UNIVERSITY OF VERONA

Mining Massive Datasets

FINAL PROJECT - A.Y. 2024/25

Andrea Arragoni - VR510366
Pedro Alonso Lopez Torres - VR501305

Contents

1 Introduction

In this project, we explored modern recommender systems based on latent-factor models, with the goal of comparing their performance against more classical collaborative filtering approaches. The main objective was to assess whether the additional complexity introduced by latent representations translates into measurable performance improvements in top-N recommendation tasks.

The work was primarily inspired by the paper *Local Latent Space Models for Top-N Recommendation* by Evangelia Christakopoulou and George Karypis [1], which introduces the rGLSVD and sGLSVD algorithms for combining global and local latent spaces. Furthermore, the theoretical background for the collaborative filtering component was drawn from the topics covered in the *Mining Massive Datasets* course held at the University of Verona during the 2024/2025 academic year by Professor Damiano Carra.

The following sections first review the fundamental top-N recommendation approaches and then describe the implementation, parameter selection, and experimental evaluation of the proposed latent-space models.

2 Top-N Recommendation Approaches

Many modern commercial recommender systems adopt the top-N approach, which focuses on identifying the n most relevant items for each user u. These systems compute predicted rating scores internally to rank items, but only the relative ordering of the scores is used to generate the final recommendation list. Typically, the scores themselves are not shown [2].

In this report, we start by describing a widely used baseline method for generating top-N recommendations: *PureSVD* [2]. We then extend this discussion to more advanced latent-space models that build upon the latter, namely *rGLSVD* and *sGLSVD* [1].

These approaches are all grounded in the factorization of the utility matrix, which enables us to extract latent factors that capture the underlying structure of user-item interactions. These techniques are commonly referred to as SVD models after the related *Singular Value Decomposition*.

The key idea is to divide the original utility matrix into two lower-rank matrices: a *user matrix* and an *item matrix*. Formally [2]:

$$\hat{r}_{ui} = b_{ui} + \mathbf{p}_u \mathbf{q}_i^\top \qquad (1)$$

where $\hat{r}_{ui}$ is the predicted rating that user u would assign to item i, b_{ui} is a bias term accounting for global, user, and item-specific deviations, $\mathbf{p}_u \in \mathbb{R}^f$ is the f-dimensional latent vector representing user u, and $\mathbf{q}_i \in \mathbb{R}^f$ is the corresponding latent vector representing item i. The inner product $\mathbf{p}_u \mathbf{q}_i^\top$ captures the interaction between the user's and item's latent features, providing the personalized component of the predicted score.

The main problem with this approach is that the user-item utility matrix is *sparse*, meaning most of its values are missing. This makes it impossible to apply a classic SVD, which requires a *dense* matrix. The following subsection describes how PureSVD overcomes this limitation.

2.1 PureSVD

As previously mentioned, our objective in top-N recommendation tasks is to generate a ranking of n potentially interesting items $i \in \{i_1, i_2, \ldots, i_n\}$ for user u, disregarding their specific prediction scores (p_{ui}). This allows us to make certain assumptions. For example, missing values in the utility matrix could be treated as 0, or any other value outside the rating scale.

So effectively, the utility matrix $\mathbf{R}$ is completed as:

$$r_{ui} = \begin{cases} \text{observed rating,} & \text{if known} \\ 0, & \text{otherwise.} \end{cases}$$

All matrix entries are now non-missing. Thus, the user–item rating matrix $\mathbf{R}$ is estimated by the factorization:

$$\hat{\mathbf{R}} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{Q}^\top \qquad (2)$$

where $\mathbf{U}$ is an $n \times f$ orthonormal matrix representing the latent features of the users, $\mathbf{Q}$ is an $m \times f$ orthonormal matrix representing the latent features of the items, and $\mathbf{\Sigma}$ is an $f \times f$ diagonal matrix containing the first f singular values [2]. In other words, a *truncated SVD* is applied.

2.2 GLSVD: Global and Local Singular Value Decomposition

Although PureSVD models user preferences using a single, global, low-rank factorization, this approach has its limitations. Users may base their decisions on both *global* aspects, that apply to all users, and *local* aspects, shared only within certain groups. As stated by Christakopoulou and Karypis [1]: *"...a young girl can decide on a piece of clothing to purchase, based on some general aspects, such as whether it is in good condition, and also on some more specific aspects, such as whether this item of clothing is fashionable for girls her age. Thus, such a preference model contains both global as well as local elements."*

The *Global and Local Singular Value Decomposition* (GLSVD) framework extends PureSVD by explicitly modeling both kinds of latent factors. In this approach, the user-item matrix is explained by two components: a global low-rank model that captures patterns shared across all users, and a set of local low-rank models that capture the preferences of specific user subsets. Two main variants are proposed: rGLSVD and sGLSVD.

2.2.1 rGLSVD

The *rank-varying GLSVD* (rGLSVD) method assumes that the set of user clusters is fixed throughout the process. First, users are assigned to k clusters based on their rating patterns. Then, a global model and multiple local models are estimated by performing a truncated SVD on both the global utility matrix, $\mathbf{R}_g$, and each cluster-specific local matrix, $\mathbf{R}_c$. The global SVD has a fixed number of latent dimensions f_g, while each local model is allowed to have its own rank f_c, enabling different subsets of users to be modeled with varying levels of complexity [1]. The choice of both this type of parameters is discussed in Subsection 3.1.

Formally, $\mathbf{R}_g$ and $\mathbf{R}_c$ are defined as:

$$\mathbf{R}_g = [g_u\, \mathbf{r}_u^\top]_{u=1}^n, \qquad \mathbf{R}_c = [(1 - g_u)\, \mathbf{r}_u^\top]_{u\in c}, \tag{3}$$

where g_u is a user-specific weight representing the contribution of the global model for user u, and the weight of the local component is consequently given by $(1-g_u)$ [1]. Further details on the computation of g_u, common to both rGLSVD and sGLSVD, are provided in Subsection 2.3.

Truncated SVD is then applied to both $\mathbf{R}_g$ and $\mathbf{R}_c$:

$$\mathbf{R}_g \approx \mathbf{P}\boldsymbol{\Sigma}_{f_g}\mathbf{Q}^\top, \qquad \mathbf{R}_c \approx \mathbf{P}_c\boldsymbol{\Sigma}_{f_c}\mathbf{Q}_c^\top, \tag{4}$$

where $\mathbf{P}$ and $\mathbf{Q}$ are the orthonormal matrices containing the latent factors for users and items, respectively, while $\boldsymbol{\Sigma}_f$ is a diagonal matrix with the f largest singular values.

User weights g_u are updated iteratively until convergence, based on the reconstruction error and balancing the influence of global and local components. In practice, the algorithm alternates between factorization and weight updates until fewer than 1% of users change their weights by more than 1% between iterations [1].

2.2.2 sGLSVD

The *subset-varying GLSVD* (sGLSVD) method inverts the flexibility of rGLSVD. Here, all local models share the same number of latent dimensions f_c, but the user clusters are allowed to change during training [1].

Like rGLSVD, a global model and multiple local models are estimated through truncated SVD:

$$\mathbf{R}_g \approx \mathbf{P}\boldsymbol{\Sigma}_{f_g}\mathbf{Q}^\top, \qquad \mathbf{R}_c \approx \mathbf{P}_c\boldsymbol{\Sigma}_{f_c}\mathbf{Q}_c^\top,$$

but after each iteration, users are projected into every cluster and reassigned according to the lowest training error. The process continues until fewer than 1% of users switch clusters [1].

The key operation that enables cluster reassignment in sGLSVD is the projection of a user into the latent space of a different cluster. For a user u being evaluated for membership in cluster c, the projected user latent factor is computed as follows:

$$\mathbf{p}_u^{c\top} = \mathbf{r}_u^\top \mathbf{Q}_c \boldsymbol{\Sigma}_{f_c}^{-1}. \tag{5}$$

This projection allows the model to express user u in terms of the latent factors of cluster c, even if the user did not originally belong to it. By comparing the resulting reconstruction errors across clusters, the algorithm can identify the subset that provides the best low-rank representation for each user. This mechanism is what enables sGLSVD to iteratively refine user assignments while keeping the local model ranks fixed [1].

After describing how these models extend PureSVD to capture user heterogeneity, it is important to clarify that rGLSVD and sGLSVD are intentionally kept separate. Combining them within a single optimization framework would lead to overfitting, as users would tend to be reassigned to the subset with the largest number of local dimensions, thereby minimizing training error but compromising generalization performance [1].

We now turn our attention to the computation of the user-specific weights g_u and the prediction scores p_{ui}, which are common to both GLSVD variants.

2.3 GLSVD: User Weights and Prediction Scores

In both rGLSVD and sGLSVD, the user-specific weight g_u determines the relative contribution of the global and local components in reconstructing the user-item interactions. This value is obtained by minimizing the squared reconstruction error over all items, leading to the following closed-form solution [1]:

$$g_u = \frac{\sum_{i=1}^{m}(a-b)(r_{ui}-b)}{\sum_{i=1}^{m}(a-b)^2}, \tag{6}$$

where $a = \frac{1}{g_u}\mathbf{p}_u^\top \boldsymbol{\Sigma}_{f_g}\mathbf{q}_i$ represents the prediction from the global component, and $b = \frac{1}{1-g_u}\mathbf{p}_u^{c\top}\boldsymbol{\Sigma}_{f_c}\mathbf{q}_i^c$ represents the corresponding contribution from the local component. Here, $\mathbf{p}_u^\top$ denotes the uth row of $\mathbf{P}$ corresponding to user u, and $\mathbf{q}_i$ is the ith column of $\mathbf{Q}^\top$ corresponding to item i. Similarly, $\mathbf{p}_u^{c\top}$ and $\mathbf{q}_i^c$ represent the user and item latent factors within cluster c, respectively, where c indexes the local subset to which user u belongs. This balance between global and local models assigns higher g_u values to users whose preferences are better captured by the global structure and lower values to those better modeled within their local cluster. The computation of g_u is therefore essential for the convergence of rGLSVD and for the optimization of sGLSVD.

Once convergence has been reached, the predicted score $\tilde{r}_{ui}$ for user u and item i is computed as the sum of the global and local reconstructions:

$$\tilde{r}_{ui} = \mathbf{p}_u^\top \boldsymbol{\Sigma}_{f_g}\mathbf{q}_i + \mathbf{p}_u^{c\top}\boldsymbol{\Sigma}_{f_c}\mathbf{q}_i^c. \tag{7}$$

It is important to note that g_u does not appear explicitly in the prediction formula. Its effect is implicitly embedded in the latent user factors $\mathbf{p}_u$ and $\mathbf{p}_u^c$, which are derived from the global and local matrices $\mathbf{R}_g$ and $\mathbf{R}_c$ that already incorporate the terms g_u and $(1-g_u)$, respectively. This guarantees that both global and local patterns contribute to the final recommendation scores, while maintaining a unified prediction framework across rGLSVD and sGLSVD.

2.4 GLSVD: Evaluation Methodology and Performance Metrics

To evaluate the performance of both rGLSVD and sGLSVD, we employed the *Leave-One-Out Cross-Validation* (LOOCV) strategy, as described in the referenced paper [1]. For each user, one of the rated items is randomly selected and placed in the test set, while its corresponding entry in the training matrix is set to zero. This ensures that every user contributes exactly one held-out item for evaluation, while the remaining observed interactions are used for model training. The process is repeated for all users, resulting in a training set and a corresponding test set.

After training the model, recommendation quality is assessed through two standard top-N metrics: the *Hit Rate* (HR) and the *Average Reciprocal Hit Rank* (ARHR). Both metrics evaluate whether the held-out item for each user is among the top-N recommended items. In the case of ARHR, the metric also considers the item's position in the list. Formally [1]:

$$HR = \frac{\#\text{hits}}{\#\text{users}}, \tag{8}$$

$$ARHR = \frac{1}{\#\text{users}} \sum_{i=1}^{\#\text{hits}} \frac{1}{p_i}, \tag{9}$$

where #users denotes the total number of users, #hits the number of users whose held-out item appears in their top-N recommendation list, and p_i the position of that item in the ranked list ($p_i = 1$ corresponds to the top position). Both HR and ARHR range between 0 and 1, with higher values indicating better recommendation performance. More specifically, ARHR can be viewed as a weighted version of HR, where higher-ranked hits contribute more strongly to the final score. This is achieved by taking the reciprocal of each item's rank position ($1/p_i$). For example, a hit at rank 1 contributes more (e.g., $1/1$) than a hit at rank 10 (e.g., $1/10$) when $N = 10$.

3 Recommendation System Implementation

After describing the theoretical foundations of rGLSVD and sGLSVD, we focus on their practical implementation in Python. In this section, we describe the structure of the main scripts, helper functions, and accompanying notebooks used for experimentation and analysis.

All the code here discussed is publicly available in our project repository:

`github.com/mmd_project`

All experiments and evaluations were conducted using the *MovieLens Latest Small* dataset, which contains approximately 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users. The dataset is publicly available at:

`grouplens.org/datasets/movielens/latest`

3.1 Parameters Selection

In our implementation, we did not use the same parameter values as in the original paper, namely the number of clusters, the global latent dimension f_g, and the local dimensions f_c. This choice was mainly due to the fact that we worked with a much smaller dataset, for which the original parameters were not suitable. In addition, the paper does not explain how those values were selected, particularly for the latent factors, but only refers to a technical report that lists the parameters without describing the selection process [3]. For these reasons, we defined our own method to find parameter values that performed well in our setting.

3.1.1 Number of clusters

The original paper cites *CLUTO* [4] as the clustering method used. CLUTO is a software package written in C that offers efficient clustering algorithms for high-dimensional data. However, since our implementation was entirely developed in Python, we could not directly integrate CLUTO and therefore adopted an alternative clustering approach compatible with our environment.

We used the *k-modes* algorithm from the homonym Python library `kmodes`, which is better suited for binary implicit-feedback data than the traditional *k-means*. To identify the optimal number of clusters, we fitted the algorithm on a random subset of 1,500 item columns from the binarized utility matrix $\mathbf{R}_{bin}$, testing values of k in the range $[1, 19]$. The initialization method was set to *Huang*, with a single initialization (`n_init=1`) to reduce computational cost, particularly relevant for larger datasets. While a higher number of initializations typically produces smoother and more stable clustering results, and may lead to a different optimal number of clusters, using one initialization was sufficient for our exploratory setup, which yielded an elbow around $k = 7$ (See Figure 1).

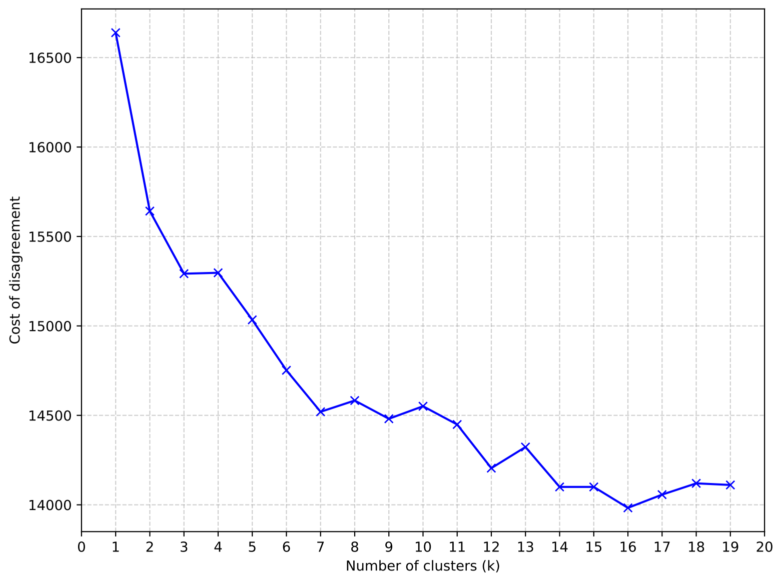

Figure 1: Elbow method applied to the *k-modes* clustering cost function, computed on a random subset of 1,500 item columns from the binarized utility matrix $\mathbf{R}_{bin}$. Each point represents the disagreement cost obtained for values of k ranging from 1 to 19 using a single initialization. The elbow, observed around $k = 7$, indicates the point of diminishing returns in cluster compactness.

3.1.2 Number of global and local latent factors

To determine the dimensionality of both the global and local latent spaces, we analyzed the decay of the singular values obtained through truncated SVD. For the global component, we performed SVD on the full binarized utility matrix $\mathbf{R}_{bin}$ and plotted the first 100 singular values in descending order. Similarly, for each local cluster, we applied SVD on the corresponding submatrix $\mathbf{R}_c$ obtained from users belonging to that cluster, and examined the decay of its singular values. In both cases, the goal was to identify the "elbow point" where additional factors contribute minimally to the explained variance, analogous to the logic used for selecting the optimal number of clusters. Based on this inspection, we set the number of global latent factors to $f_g = 18$, and the number of local latent factors to $f_c = \{7, 5, 10, 8, 6, 7, 6\}$, balancing model expressiveness and computational efficiency while avoiding overfitting by including only those latent factors that capture significant variance in the data and excluding components dominated by noise. The corresponding implementation details are provided in the `parameters_research.ipynb` notebook.

3.2 rGLSVD Implementation

The implemented `RGLSVDRecommender` class follows the iterative structure of the rGLSVD algorithm described in Subsection 2.2, ensuring numerical stability, modularity, and reproducibility. The binary user-item utility matrix $\mathbf{R}_{bin}$ is first constructed from the rating data using dedicated preprocessing functions (See `utils.py`). All positive ratings are mapped to 1 and missing ratings to 0, so that $\mathbf{R}_{bin}$ captures implicit rather than explicit feedback. Users are assigned to k predefined clusters before training (in our experiments, $k = 7$), consistent with the rGLSVD assumption that user subsets remain fixed during optimization. Each cluster c has its own local rank f_c, while the global model uses a single rank f_g, allowing heterogeneous user groups to be modeled with different representational complexity. This configuration captures diverse preference structures without re-clustering at each iteration. In our setup, $f_g = 18$ and $f_c = \{7, 5, 10, 8, 6, 7, 6\}$ for each cluster, respectively.

The optimization alternates between three main steps until convergence or a maximum number of iterations:

1. **Global and local matrices.** Construct the global weighted matrix $\mathbf{R}_g$ by multiplying each user's interaction vector by its current weight g_u, and, for each cluster c, construct the local weighted matrix $\mathbf{R}_c$ by scaling the corresponding users' rows by $(1 - g_u)$.

2. **Latent factors via truncated SVD.** Compute a rank-f_g truncated SVD of $\mathbf{R}_g$ and, for each cluster, a rank-f_c truncated SVD of $\mathbf{R}_c$, using `scipy.sparse.linalg.svds` on CSR-formatted (*Compressed Sparse Row*) matrices. The CSR format stores only the nonzero entries of a sparse matrix along with their indices, thus allowing the truncated SVD to efficiently handle the high-dimensional and sparse structure of $\mathbf{R}_g$ and $\mathbf{R}_c$ without processing the zero elements explicitly.

3. **User-weight update.** Update the weights g_u according to the closed-form expression derived from Equation 6, clipping values to $[0, 1]$ for stability. A small constant $\varepsilon = 10^{-12}$ is added to denominators to prevent numerical instability when they approach zero.

The algorithm starts with $g_u = 0.5$ for every user, giving equal importance to global and local components. It iterates until fewer than 1% of users change their weights by more than 0.01 (defaults: `convergence_threshold=` 0.01, `weight_change_threshold=` 0.01), or when `max_iterations` is reached. After convergence, the trained state (global and local factors, user weights g_u, and final assignments) is stored.

Predicted scores are then computed by combining the global and local latent components according to Equation 7. Once predictions are obtained for all user-item pairs, recommendation quality is assessed through LOOCV, following the steps illustrated in Subsection 2.4. Finally, the `evaluate_metrics()` method computes the HR and ARHR metrics, using the same definitions as in Equations 8 and 9, respectively. The complete implementation of the described class is provided in the `rglsvd_class.py` file.

3.3 sGLSVD Implementation

The `SGLSVDRecommender` class implements the sGLSVD variant in which *user subsets can change during training* while the *local rank is fixed across clusters*, as described in Subsection 2.2. Like the rGLSVD implementation, the binary user-item matrix $\mathbf{R}_{bin}$ is built from ratings using dedicated preprocessing utilities (See `utils.py`), mapping positives to 1 and missing entries to 0. The model receives an initial clustering of users into k subsets and uses a single global rank f_g together with a *shared* local rank f_c for all clusters. We used the same f_g value as in rGLSVD, while the local rank f_c in sGLSVD was set to the average of the f_c values used in rGLSVD.

The optimization process, similar to rGLSVD, involves three steps that alternate until convergence or the maximum number of iterations is reached:

1. **Global and local matrices.** Construct the global weighted matrix $\mathbf{R}_g$ by scaling each user row by its current weight g_u, and for each cluster c construct the local weighted matrix $\mathbf{R}_c$ by scaling its users' rows by $(1 - g_u)$.

2. **Latent factors via truncated SVD.** Compute a rank-f_g truncated SVD of $\mathbf{R}_g$ and, for each cluster, a rank-f_c truncated SVD of $\mathbf{R}_c$, using `scipy.sparse.linalg.svds` on CSR inputs for efficiency on high-dimensional, sparse data. Small-shape safeguards ensure $k < \min(n, m)$, and empty clusters skip the local SVD. When `svds` is not applicable (e.g.,

degenerate shapes), a dense SVD fallback is used.

3. **Cluster reassignment and user-weight update.** For each user u, evaluate its current cluster and all other candidate clusters. For a candidate cluster c, project u into the cluster's local latent space using Equation 5, then compute the optimal g_u from Equation 6 (clipped to $[0, 1]$) and the corresponding squared reconstruction error. Reassign u to the cluster that yields the lowest error if it improves upon the current assignment by a factor specified by the parameter `min_error_improvement`. Numerical safeguards (e.g., $\varepsilon = 10^{-12}$) prevent division by zero in Equations 5 and 6.

Initialization sets $g_u = 0.5$ for all users. At each iteration, users may update their weights g_u and are reassigned to a different cluster only if the resulting reconstruction error improves by at least the factor specified by `min_error_improvement` (set to 0.01). The loop terminates when the fraction of users that switch clusters in a given iteration falls below the threshold (default 0.005), or when `max_iterations` is reached. After convergence, local factors are recomputed once for the final clusters, and the trained state (global and local factors, user weights g_u, and final assignments) is stored.

The final steps mirror those used in the rGLSVD evaluation pipeline. Predictions are generated by combining the global and local latent components as defined in Equation 7, and evaluation follows the LOOCV protocol with the top-N metrics introduced in Equations 8 and 9.

4 Experimental Results and Discussion

After implementing rGLSVD and sGLSVD, we compared the quality of their recommendations against that of a classical *user-user collaborative filtering* baseline. All models were evaluated on the same binarized version of the MovieLens Latest Small dataset, following the LOOCV protocol and top-N metrics (HR and ARHR) described in Subsection 2.4.

4.1 Collaborative Filtering

The user-user collaborative filtering (CF-UU) approach computes pairwise similarities between users over the binary utility matrix $\mathbf{R}_{bin}$, where $r_{u,i} = 1$ if user u has interacted with item i and 0 otherwise. Let $\mathbf{r}_u$ denote the rating vector of user u, and let N be the set of the k users most similar to u who have rated item i. The predicted rating of user u for item i is then given by the similarity-weighted average of these neighbors [5]:

$$\hat{r}_{u,i} = \frac{\sum\limits_{v \in N} s_{uv}\, r_{v,i}}{\sum\limits_{v \in N} s_{uv}}, \tag{10}$$

where s_{uv} denotes the similarity between users u and v.

In that regard, two similarity metrics were evaluated:

- **Jaccard similarity:**
$$s_{\text{Jaccard}}(u, v) = \frac{|I_u \cap I_v|}{|I_u \cup I_v|}, \tag{11}$$
where I_u and I_v are the sets of items rated by users u and v, respectively.

- **Cosine similarity:**
$$s_{\text{Cosine}}(u, v) = \frac{\mathbf{r}_u \cdot \mathbf{r}_v}{\|\mathbf{r}_u\|\,\|\mathbf{r}_v\|}, \tag{12}$$
where $\mathbf{r}_u$ and $\mathbf{r}_v$ denote the user rating vectors extracted from $\mathbf{R}_{\text{bin}}$.

Results confirmed that the choice of similarity metric has a substantial impact on performance. The Jaccard-based model performed poorly (HR@10 = 0.0869), largely due to the extreme sparsity of the matrix (1.7%), which causes most user intersections to be negligible relative to their unions. By contrast, Cosine similarity achieved HR@10 = 0.2803 and ARHR = 0.1486, approximately 3.2 times higher than Jaccard, since it is less sensitive to sparsity and better captures angular correlations between user profiles. In general, cosine-based measures tend to produce more stable similarity estimates, as they account for the relative orientation of user rating vectors rather than the absolute number of co-rated items [5]. A further refinement commonly applied in rating-based collaborative filtering is the mean-centered variant of cosine similarity, equivalent to the Pearson correlation coefficient, which normalizes each user's ratings by subtracting their average before computing similarities. However, this normalization step is not applicable in our case, since $\mathbf{R}_{\text{bin}}$ is a binary implicit-feedback matrix and does not contain continuous rating values from which user averages could be subtracted. All the code implementation can be found in the `collaborative_class.py` file.

4.2 Performance of Latent-Space Models

Following the analysis of the collaborative filtering baseline, we now examine the results obtained using latent-factor models. Unlike similarity-based approaches, which rely on direct pairwise comparisons between users, latent-space methods infer hidden dimensions that capture underlying patterns of user–item interactions. In our experiments, both SVD-based variants achieved substantially higher scores:

- **rGLSVD:** HR@10 = 0.3951, ARHR = 0.2841;
- **sGLSVD:** HR@10 = 0.3164, ARHR = 0.2036.

These values correspond to improvements of roughly 9–41% over the best CF-UU baseline, depending on the metric. The superior performance of matrix factorization demonstrates its ability to uncover latent structures that extend beyond surface-level co-occurrence statistics. By jointly modeling both global and local latent spaces, the GLSVD framework captures heterogeneous user preferences more effectively, outperforming methods that rely solely on explicit similarity computations. These results can be found in the `main.py` file, where the rGLSVD, sGLSVD, and CF-UU models are run in sequence.

4.3 Discussion

The comparative results highlight several key insights:

- **Robustness to sparsity:** latent-factor models generalize better in sparse environments by leveraging global structures and shared latent dimensions.
- **Dual representation:** combining global and local components enhances personalization for subgroups of users with distinct preferences.
- **Model design trade-offs:** rGLSVD, which maintains fixed clusters but allows varying local ranks, slightly outperformed sGLSVD, where clusters can be reassigned but ranks remain fixed. This indicates that flexible model complexity may yield higher benefits than dynamic cluster assignment for this dataset.

Overall, the experimental evidence confirms that incorporating latent representations yields a clear performance advantage over classical collaborative filtering, particularly in sparse, real-world settings. The global-local decomposition provides a principled framework to balance shared structure with user-specific variation, resulting in more accurate top-N recommendations.

5 Conclusion

This work presented an implementation and evaluation of latent-factor recommender systems and compared their performance with that of classical collaborative filtering approaches. The results confirmed that latent-space models are superior at capturing hidden structures in user-item interactions, resulting in greater accuracy in top-N recommendation tasks.

There are two main aspects that could be improved in future implementations. The first concerns the parameter selection process. In latent-factor models, parameters such as the number of clusters and the dimensionality of the global and local latent spaces are critical in determining model quality. Since the reference paper only reports the final values of these parameters in a technical appendix and does not detail the tuning procedure, our approach relied on straightforward empirical exploration. A more systematic parameter optimization strategy would likely produce better, more generalizable results.

The second aspect relates to scalability. Due to computational constraints, the experiments were conducted on the MovieLens Latest Small dataset. Future work could extend the evaluation to larger datasets, allowing for a more comprehensive analysis of model robustness and performance at scale. This would be possible if sufficient computational resources were available.

References

[1] Evangelia Christakopoulou and George Karypis. ‘Local Latent Space Models for Top-N Recommendation’. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’18. London, United Kingdom: Association

for Computing Machinery, 2018, pp. 1235–1243. ISBN: 9781450355520. DOI: 10.1145/3219819.3220112. URL: https://doi.org/10.1145/3219819.3220112.

[2] Paolo Cremonesi, Yehuda Koren and Roberto Turrin. 'Performance of recommender algorithms on top-n recommendation tasks'. In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. Barcelona, Spain: Association for Computing Machinery, 2010, pp. 39–46. ISBN: 9781605589060. DOI: 10.1145/1864708.1864721. URL: https://doi.org/10.1145/1864708.1864721.

[3] Evangelia Christakopoulou and George Karypis. *Local Latent Space Models for Top-N Recommendation Appendix - Technical Report*. Tech. rep. 18-010. University of Minnesota, Department of Computer Science & Engineering, 2018. URL: https://www.cs.umn.edu/sites/cs.umn.edu/files/tech_reports/18-010.pdf.

[4] George Karypis. *CLUTO: A Clustering Toolkit*. Tech. rep. UMN-CS-2002-XXX. Accessed: YYYY-MM-DD. University of Minnesota, 2002. URL: https://conservancy.umn.edu/items/4fbef165-f964-41ed-a239-86a8f931ffbe.

[5] Jure Leskovec, Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. 2nd. USA: Cambridge University Press, 2014. ISBN: 1107077230.