Leiden University

Advances in Data Mining

# Assignment 1: Recommender Systems

Group 44

*Authors:*
Louis Siebenaler (s3211126)
Theofilus Hobba Pramono (s3613291)
Peter Breslin (s3228282)


October 24, 2023

# 1 Introduction

Recommender systems refer to the assortment of algorithms designed to make recommendations to users based on data collected about the user and/or system. These operate to predict the most likely user responses so that the most relevant items are recommended. The ability to make such predictions is extremely important to many companies in the world today that rely on user interaction and engagement (e.g. Netflix, Amazon, YouTube). Furthermore, these type of recommendations are crucial in many real-world applications where data is often incomplete and predictions are required (e.g. to fill in missing entries in sparse datasets).

While an array of recommender systems exist for a wide variety of purposes, two broad groups can be distinguished: content-based systems and collaborative filtering systems. The former can be thought of as a more simplistic approach which uses the properties of the recommended items as the basis to make recommendations (e.g. using a user's most listened to music genre as grounds to make future recommendations). On the other hand, collaborative filtering systems recommends items based on similarities between users and/or items such that the recommendations made to a given user are those favoured by 'similar' users (Leskovec et al. 2014).

This assignment aimed to explore a variety of algorithms and computational techniques to design and implement a range of recommender systems for recommending movies. In particular, a selection of rudimentary approaches were examined and contrasted against one another, and more sophisticated routines were explored. Furthermore, efforts were made to better interpret and visualize the operation of such methods using different dimensionality reduction techniques.

# 2 Method

## 2.1 Data

The MovieLens 1M dataset (Harper & Konstan 2015) was employed in this assignment, a collection of 1,000,209 ratings of approximately 3,900 movies made by 6,040 users. In the context of this work, the ultimate goal of the recommender system is to predict ratings for each user-item pair (where a given item refers to a given movie). These ratings can be represented by a $U \times I$ matrix, where $U$ is the dimension of the user data and $I$ being that of the item data. Hence, the rows are given by users while the columns represent the movies. Since there are many user-item pairs with an unknown rating, this matrix will be extremely sparse and will only contain a very limited number of ratings. This is known as the utility matrix, $R$, and it is the empty entries in this matrix we aim to predict using recommender systems. We choose to fill in the empty (i.e. missing) entries in $R$ with value zero, while known ratings are given a value on a 1–5 scale (5 being highest).

## 2.2 Accuracy

To assess the performance of the recommender systems we use the mean squared error (RMSE) and mean absolute error (MAE) metrics. These are defined as

$$\text{RMSE} = \sqrt{\frac{\Sigma_{(r,s)\in\mathfrak{R}}(\hat{r}_{rs} - r_{rs})^2}{N}}, \tag{1}$$

$$\text{MAE} = \frac{\Sigma_{(r,s)\in\mathfrak{R}}|\hat{r}_{rs} - r_{rs}|}{N}, \tag{2}$$

where $\hat{r}_{rs}$ is a predicted rating, $r_{rs}$ is the true rating of a utility matrix $R$ and $\mathfrak{R}$ is the subset of size $N$ of all the known elements of $R$. To verify the stability of a model, we perform a 5-fold cross validation. Hence, our dataset is randomly split into 5 approximately equally sized subsets, using a random seed 10. An algorithm is trained on 4 out of 5 subsets of the total data and the final part is used as the testing set. Each of the 5 subsets is used once as the test set along with the corresponding unique training set. At the end, we average the RMSE and MAE over the 5 runs. Additionally, we improve the predictions by rounding values bigger than 5 to 5 and smaller than 1 to 1.

## 2.3 Naive Approaches

A naive approach in developing recommender systems is to use averages of known ratings as a basis of estimation. In this work, we develop 5 kinds of naive models based on global average, average rating of each item, average rating of each user, and linear combinations of both item and user averages with and without an intercept value.

In the global average approach, the predicted rating $\hat{r}_{rs}$ of a utility matrix corresponds to the average value of all known ratings of $R$. Mathematically, this implies

$$\hat{r}_{rs} = \frac{1}{N}\sum_{(r,s)\in\mathfrak{R}} r_{rs}, \tag{3}$$

where all the quantities have the same meaning as in equation (1) and (2).

Given a movie's uniqueness, an improved method for predicting its rating is to use the average value of all its known ratings as the prediction. Hence, for the average item approach, the predicted rating of a movie of index $m$ is given by

$$\hat{r}_{rm} = \frac{1}{N_m}\sum_{r\in\mathfrak{R}_m} r_{rm}, \tag{4}$$

where we are summing over all users whose ratings of movie $m$ are available, that is users belonging to the subset $\mathfrak{R}_m$ of size $N_m$. However, it is possible that there exists movies for which there are no available ratings and $\mathfrak{R}_m$ is of size $N_m = 0$. To overcome this, we employ the global average of the utility matrix $R$ (equation (3)) and use this as the fall-back value for the prediction $\hat{r}_{rm}$.

Alternatively, we can use the average value of known ratings of a user $u$ to predict an unknown movie rating for that user. Hence, for the average user approach, the predicted rating of a user $u$ is given by

$$\hat{r}_{us} = \frac{1}{N_u} \sum_{s \in \mathfrak{R}_u} r_{us}, \tag{5}$$

where we are summing over the subset $\mathfrak{R}_u$ of length $N_u$, which corresponds to all the available ratings of user $u$. Similarly to the movie average approach, it is possible that $\mathfrak{R}_u$ is of length $N_u = 0$. We overcome this in the same way by employing the global average value as a fall-back value for the prediction $\hat{r}_{us}$.

It is possible to combine equation (4) and (5) to create a linear regression model for predicting the rating of a movie $m$ by a user $u$. This is expressed by

$$\hat{r}_{um} = \alpha \hat{r}_{us} + \beta \hat{r}_{rm} + \gamma, \tag{6}$$

where $\hat{r}_{rm}$ and $\hat{r}_{us}$ are given by equation (4) and (5) respectively and we use the global average value as the fall-back value if needed. The parameters $\alpha$ and $\beta$ correspond to the regression coefficients whereas $\gamma$ is the intercept parameters. These fitting parameters are obtained through linear regression which is performed using the `sklearn.linear model.LinearRegression` package.

## 2.4 UV Matrix Decomposition

An entirely different approach to estimate the ratings in the utility matrix $R$ is to assume that $R$ can be modelled as a product of two long, thin matrices. This approach is called '*UV matrix decomposition*' and is derived from the more general singular-value decomposition theory. Assuming we have a sparse utility matrix $R$, with $n$ rows and $m$ columns (i.e $n$ users and $m$ movies), then we can decompose it into a product of a matrix $U$, with $n$ rows and $d$ columns, and $V$, with $d$ rows and $m$ columns. Hence

$$\begin{bmatrix} r_{11} & \dots & r_{1m} \\ \vdots & \ddots & \vdots \\ r_{n1} & \dots & r_{nm} \end{bmatrix} = \begin{bmatrix} u_{11} & \dots & u_{1d} \\ \vdots & \ddots & \vdots \\ u_{n1} & \dots & u_{nd} \end{bmatrix} \begin{bmatrix} v_{11} & \dots & v_{1m} \\ \vdots & \ddots & \vdots \\ v_{d1} & \dots & v_{dm} \end{bmatrix}, \tag{7}$$

where $r_{rs}$, $u_{rs}$ and $v_{rs}$ are the elements of the $R$, $U$ and $V$ matrix respectively. The convention of our notation for matrix elements is that the first subscript denotes row number and the second subscript denotes column number. To predict a certain rating $r_{rs}$, we must multiply the $r^{\text{th}}$ row from $U$ and the $s^{\text{th}}$ column of $V$

$$\hat{r}_{rs} = \begin{bmatrix} u_{r1} & \dots & u_{rd} \end{bmatrix} \begin{bmatrix} v_{1s} \\ \vdots \\ v_{ds} \end{bmatrix}, \tag{8}$$

where $\hat{r}_{rs}$ is the predicted rating of movie $s$ by user $r$. From this, the goal is to find the elements of $U$ and $V$ such that the RMSE between the product $UV$ and $R$ is minimized. The technique we

use for finding the optimal $UV$-decomposition is gradient descent. Hence, for a single element of $U$ or $V$, we compute the direction of change that most decreases the RMSE. The general formula for adjusting an element $u_{rs}$ from $U$ to a new value $u_{rs,\text{new}}$ is

$$u_{rs,\text{new}} = \frac{\sum_j^m v_{sj}(r_{rj} - \sum_{k\neq s}^d u_{rk}v_{kj})}{\sum_j^m v_{sj}^2}, \tag{9}$$

where $\sum_j^m$ is the sum over all column $j$ such that $r_{rj}$ is non-blank and $\sum_{k\neq s}^d$ is the sum for all $k$ except $k = s$. There is an analogous formula for adjusting an element $v_{rs}$ from $V$ to a new value $v_{rs,\text{new}}$

$$v_{rs,\text{new}} = \frac{\sum_i^n u_{ir}(r_{is} - \sum_{k\neq r}^d u_{ik}v_{ks})}{\sum_i^n u_{ir}^2}, \tag{10}$$

where $\sum_i^n$ corresponds to the sum over all row $i$ such that $r_{is}$ is non-blank and $\sum_{k\neq s}^d$ is the sum for all $k$ except $k = r$.

Before applying equation (9) and (10) for the optimization, we preprocess the utility matrix $R$ in order to eliminate biases in our learning algorithm. Hence, we assign equal importance to each known rating to avoid having large ratings steer our algorithm in one direction. This is achieved through normalization, meaning that we subtract from each known rating $r_{rs}$ the average of the average rating of user $r$ and movie $s$.

We experiment with different initialization methods for $U$ and $V$. One approach is to assign each element the same value in a way that each element of the product $UV$ corresponds to the global average of $R$. In general, this implies that the elements of $U$ and $V$ are initialized as $\sqrt{r_{\text{av}}/d}$, where $r_{\text{av}}$, given by equation (3), is the global average of $R$ and $d$ is the latent dimension of $U$ and $V$. After normalization we have $r_{\text{av}} = 0$, however to prevent divergence to infinity in equation (9) and (10), which occurs when $u_{rs} = v_{rs} = 0$ for all $r$ and $s$, we decide to use small initial values $u_{rs} = v_{rs} = 10^{-3}$. As an alternative initialization method, we use a normal distribution $\mathcal{N}(0, 0.1)$ of mean $\mu = 0$ and standard deviation $\sigma = 0.1$.

The order in which we visit the elements of $U$ and $V$ will have an impact on reaching the global minimum. In this work, we choose to update column-by-column the values of $U$, while we update row-by-row the values of $V$. Moreover, after having updated a value of $U$, the next value to be adjusted is from $V$ and vice-versa. We also experiment with an alternative optimization path, where we choose a random permutation of the elements and follow that order for every round. We define the optimization path as *ordered* if we perform the column-by-column and row-by-row update rule of $U$ and $V$ and *random* if we follow a random permutation (random seed 44) for the optimization path.

We stop the optimization algorithm after reaching convergence in the RMSE on the test set. Hence, training ends once there is no improvement in the RMSE after 400 iterations of updating a single

element of $U$ and $V$. This choice of iterations is motivated by the fact that it helps preventing getting stuck in a local minima and accelerates the code.

Lastly we experiment with different dimensions $d$ for the $U$ and $V$ matrices.

## 2.5 Matrix Factorization

Another approach for a recommender system is the Matrix Factorization algorithm which was originally proposed in a blog post in 2006 by S. Funk. The key idea behind the algorithm is identical to $UV$ matrix decomposition, in a sense that the utility matrix $R$ is factorized as the product of two matrices of lower dimensions. These matrices, defined as $U$ and $M$, are of the same dimensions as the $U$ and $V$ matrices from Section 2.4, and can be thought of as storing $d$ latent parameters (i.e generalities) of the data in the utility matrix $R$. These parameters are not necessarily observable variables, but can be thought of as movie genre, year of release or user gender in the case of movie recommendations. $U$ would store user features, whereas $M$ holds movie features. Both equations (7) and (8) are equivalent for matrix factorization, with the exception that we replace $V$ with $M$ and $m_{rs}$ being the corresponding elements.

To find the optimal values of $U$ and $M$ which minimize the RMSE between the product $UM$ and $R$, the algorithm of Takács et al. (2007) is implemented. Again, this is based on the Gradient Descent algorithm but will differ to the updating rules employed in the $UV$ matrix decomposition approach. The first step consists in calculating for a non-blank element $r_{rs}$ of $R$ its difference $e_{rs}$ to the predicted rating $\hat{r}_{rs}$

$$e_{rs} = r_{rs} - \hat{r}_{rs} = r_{rs} - \sum_{k=1}^{d} u_{rk} m_{ks}. \tag{11}$$

The goal is to minimize the sum $\text{SE} = \sum_{(r,s)\in\mathfrak{R}} e_{rs}^2$ where $\mathfrak{R}$ is the subset of all the known elements of $R$. This is equivalent to minimizing the RMSE and is achieved by updating the weights of $U$ and $M$ in the opposite direction of the gradient of SE. Hence, the generalized formulas for updating the $r^{\text{th}}$ row of $U$ and the $s^{\text{th}}$ column of $M$ are

$$u_{rk,\text{new}} = u_{rk} + \eta(2e_{rs}m_{ks} - \lambda u_{rk}), \tag{12}$$

$$m_{ks,\text{new}} = m_{ks} + \eta(2e_{rs}u_{rk} - \lambda m_{ks}), \tag{13}$$

where $k \in \{1, .., d\}$. Here, $\eta$ is the learning rate parameter which dictates the speed of convergence and can be varied to avoid convergence at a local minima whereas $\lambda$ is the regularization parameter used to prevent large weights and thus, is a measure against overfitting. We note here that unlike the $UV$ Decomposition algorithm, where each element is updated individually, the Matrix Factorization algorithm enables updating all the $u_{rk}$ and $m_{rk}$ (i.e $2d$ elements) simultaneously.

The preprocessing of the utility matrix consists again of normalization which is achieved in the same way as described in Section 2.4. We initialize the weights of $U$ and $M$ uniformly between $-0.01$ and $0.01$ with a random seed 10. The choice of a uniform distribution for the initialization is identical to that employed in Takács et al. (2007). Due to the time complexity of the algorithm, we only perform runs with $d = 5, 10, 15$ latent factors, a regularization term $\lambda = 0.05$ and a learning rate $\eta = 0.005$. We stop training after we observe convergence in the RMSE of the test set or after 75 iterations through the entire training set. These correspond to the parameters reported on the MyMedialite website. We emphasize that the definition of an iteration is different to that used in the $UV$ decomposition algorithm.

## 2.6 Data Visualization

As described in Section 2.5, the utility matrix $R$ can be factorized as a product of the $U$ and $M$ matrices. This is achieved by introducing a number of latent factors that help to describe the relationship between users and movies. To better understand these relationships, it is important to search for underlying patterns that may exist in the data. Although $U$ and $M$ are constructed at a lower dimension to $R$, these matrices will generally still be of high-dimensionality. Therefore, their interpretation can prove challenging due to the large number of variables present in the latent space which itself becomes increasingly large as the dimension increases (i.e. the curse of dimensionality). This can make the search for underlying patterns difficult, which is why it is necessary to consider dimensionality reduction techniques when analysing the results.

While there are several computational techniques for reducing a dataset's dimensionality, three in particular were explored in this assignment: Principal Component Analysis (PCA), t-distributed Stochastic Neighbor Embedding (t-SNE), and Uniform Manifold Approximation and Projection (UMAP). Although these procedures differ in operation, they all work to disentangle latent features from each other and offer insight into how the data is distributed. They are also important visualization tools that help to reduce the complexity of a model and represent underlying patterns in a more meaningful way.

When given an input set of several variables, PCA offers a way of explaining the global variation in large sets of data using a few number of variables. The operation of this scheme is to linearly transform the data into its components of greatest variance, i.e. the principal components (PCs). This is achieved by projecting the data onto the hyperplane lying closest to it such that the amount of variation retained in the data is maximized. The axis of this hyperplane that best explains the variance is the first PC, while the second PC is captured by the axis orthogonal to this. Although subsequent PCs exist depending on the dimension of the hyperplane, generally the first two PCs are most important as they encapsulate the greatest amount of variance in the data.

In contrast, the t-SNE approach transforms local similarities between data points to probabilities and reduces dimensionality by keeping similar instances close and dissimilar instances apart. Unlike PCA, the axes are arbitrary and it is the relative distances that are most important. Applied to our case, the high dimensional latent factors are modelled by 2D points based on the probabilities of their similarities. This means that, if two points are close to one another in the lower dimensional

depiction, they are likely to be close in the higher dimensional space. This is also true for the contrary, i.e. if the points are further apart from each other. This method is therefore very effective at finding and visualizing clusters that may exist in the data.

UMAP is similar to the t-SNE technique in that it also determines the relative distances between data points. The difference between both methods lies in how these distances are determined. t-SNE employs Gaussian probabilistic means to compute how likely a point most closely neighbours another. On the other hand, UMAP firstly creates a 'fuzzy' space that resembles the topology or shape of the high dimensional space. Calculating the weights for the edges of this space allows for it to be then compressed to a low dimensional (generally 2D) representation while retaining topological similarities. This is based on the so-called Nerve theorem, which states that a subset of points can accurately reflect the topology of a latent space. The t-SNE method performs computations on a point-to-point basis while UMAP does so more globally (i.e. on the shape of the space formed by all points). In other words, UMAP gives a rough estimation of the high dimensional space rather than measuring every point. For this reason, t-SNE is more computationally expensive.

For this assignment, PCA and t-SNE was performed using the `sklearn.decomposition` and `sklearn.manifold` packages respectively, while the UMAP routine was employed from the UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction python package.

# 3 Results

The computer that is used in this research is a Apple MacBook Pro M2.

## 3.1 Naive Approaches

Table 1: The average RMSE and MAE results on the five folds using the naive approaches. For comparison, RMSE and MAE values obtained from `http://mymedialite.net/examples/datasets.html` are written under the label mml. The website does not list RMSE and MAE values for linear regression approach. The run time corresponds to the duration of training over one fold.

| *Approach* | $\mathbf{RMSE}_{\text{train}}$ | $\mathbf{RMSE}_{\text{test}}$ | $\mathbf{RMSE}_{\text{mml}}$ | $\mathbf{MAE}_{\text{train}}$ | $\mathbf{MAE}_{\text{test}}$ | $\mathbf{MAE}_{\text{mml}}$ | Run Time ($s$) |
|---|---|---|---|---|---|---|---|
| Glob. Avg. | 1.117 | 1.117 | 1.117 | 0.934 | 0.934 | 0.934 | $\sim 1$ |
| Avg. User | 1.028 | 1.035 | 1.036 | 0.823 | 0.829 | 0.827 | $\sim 1$ |
| Avg. Item | 0.974 | 0.979 | 0.983 | 0.778 | 0.782 | 0.783 | $\sim 1$ |
| Lin. Reg. with Intercept | 0.914 | 0.924 | N/A | 0.725 | 0.732 | N/A | $\sim 12$ |
| Lin. Reg. without Intercept | 0.946 | 0.952 | N/A | 0.758 | 0.763 | N/A | $\sim 12$ |

The results of all naive approaches are summarized in Table 1. The RMSE and MAE values are nearly identical to those listed from My Media Lite, which suggests that the methods have been correctly implemented. The differences are negligible and could be a consequence of different random data splittings employed in both experiments.

Overall, we observe an improvement in error values from the global average approach to the linear regression with intercept approach, which implies that subsequent approaches perform better as recommender systems. Within this trend, it is interesting to see that predictions from the average item method are better than that of the average user approach. A possible explanation is the larger number of users than items, resulting in more data to be summed per item than per user. Hence, the average item ratings could be more representative than the average user ratings. Another thing to note is that the RMSE and MAE values are stable over the 5 folds for each recommender system, thus indicating the stability of the results presented in Table 1. Lastly, the increase in RMSE and MAE values from the linear regression without intercept method suggests that our data is not centered.

## 3.2 UV Matrix Decomposition

The results of the $UV$ matrix decomposition method are summarized in Table 2. We have carried out several runs, using different initialization methods, optimization paths and dimension $d$ for the $U$ and $V$ matrix. We obtain the best final average error over the five folds for *run 1*, with a $\text{RMSE}_{\text{test}} = 0.927$ and $\text{MAE}_{\text{test}} = 0.731$ on the test set. This is almost identical to the results obtained from the full variant of linear regression models and is an improvement to the remaining naive methods. For this run, we initialize the $U$ and $V$ matrix elements by the same value $10^{-3}$, employ an ordered optimization path for updating elements, and set $d = 2$ for the dimension of the $U$ and $V$ matrix. The learning curves on the training and test set on one of the five folds for *run 1* are shown in Figure 1. Notice that we relax the stopping condition of no improvements in $\text{RMSE}_{\text{test}}$ for the first 4000 iterations of the algorithm due to the initial rise in $\text{RMSE}_{\text{test}}$, which is then followed by a decrease. Both learning curves converge after 12000 iterations at which point each element of the $U$ and $V$ matrix has been updated at least once.

The second best results are obtained for *run 5*, which is identical to *run 1* with the exception of an increased dimension $d = 3$. We obtain $\text{RMSE}_{\text{test}} = 0.928$ and $\text{MAE}_{\text{test}} = 0.731$ on the test set, which are almost identical to those found in *run 1*. The absence of an improvement in *run 5* comes as a surprise, given the increased time and memory complexity associated with higher $d$. Moreover, analyzing the results of the other runs, we discern that in general the algorithm performs better for a lower dimension $d = 2$ compared to $d = 3$. This does not imply that increasing $d$ leads to weaker recommender systems but most likely suggests that a different initialization method or optimization path should be used to obtain improved results for higher $d$. The parameter space for the UV Matrix Decomposition algorithm is very large and our results reflect the high sensitivity of the algorithm on the initial conditions of $U$ and $V$ and the chosen optimization path. From our limited experiments, we find that an ordered optimization path with small initial identical values for $U$ and $V$ produces the best results. It would be interesting to experiment with more random permutations for the optimization path and increase the dimension $d$ further to improve our recommender system using the $UV$ matrix decomposition algorithm.

Table 2: The average RMSE and MAE results on the five folds using the UV matrix decomposition. Each run corresponds to a different experiment involving the initialization of the $U$ and $V$ matrix, optimization path and the dimension $d$. The run time corresponds to the duration of training over one fold. The short times are a result of the early end of training.

| *# Run* | $u_{rs,\text{init}}$ | Opt. Path | $d$ | $\textbf{RMSE}_{\text{train}}$ | $\textbf{RMSE}_{\text{test}}$ | $\textbf{MAE}_{\text{train}}$ | $\textbf{MAE}_{\text{test}}$ | Run Time ($s$) |
|---|---|---|---|---|---|---|---|---|
| 1 | $10^{-3}$ | ordered | 2 | 0.895 | 0.927 | 0.707 | 0.731 | $\sim 15$ |
| 2 | $10^{-3}$ | random | 2 | 0.932 | 0.973 | 0.747 | 0.776 | $\sim 15$ |
| 3 | $\mathcal{N}(0, 0.1)$ | ordered | 2 | 0.934 | 0.966 | 0.751 | 0.774 | $\sim 15$ |
| 4 | $\mathcal{N}(0, 0.1)$ | random | 2 | 0.924 | 0.957 | 0.741 | 0.766 | $\sim 15$ |
| 5 | $10^{-3}$ | ordered | 3 | 0.895 | 0.928 | 0.707 | 0.731 | $\sim 20$ |
| 6 | $10^{-3}$ | random | 3 | 0.939 | 0.967 | 0.754 | 0.774 | $\sim 20$ |
| 7 | $\mathcal{N}(0, 0.1)$ | ordered | 3 | 0.941 | 0.964 | 0.757 | 0.774 | $\sim 20$ |
| 8 | $\mathcal{N}(0, 0.1)$ | random | 3 | 0.942 | 0.964 | 0.758 | 0.775 | $\sim 20$ |

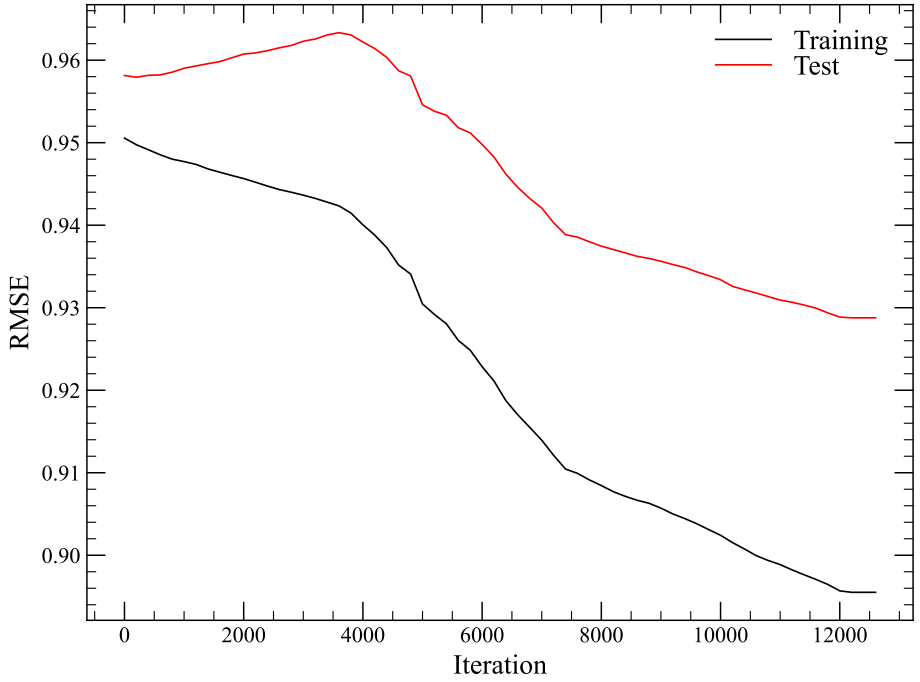


Figure 1: Learning curves for our best run (run 1) using the UV matrix decomposition algorithm of one of the five folds. The black and red curve correspond to the evolution of the RMSE on the training and test set respectively.

## 3.3 Matrix Factorization

Table 3 shows the results of the Matrix Factorization algorithm, with $\lambda = 0.05$, $\eta = 0.005$ and varying latent dimension $d$. For $d = 10$, we achieve an average $\text{RMSE}_{\text{train}} = 0.760$ and $\text{MAE}_{\text{train}} = 0.597$ on the training set of the five folds after 75 iterations through the algorithm. On the test set, we achieve an average $\text{RMSE}_{\text{test}} = 0.862$ and $\text{MAE}_{\text{test}} = 0.673$ which represents the best results. This is a significant improvement over our naive approaches and the $UV$ decomposition algorithm-based recommender systems. Figure 2 shows the learning curves of the Matrix Factorization algorithm on one of the folds of the dataset. It can be concluded that the training is much more

Table 3: The average RMSE and MAE results on the five folds using the matrix factorization algorithm with varying latent dimension $d$. For each model we used $\lambda = 0.05$ and $\eta = 0.005$. The run time corresponds to the duration of training over one fold after 75 iterations through the training set.

| $d$ | $\mathbf{RMSE}_{\text{train}}$ | $\mathbf{RMSE}_{\text{test}}$ | $\mathbf{MAE}_{\text{train}}$ | $\mathbf{MAE}_{\text{test}}$ | Run Time ($s$) |
|---|---|---|---|---|---|
| 5 | 0.812 | 0.866 | 0.639 | 0.679 | $\sim 610$ |
| 10 | 0.720 | 0.862 | 0.597 | 0.673 | $\sim 680$ |
| 15 | 0.723 | 0.870 | 0.567 | 0.677 | $\sim 750$ |

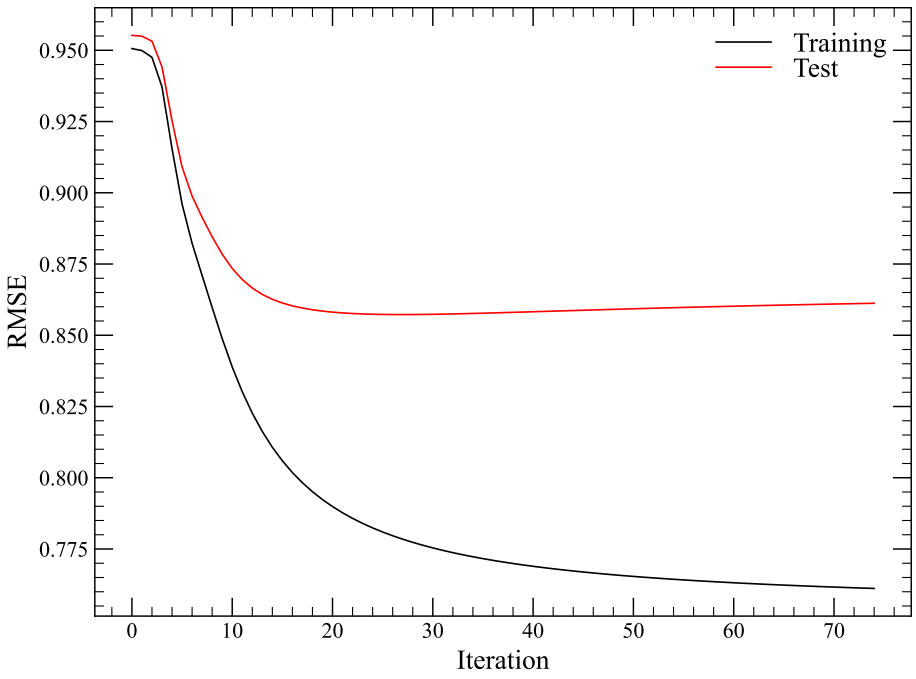


Figure 2: Learning curves using the matrix factorization algorithm of one of the five folds. The black and red curve correspond to the evolution of the RMSE on the training and test set respectively.

stable than in the $UV$ decomposition approach, where the algorithm did not exhibit learning in the early stages (see red curve in Figure 1). We actually find that the model is slightly overfitting after 75 iterations, given that the best results on the test sets are obtained at iteration 27. At that point we obtain an average $\text{RMSE}_{\text{test}} = 0.858$ and $\text{MAE}_{\text{test}} = 0.672$. The final results of the recommender systems with $d = 5$ and $d = 15$ are very similar to our best model with $d = 10$. Similarly to the $UV$ decomposition algorithm, we expect improvements in the Matrix Factorization algorithm with increasing dimension $d$. The absence of this suggests that different free parameters $\lambda$ and $\eta$ should be employed to achieve this. It is important to emphasize that having a too large number $d$ not only increases the algorithm's time and memory complexity, but also increases its susceptibility to overfitting the training data. This is reflected by the reduced final $\text{RMSE}_{\text{train}}$ on the training set for higher $d$, although no improvements in the $\text{RMSE}_{\text{test}}$ are discerned.

The presence of a regularization term and learning rate cause the Matrix Factorization algorithm to be superior to the simple $UV$ decomposition method and the increased time and memory complexity help outperforming the naive approaches.

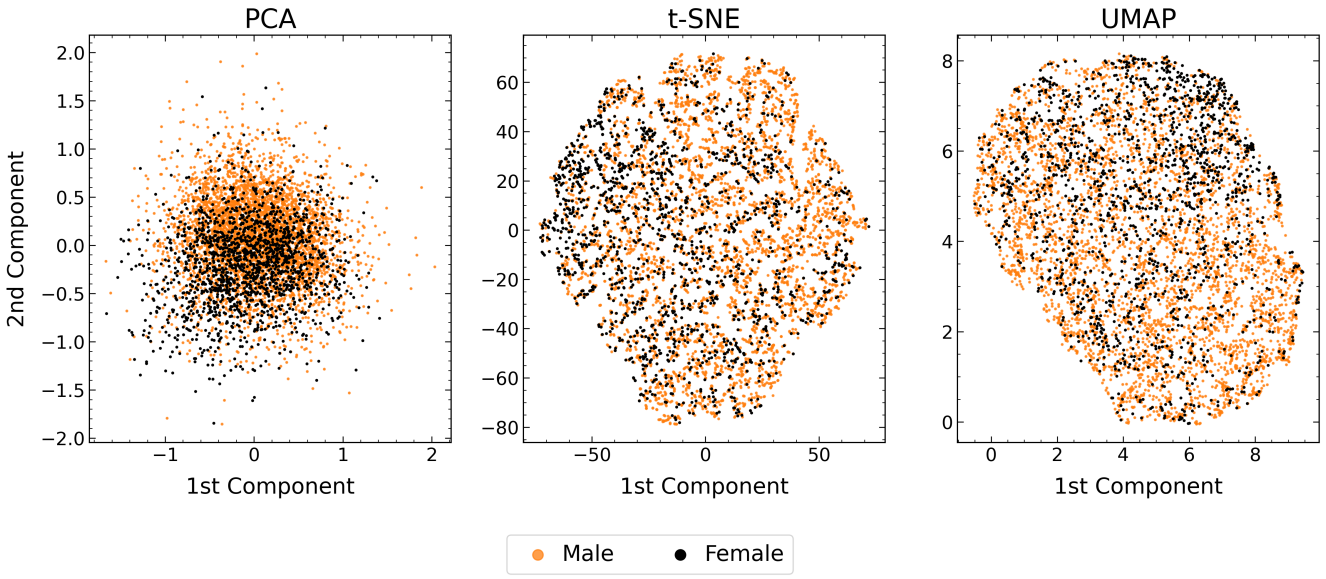


Figure 3: Result of applying the three dimensionality reduction techniques to the $U$ matrix of user features based on the gender attribute.

## 3.4 Data Visualization

As described in Section 2.6, the PCA, t-SNE, and UMAP algorithms help visualize any underlying patterns that may exist in the data. These methods were implemented on the user and movie matrices ($U$ and $M$, respectively) obtained from performing the matrix factorization algorithm on the dataset. This was done so to reduce the dimensionality of each dataset such that the user and movie features could be visualized in two-dimensions. The results presented here were found using the matrices computed with a latent dimension of $d = 5$. We choose to show these results in particular as they give the most meaningful insight into how the data is distributed, and include more significant clustering when compared to the matrices with $d = 10$ and $d = 15$.

For visualizing $U$, the three dimensionality reduction techniques were employed with a labelling scheme based on the gender, age, and occupation attributes. In summary, the results of these applications did not uncover much about how this data is distributed. No tangible patterns or clusters could be resolved for each of the algorithms when applied to the age and occupation labelling schemes (the results of which can be seen in Figure A1). However, this does make sense intuitively as it is reasonable to expect users of different age groups or occupations to have movies tastes that do not correlate with each other. It can be argued, however, that the results when using a gender labelling scheme exhibit some clustering. This is demonstrated by the PCA plot in Figure 3, where the male and female data points are somewhat clustered and can be distinguished from each other, suggesting that the user data can be reasonably summarized by the gender attribute. It should be noted that, unlike the t-SNE and UMAP algorithms, PCA captures similarities on a global scale, hence why there is little information to be gleaned from the t-SNE and UMAP results.

Following this, these algorithms were applied to the $M$ matrix to visualize movie features with a labelling scheme based on the year of release and genre attributes. Due to the wide range of years and genres within the data, the resulting plots are difficult to interpret simply because they are

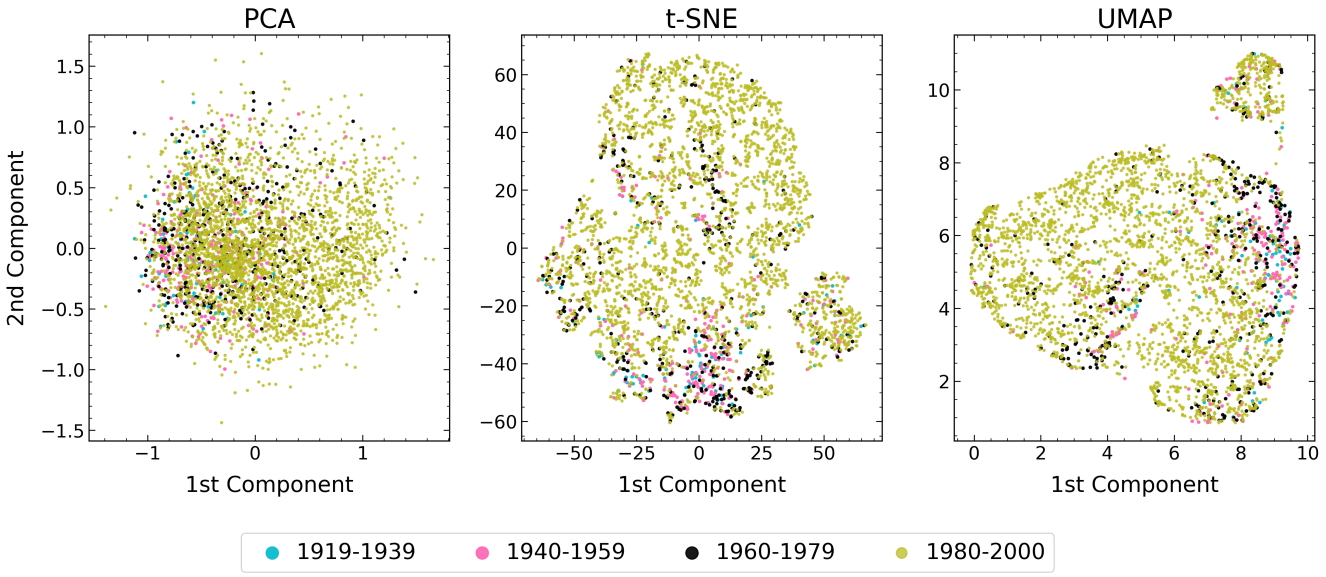


Figure 4: Result of applying the three dimensionality reduction techniques to the $M$ matrix of movie features based on the year of release attribute. This particular labelling scheme is categorized into time periods of 20 years each.

over-populated with data points belonging to too many categories (see Figure A2 for such plots). We therefore experimented with dividing the data in different ways to search for interesting and meaningful insights. For example, Figure 4 shows the results when the year of release labelling scheme is categorized into time periods spanning 20 years each. As observed in the t-SNE and UMAP plots, movies released before 1980 tend to cluster together and separate themselves from the spread of movies released after 1980. This suggests that there exists non-observable relationships between movies in the data that can be considered 'old'. It should be considered however that there is a large imbalance between these movies in the dataset (i.e. there are many more movies released after 1980 than there are before).

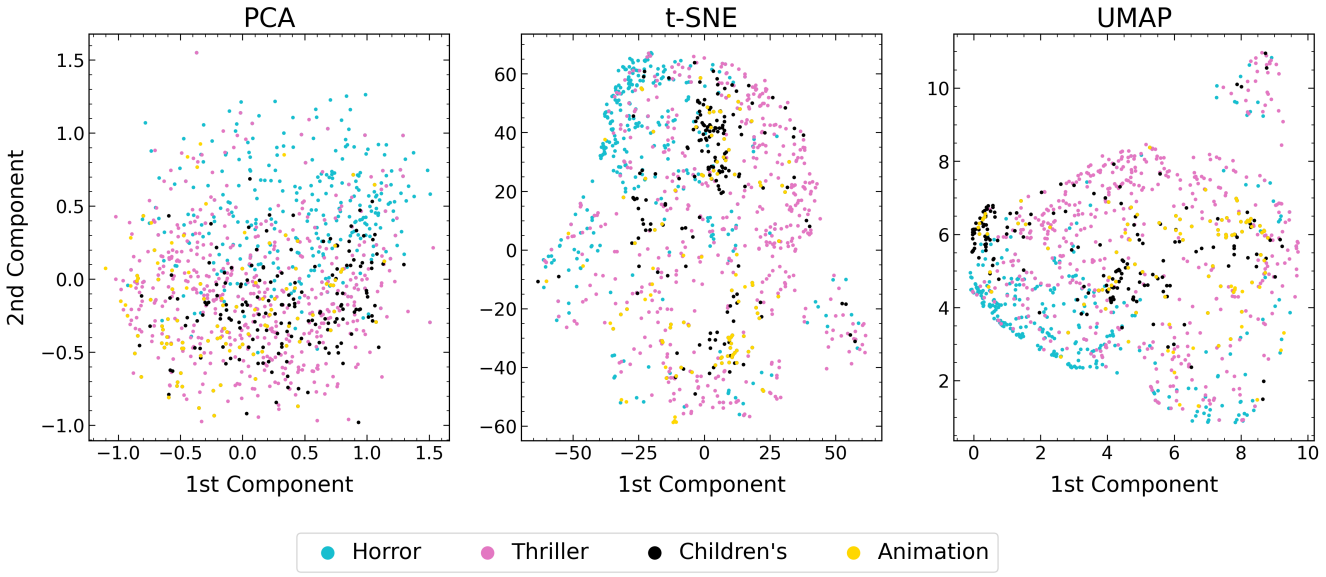


Figure 5: Result of applying the dimensionality reduction techniques to $M$ based on a subset of the genre attribute.

Table 4: Time and memory complexity of the algorithms presented in this work. We quantify the complexity using the big-O notation. $r$ represents the number of ratings, $m$ is the number of movies, $u$ corresponds to the number of users and $d$ is the latent dimension in the $UV$ decomposition and Matrix Factorization algorithm. We note that $r = um$.

| Algorithm | Time Complexity | Memory Complexity |
|---|---|---|
| Global average | $\mathcal{O}(r)$ | $\mathcal{O}(1)$ |
| Movie average | $\mathcal{O}(r)$ | $\mathcal{O}(m)$ |
| User average | $\mathcal{O}(r)$ | $\mathcal{O}(u)$ |
| Linear regression | $\mathcal{O}(r)$ | $\mathcal{O}(r)$ |
| UV decomposition | $\mathcal{O}(rd)$ | $\mathcal{O}(r)$ |
| Matrix factorization | $\mathcal{O}(rd)$ | $\mathcal{O}(r)$ |

When employing a labelling scheme based on the genre attribute, interesting results were found when the genres were limited to a subset containing the Horror, Thriller, Children‘s and Animation genres, as shown in Figure 5. In each plot, there are a number of regions that exhibit clustering behaviour. This is especially clear in the t-SNE and UMAP plots, where many data points belonging to the Horror and Thriller genres group together in close proximity. Moreover, those belonging to the Children‘s and Animation genres cluster together and do so at a further distance away from the Horror-Thriller clumps, showing how these genre groupings differ from each other. Both the similarities and dissimilarities presented in these plots make sense intuitively; it is reasonable to assume correlations between genres of similar types (e.g. horrors and thrillers) as well as the contrary, i.e. that horror movies would not be strongly correlated with children's movies.

# 4 Discussion

## 4.1 Time and Memory Complexity

Each recommender system in this work exhibits a different performance on the dataset. These differences can be understood in terms of the time and memory complexity involved in each algorithm. Table 4 shows the time and memory complexity, using the big O-notation, of each algorithm. This shows how their time and memory scale with the data. In general, the more expensive the algorithm, the better the final accuracy of the recommender system. In this work, this applies to the Matrix Factorization algorithm, where time scales linearly with both the number of users, movies (i.e $r = um$) and the latent dimension $d$. On the other hand, the time and memory of the $UV$ decomposition algorithm scales identically to that of the Matrix Factorization. However, this produces worse results due to the high sensitivity of the algorithm's initialization method and optimization path. Although, Matrix Factorization performs best, this method may not be the appropriate choice if the dataset increases significantly. In that case, it would be more feasible to employ a naive approache such as the average user/movie rating or the linear regression model, which exhibit smaller time complexities and require a reasonable amount of memory.

# 5 Conclusion

We have developed recommender systems for the MovieLens 1M dataset based on various algrithms. We found that the most simple systems, based on the naive approaches, produce surprisingly good results despite their low time and memory complexity. The linear regression with intercept model performs best with a $\text{RMSE}_{\text{test}} = 0.924$ and $\text{MAE}_{\text{test}} = 0.732$ over the five test folds. This model is able to compete with the more sophisticated $UV$ matrix decomposition algorithm, for which our best model gives $\text{RMSE}_{\text{test}} = 0.927$ and $\text{MAE}_{\text{test}} = 0.731$. This is achieved for a latent dimension $d = 2$, initial weights of $10^{-3}$ and an ordered optimization path. We found that the performance of the $UV$ decomposition approach is very sensitive to these parameters and as a result conclude that better results exist. Moreover, we used Matrix Factorization to predict ratings, for which we achieve the best overall results among all algorithms, with $\text{RMSE}_{\text{test}} = 0.862$ and $\text{MAE}_{\text{test}} = 0.679$ when using $d = 10$, $\lambda = 0.05$ and $\eta = 0.005$. It is likely that we have not reached the true global minimum and thus, we expect that improvements are possible for different values of $\lambda$ and $\eta$.

We used the latent factors of the $U$ and $M$ matrix, produced by the Matrix Factorization algorithm, to discern non-observable relationships in the rating data. For this purpose, we employed PCA, t-SNE and U-map techniques. In general, it is difficult to observe trends in the rating data given the unique preferences of each person. However, some intuitive trends could be discerned using t-SNE and U-map such as the clustering of horror movies and their proximity to thrillers and the same trend applies to family friendly movie genres. Moreover, we observed a clear separation between old and recent movies. These observations reflect that there exists users with similar preferences and that collaborative filtering systems could be used as alternative recommender models. Furthermore, the recommender systems on the dataset can be further improved by implementing more sophisticated algorithms such as deep neural networks. In this work, we considered supervised machine learning approaches and it would be interesting to experiment with unsupervised machine learning methods.

# References

Harper, F. M., & Konstan, J. A. 2015, ACM Trans. Interact. Intell. Syst., 5, doi: 10.1145/2827872

Leskovec, J., Rajaraman, A., & Ullman, J. 2014, Mining of massive datasets (Cambridge University Press)

Takács, G., Pilaszy, I., Németh, B., & Tikk, D. 2007, in On the Gravity Recommendation System

# A Data Visualization

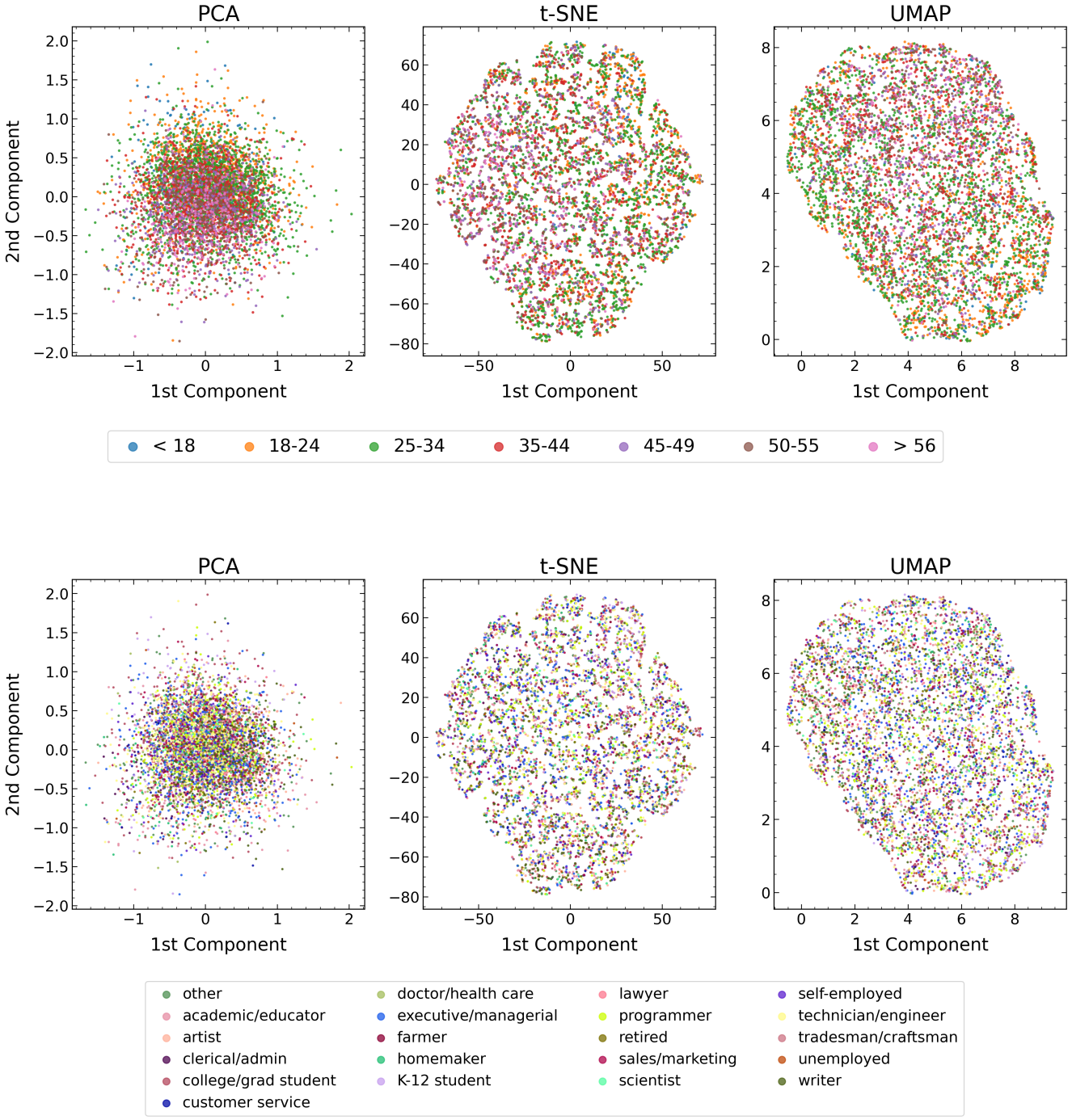


Figure A1: Results of applying the three dimensionality reduction techniques to the $U$ matrix of user features based on the age (*top*) and occupation attributes (*bottom*).

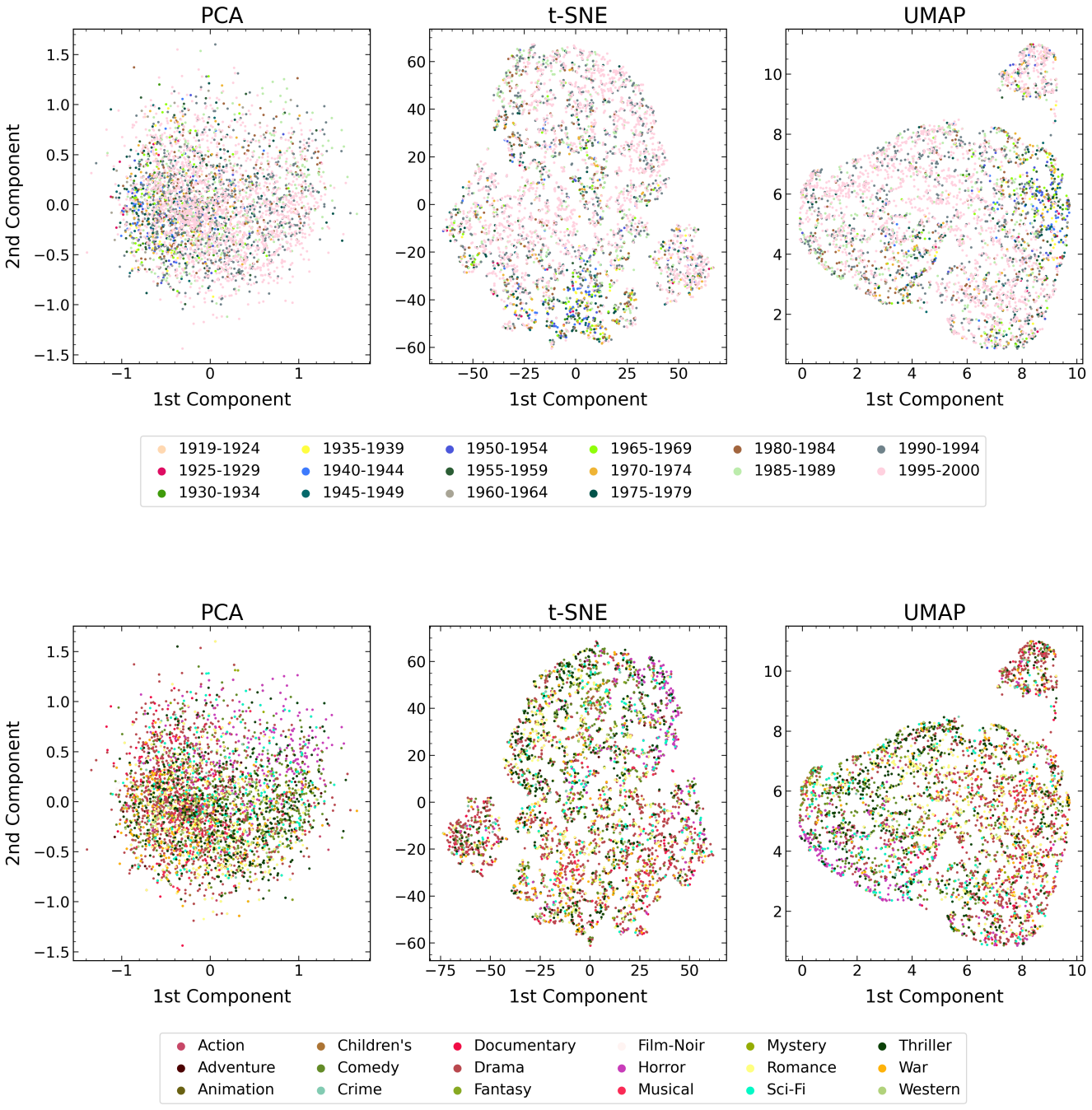


Figure A2: Results of applying the three dimensionality reduction techniques to the $M$ matrix of movie features based on the year of release (*top*) and genre attributes (*bottom*).