1}{F_n}f_\ell\sum_{n:\ell\in\text{features}(n)}\frac{1}{F_n}\sum_{j\in\text{features}(n)\setminus\ell} f_j$$

$$f_\ell = \frac{\sum_{n:\ell\in\text{features}(n)}\frac{1}{\sqrt{F_n}}\left(v_n - \frac{1}{\sqrt{F_n}}\sum_{j\in\text{features}(n)\setminus\ell} f_j\right)}{1 + \sum_{n:\ell\in\text{features}(n)}\frac{1}{F_n}}$$