Advances in Data Mining: Assignment 1

Group 4

October 25, 2022

1 Introduction

Netflix is one of the largest streaming services that offer users a huge variety of TV shows and movies. However, many users experience recommendations that are not perfectly suitable for their taste, which might cause a loss of potential customers. Thus, well-working recommendation systems are required in order to maintain Netflix as a successful streaming service.

The *MovieLens 1M* dataset that we are working with is downloaded from GroupLens[1]. It contains 1.000.209 ratings of 6040 users for 3883 movies. Demographic information is also available for the users. Furthermore, movie information, such as genres is included in the dataset as well.

In this report, we present several naive approaches, as well as some advanced methods, namely the UV-decomposition and matrix factorisation. The structure of this report is the following: in section 2 we describe the recommendation algorithms. section 3 presents the results of our methods including the visualisation of the dataset and section 4 shows their comparison.

2 Method

2.1 Naive Approaches

For this part we construct five different algorithms that generate predictions for the ratings based on the "global average" , the "average rating per item", the "average rating per user", and a "linear combination of the two averages" with and without the intercept parameter.

The "Global Average" approach takes the average of all the ratings on the training set and then assigns this value as the predicted rating of all the unknown ratings on the testing set.

$$R_{global}(user, item) = mean(all\ ratings)$$

The "Average Rating per Item" calculates the average rating of each movie that is present in the training set. Then for every unknown rating in the testing set, it gives the average rating of this particular movie. If a movie is present in the testing set but not in the training set then there is no average for that particular movie so we assign it the value of the total average of all the ratings in the training set. When using this approach there are gaps in the IDs of the movies. In order for the code to work we map all the IDs in a new array that has no gaps, use this new array for all the calculations and then assign the average to the corresponding movie ID.

$$R_{item}(user, item) = mean(all\ ratings\ per\ item)$$

For the next approach, we use the "Average Rating per User". This time we look at each user that is present in the training set and calculate the average of all the ratings they gave (that is part of the training set). Then we generate the predictions for the testing set based on this average. If a user is not found in the training set, but we have to give a prediction in the testing set, we then use again the average of all the ratings of the training set. Since there are no gaps in the user IDs (in contrast to the movie IDs) the technique used in the previous approach with the mapping of the IDs is not necessary.

$$R_{user}(user, item) = mean(all\ ratings\ per\ user$$

The next two approaches take it one step further and use a linear regression model to generate the predicted ratings. For each rating, we have the user that gave the rating and the ID of the movie. We can take the average rating this particular user has given to all the movies in the training set and the average rating this particular movie has received in the training set. We are thus creating a list

[1] http://grouplens.org/datasets/movielens/

of $(x_i, y_i, z_i) -> (user\ average\,, item\ average\,, rating)$. We then perform a linear regression fitting using the sklearn.LinearRegression package for $z_i = \alpha \cdot x_i + \beta \cdot y_i (+\gamma)$. The only difference between the two approaches is the use of the intercept parameter γ, one is using it and the other one is not. For an unknown rating of the testing set, we can generate a prediction based on the average rating of the particular user (in the training set) that should give this rating and the average rating of the particular title of the movie (in the training set). Once again the average rating of all the ratings in the training set is used as a "fallback" value if no ratings are given by a user or for a movie that exists in the testing set. We are also checking if any ratings predictions have a value smaller than 1 or greater than 5 and substitute it with 1 and 5 respectively.

$$R_{LR\,with\,\gamma}(user, item) = \alpha \cdot R_{user}(user, item) + \beta \cdot R_{item}(user, item) + \gamma$$

$$R_{LR\,without\,\gamma}(user, item) = \alpha \cdot R_{user}(user, item) + \beta \cdot R_{item}(user, item)$$

2.2 Advanced methods

2.2.1 UV Decomposition

A commonly used method is called the UV-decomposition in which we aim to estimate user ratings. In this process, we have an M utility matrix and we create two new matrices with random numbers. A user matrix U with n rows and d columns, and a rating matrix V with d and m rows and columns, respectively. Note, that in the code, M needs to be converted into a matrix from the csv table. In our case, the number of rows is equal to 6040 users and the columns equal to 3952 movies. The main purpose of this method is to update U and V in a way to get a new UV matrix that is as close to the non-blank M matrix as possible. The algorithm of our approach is the following:

- Firstly, we initialise U and V by creating random numbers following $\sqrt{\frac{mean(R)}{d}}$, where R is the average user rating
- Then, we update U and V by randomly selecting elements in the initial U and V.
- Finally, we calculate the errors of the updated U and V.

Overall, 20 steps are taken into account in this approach, because based on our tests, larger number of steps does not lead to a better result.

2.2.2 Matrix factorisation

In this section, we discuss the implementation of Matrix factorisation Algorithm in Python. In order to complete the matrix factorisation, we will create a matrix called X where X_{ij} denotes the rating of the i_{th} user for j_{th} movie. The size of X is (6040,3952) because there are 6040 users and 3952 movies in our dataset. However, not every user has rated every movie, thus we substitute zero for all empty values while creating the initial matrix. Additionally, we make an effort to maintain all ratings between 1 and 5, therefore any rating below 1 is put to 1, and any rating beyond 5 is set to 5.

The following equations and explanations are referred from *gravity-tikk* paper by Takács et al (2007). The matrix X can be defined as the dot product of two matrices U and M, where U is the user matrix and M is the movie matrix. Matrix X has the shape of (i,j) whereas matrices U and M are the shape of (i, k) and (k, j) respectively, where k value is calculated during our experimentation. The fundamental approach is that, based on previously known ratings, the product of the matrices U and M will provide precise forecasts of the ratings that a given user would assign to movies that they haven't yet rated.

We initialize the matrices U and M with zeros. The elements of U and M are computed with the following equations by Takács et al (2007) and are updated.

$$\hat{x}_{ij} = \sum_{k=1}^{K} = u_{ik} m_{kj}$$

$\hat{x}_{ij}$ denotes the predicted values of what user i will rate movie j. The error between the predicted value and the known value is then calculated. The goal is to reduce the overall amount of error over time.

$$e_{ij} = x_{ij} - \hat{x}_{ij}$$

The total error of all these ratings is the sum of the errors:

$$SE = \sum_{ij} e_{ij}^2$$

$$u`_{ik} = u_{ik} + \eta \cdot (2e_{ij} \cdot m_{kj} - \lambda \cdot u_{ik})$$

$$m`_{kj} = m_{kj} + \eta \cdot (2e_{ij} \cdot u_{ik} - \lambda \cdot m_{kj})$$

The above mentioned two algorithms by Takács et al (2007) are used to calculate the updated values for the elements in the matrices U and M.

The new parameter value is $u`_{ik}$. The learning rate, η, indicates how quickly the algorithm learns. The regularization factor to avoid large weights is λ. After computing the matrices on the training set, we proceed to predict the values using $\hat{x}_{ij}$ equation to compute the U and M matrices on the test set. The total error is then subsequently computed.

3 Results

For all the different methods we use the "5-fold cross-validation scheme". The data are split randomly into 5 parts. We then train the algorithm 5 different times each time testing with a different part of the split data and training with the rest. After that, we average both the training and the testing errors. By doing this the errors generated should be a better estimate of the error we should get in future data.

For the splitting we implemented the KFold function from sklearn.model_selection[2], setting the random state at "1" in order to make the code reproducable.

The accuracy of the recommendation algorithms is checked by calculating the Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_{true} - y_{predicted})^2}{n}}$$

and the Mean Absolute Error (MAE)

$$MAE = \frac{\sum_{i=1}^{n}|y_{true} - y_{predicted}|}{n}$$

3.1 Naive Approaches

The results of the five different methods, described in 2.1, that were implemented for the Netflix challenge are given in Table 1. After the training process was over, the algorithm generated predictions for the ratings of both the training and the testing set and then compared it with the actual values of the ratings.

Model	RMSE Train	RMSE Test	MAE Train	MAE Test	Execution Time (s)	Memory (MB)
Global Average	1.117	1.117	0.934	0.934	15.330	375.525
Movie Average	0.974	0.980	0.778	0.782	88.707	389.423
User Average	1.028	1.035	0.823	0.829	55.218	389.444
Lin Reg with intercept	0.915	0.924	0.725	0.733	178.778	416.600
Lin Reg without intercept	0.947	0.953	0.759	0.763	172.274	447.537

Table 1: The final results of all the naive approaches. For each model we have both the average RMSE and the average MAE for both the trainings and the testings datasets. The running time of each algorithm is also included in seconds, as well as the memory usage

By examining the results of the errors from the five different approaches, we can see that they are performing well, especially the more "complicated" ones making use of the linear regression to generate the predictions. The errors on the testing set are increased compared to the ones for the training set as it would be expected, however, there is not a significant increase which would suggest that there is a case of over-fitting the data.

One interesting observation of these algorithms is the time required to run each algorithm. This becomes more clear in Figure 1 where we see that the better performance of the more "complicated" algorithms comes with the cost of increased computational time. The linear regression model using the intercept has no increase of the execution time compared with the one without an intercept and increases the accuracy by a significant margin. This means that the "no intercept linear regression model" is not effective as there is an alternative with the same execution time and better results.

Lastly, we also note the memory usage while running each algorithm. From Figure 2 we can see the comparison. Once again in order to decrease the errors, the more complicated algorithms being used require more memory. The calculations of the linear regression require a lot more memory than the simpler ones where we only have the averages of either the item or the user. Also, the linear regression

[2]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

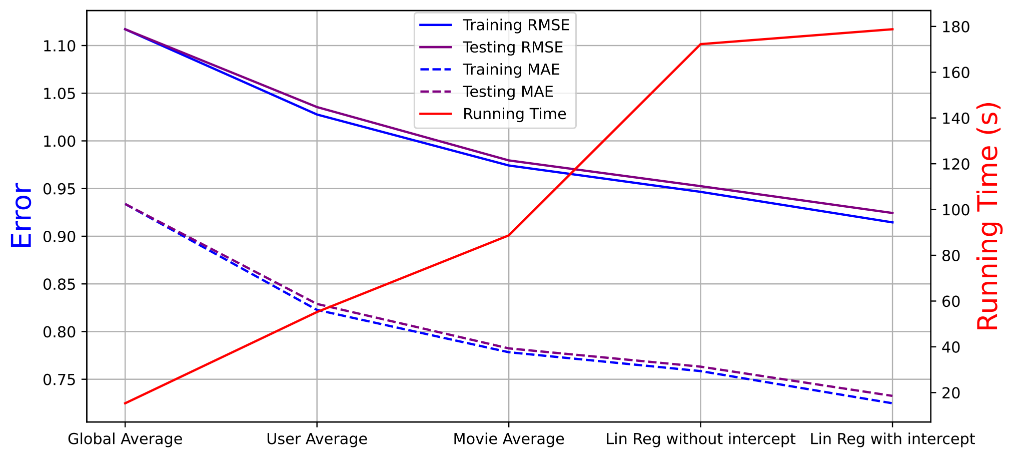


Figure 1: The RMSE (solid lines) and MAE (dotted lines) for both the training and the testing for the 5 different Naive Models from worst to best compared with the running time in seconds.

without intercept requires even more memory than the one with the intercept, while providing worse errors. This in addition to requiring the same time to perform makes it less effective.

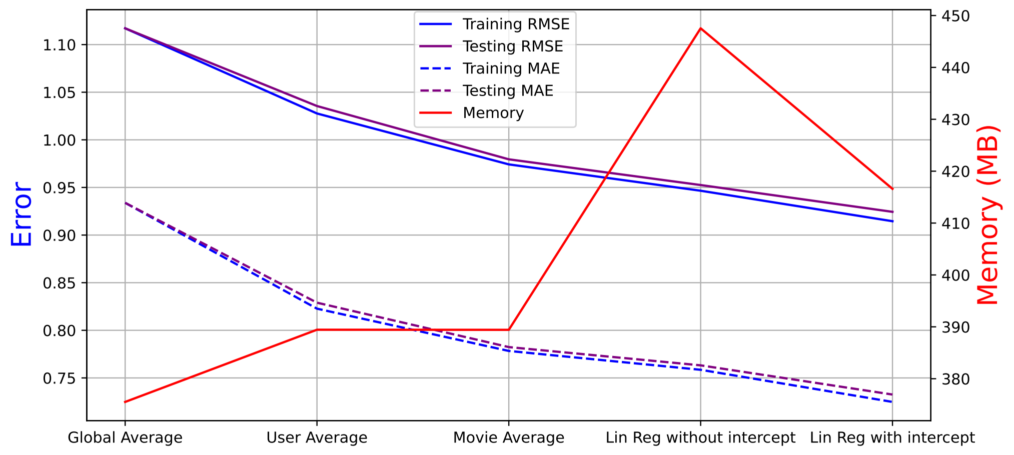


Figure 2: The RMSE (solid lines) and MAE (dotted lines) for both the training and the testing for the 5 different Naive Models from worst to best compared with the memory usage in MB.

For the “Global Average” approach the running time should be constant or in big O notation $O(1)$. The “Movie Average” and “User Average” approach is linear with respect to the number of movies (m) and the number of users (u), meaning $O(m)$ for the first one and $O(u)$ for the latter. For the linear regression methods both the number of movies and users is relevant so $O(m + u)$.

3.2 UV Decomposition

In Fig. 3 we present the results of the UV-decomposition. 20 steps are shown, which are enough to converge to the minimum error. It clearly shows that the RMSE is larger than the MAE, as expected. In addition, the testing dataset is also above the training dataset. While the UV-decomposition lasts 503.424 seconds, converting the table also takes a significant amount of time (318.644 s). Therefore, the final execution time is 822.069 seconds. Furthermore, the required memory during the UV-decomposition is 1948037120 bytes (i.e. ∼1948 MB). A summary of this method with the final minimum errors can be found in Table 2.

The required memory of this approach is O(U+V+R), where U, V, and R is the $n \times d$, $d \times m$, and $n \times m$ matrix.

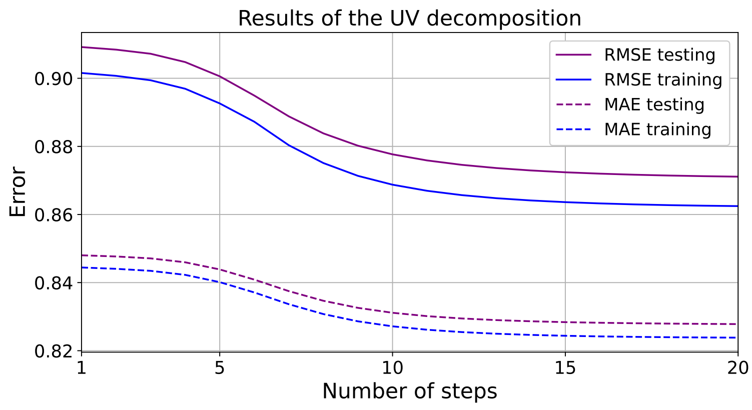


Figure 3: The RMSE (solid lines) and MAE (dotted lines) as a function of steps for the testing and training in purple and blue, respectively.

RMSE train	MAE train	RMSE test	MAE test	Execution time (s)	Required memory (MB)
0.862	0.823	0.871	0.828	822.069	1948.0

Table 2: The final results of the UV-decomposition: the RMSE and MAE for both the testing and training dataset, the execution time in seconds and the required memory in bytes.

3.3 Matrix factorisation

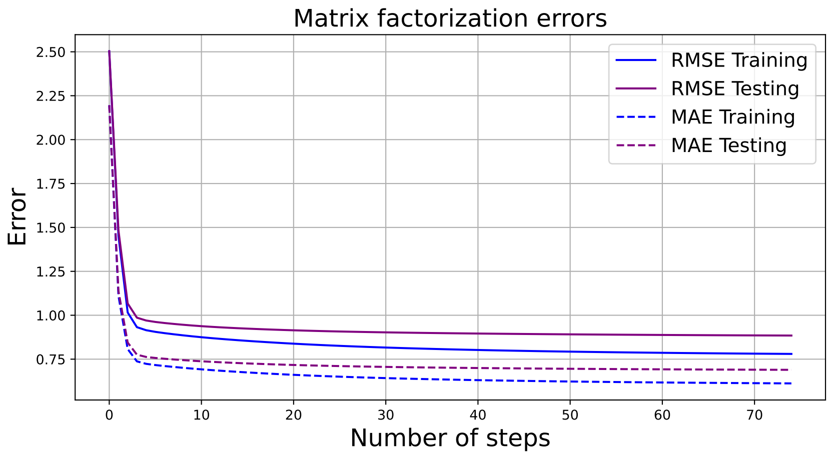


Figure 4: The RMSE (solid lines) and MAE (dotted lines) as a function of steps for the testing and training in purple and blue, respectively, for $\eta = 0.005$ and $\lambda = 0.05$

Figure 4 shows the plot of the errors (RSME and MSE) calculated on the training and testing datasets after Matrix factorisation. Take note of the rapid reduction of error at the beginning and the slow decrease in the last iterations. We observed that the RSME value often begins high, around 2.5 or 3, reduces rapidly in the first few iterations, and then gradually lowers to 0.7 or 0.8, depending on the starting parameter values for the learning rate and regularization value. If the error increases, the iterations will be stopped. This signifies that the algorithm cannot be improved any more.

Table 3 shows the final results after matrix factorisation. We can observe that Root mean square error (RSME) is higher than Mean absolute error (MAE) and it takes a total run time of approximately 2054 seconds to complete the training.

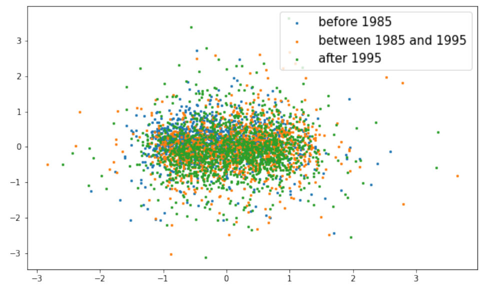
RMSE train	MAE train	RMSE test	MAE test	Execution time (s)	Required Memory (MB)
0.780	0.612	0.885	0.689	1190.862	2054.6

Table 3: The final results of the Matrix factorisation: the RMSE and MAE for both the testing and training dataset, the execution time in seconds.

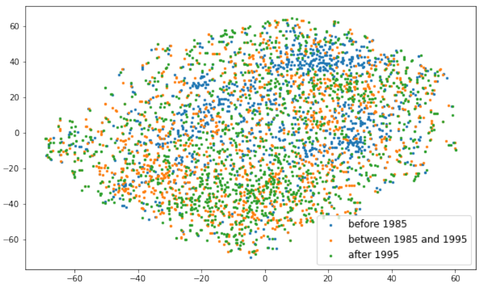
The required memory of this approach is O(U+M+R), where U, M, and R is the $i \times f$, $j \times f$, and $i \times j$ matrix.

3.4 Data visualisation

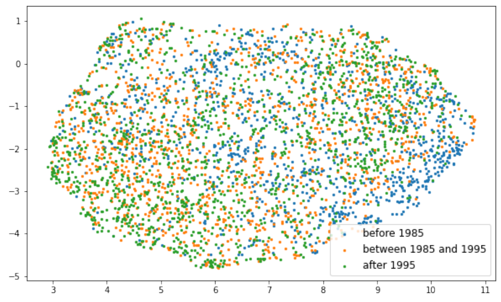
To visualise our data and search for patterns in 2D, we make use of *PCA*[3], *T-SNE*[4], and *UMAP*[5], which are dimensionality reduction techniques.



(a) PCA



(b) T-SNE



(c) UMAP

Figure 5: Clustering of movies based on their release year with (a) PCA, (b) T-SNE, and (c) UMAP. Different colours show different period as they are labeled.

PCA was the first dimensionality reduction technique that was developed more than a hundred years ago (Pearson, 1901). It aims to preserve the global structure of our data and its main purpose is to spread out our data by maximising the variance and performing a linear projection. Thus, it can cluster our data.

T-SNE was first published in a paper in 2008 by Van der Maaten & Hinton (2008). The T-SNE method is similar to PCA, but it is a stochastic and iterative method. Furthermore, it is non-linear, therefore, it is generally much slower than PCA. T-SNE aims to minimise Kullback–Leibler divergence between two distributions with respect to the point locations on the map, as it preserves the local similarities of our data.

UMAP was proposed in a recent paper of McInnes et al. (2018). It is also a non-linear and graph-based method similarly to T-SNE, but it is faster. The reason behind this is that it estimates the high dimensional graph instead of measuring every point.

[3]https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
[4]https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
[5]https://umap-learn.readthedocs.io/en/latest/

In order to use these dimensionality reduction methods, we need to obtain the final U and M matrices from the matrix factorisation. Table 4 shows the required time to get the data with each technique. As expected, PCA is the fastest, while T-SNE is the slowest.

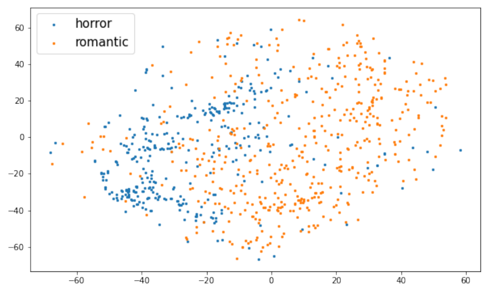
PCA	T-SNE	UMAP
0.438 s	28.010 s	15.489 s

Table 4: The required time for the three dimensionality reduction techniques in seconds.

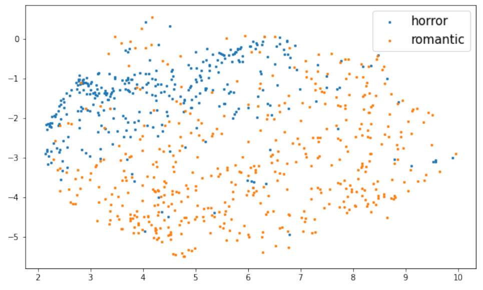
We present an example of these methods in Fig. 5, where we compare movies based on their year of release. We can see that the PCA method is the least successful with clustering as expected, however, we are able to cluster movies with a release year before 1985 and after 1995, although the clustering is not that significant with T-SNE and UMAP techniques.

Another example with the movie data is shown in Fig. 9 in the Appendix. In this plot, we present the gender of the users. We can also see that the PCA method performs worse compared to T-SNE and UMAP. With these methods, we cannot distinguish males and females in our data.

From now on, we are going to present our results based on the T-SNE and UMAP method, as they are more reliable, however, many other plots are presented in *data_visualisation.ipynb*. Firstly, we try to find clusterings in two different occupations. For instance, in Fig. 10 and 11 in the Appendix we compare artists and scientists, programmers and writers, unemployed and lawyers, and students and retired. However, we are unable to see clear distinguishable clusters. On the other hand, we do find some clusterings when we compare horror and romantic movies (see Fig. 6)



(a) T-SNE



(b) UMAP

Figure 6: Clustering of romantic (orange) vs horror movies (blue) with the T-SNE method (left panel and UMAP method (right panel).

4 Discussion

4.1 Comparison of the approaches

Model	RMSE Test	MAE Test	Execution Time (s)	Memory (MB)
Global Average	1.117	0.934	15.330	375.5
Movie Average	0.980	0.782	88.707	389.4
User Average	1.035	0.829	55.218	389.4
Lin Reg with intercept	0.924	0.733	178.778	416.6
Lin Reg without intercept	0.953	0.763	172.274	447.5
UV Decomposition	0.871	0.828	822.069	1948.0
Matrix factorisation	0.885	0.689	1190.862	2054.6

Table 5: The comparison of all the different approaches. For each model we have both the average RMSE and the average MAE for the testings datasets. The running time of each algorithm is also included in seconds, as well as the memory usage.

The different approaches that we used for this project are summarized in Table 5[6]. When comparing

[6]Note that we ran our script multiple times, therefore, we only include our latest runs here.

the results, we can see that the naive approaches perform well for their simplicity as they require much less time and memory to perform but are able to perform at the same level as the more complicated ones. Only the “matrix factorisation” was able to outperform the “linear regression with intercept” on both the RMSE and MAE of the testing set, however, at a huge cost of execution time and memory usage. The “UV Decomposition” outperformed the linear regression only at the RMSE, while also being less time efficient and more memory demanding.

In Figure 7 we can see how the different models performed as well as visualize the increase in computational time. The models are from worst performance to best, based on the RMSE of the testing set. The “UV Decomposition” is the one that performs best, however it requires much more time to run than the “Linear Regression with intercept” model. The “Matrix factorisation” has bigger RMSE and requires more time than the “UV Decomposition”, it has however lower MAE.

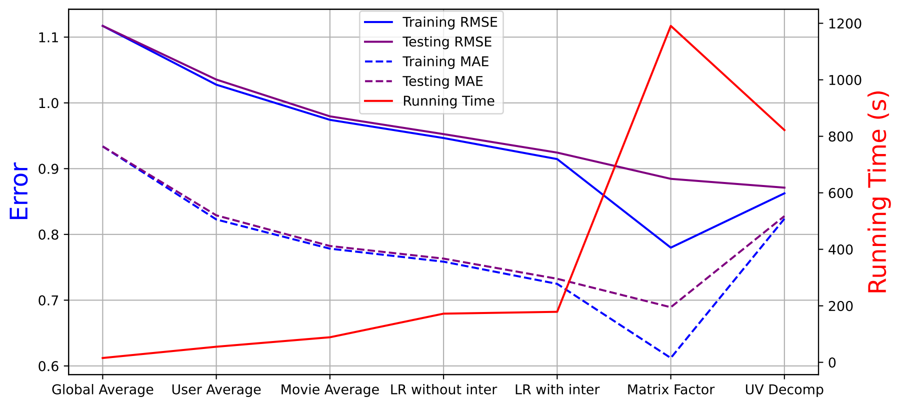


Figure 7: The RMSE (solid lines) and MAE (dotted lines) for both the training and the testing for all the models from worst to best compared with the running time in seconds.

We can see similar results in Figure 2 where we compare the performance of our models with the memory usage. The memory required for the “UV Decomposition” is less than the “Matrix factorisation”. They both require much more memory than the Naive Models for a small decrease in the RMSE.

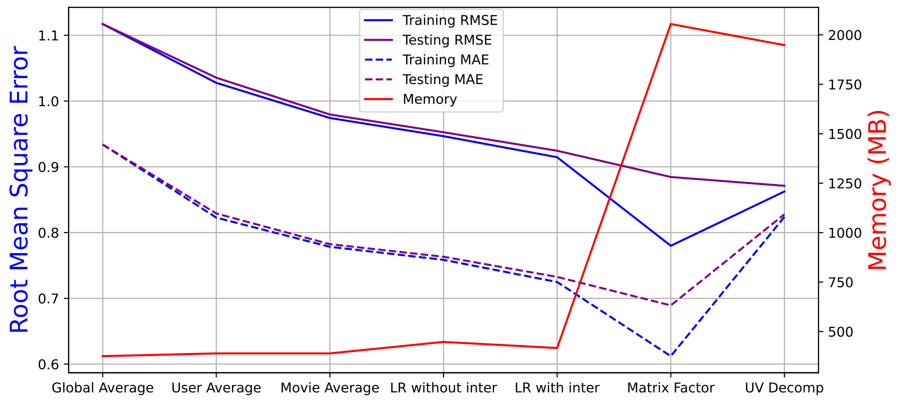


Figure 8: The RMSE (solid lines) and MAE (dotted lines) for both the training and the testing for all the models from worst to best compared with the memory usage in MB.

References

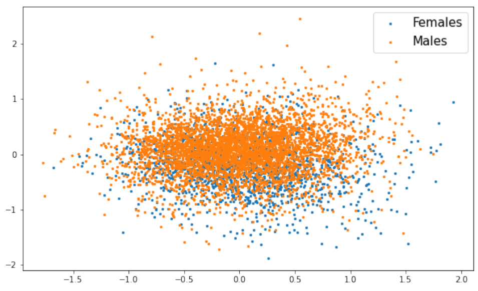
van der Maaten, L. J. P., & Hinton, G. E. (2008). Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research, 9(nov), 2579-2605

McInnes et al., (2018). UMAP: Uniform Manifold Approximation and Projection. Journal of Open Source Software, 3(29), 861

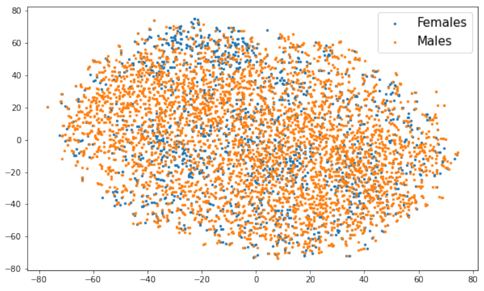
Pearson, K. 1901. “On Lines and Planes of Closest Fit to Systems of Points in Space.” The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 2: 559–572.

Gábor Takács, István Pilászy, Bottyán Németh, Domonkos Tikk
On the Gravity Recommendation Systems
Conference: Proceedings of KDD Cup Workshop at the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining

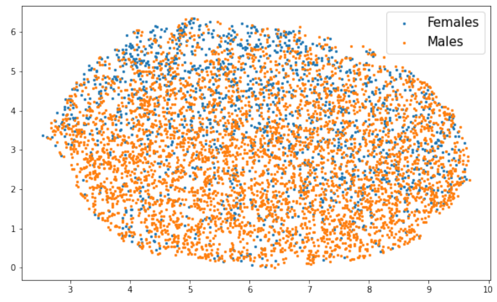
A Data visualisation Plots



(a) PCA

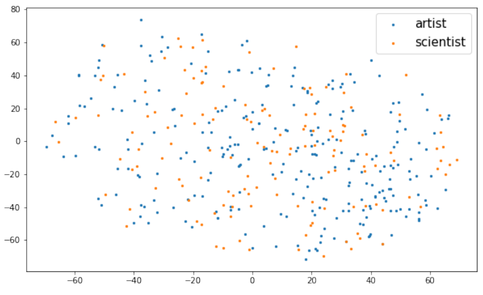


(b) T-SNE

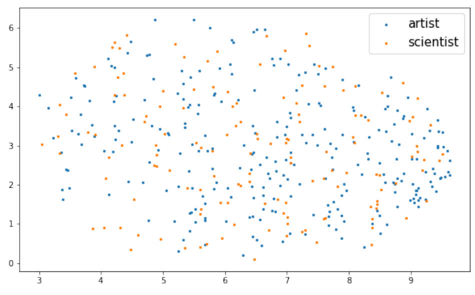


(c) UMAP

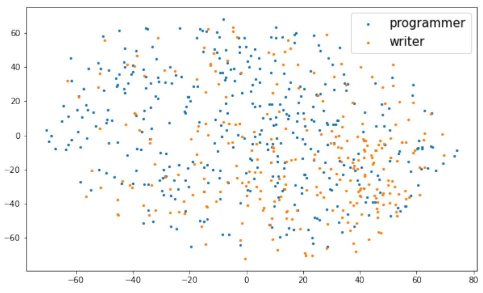
Figure 9: Clustering of females (blue dots) vs males (orange dots) with (a) PCA, (b) T-SNE and (c) UMAP.



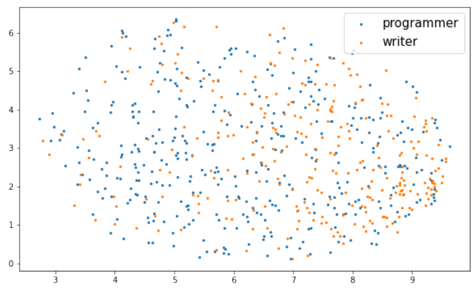
(a) T-SNE



(b) UMAP

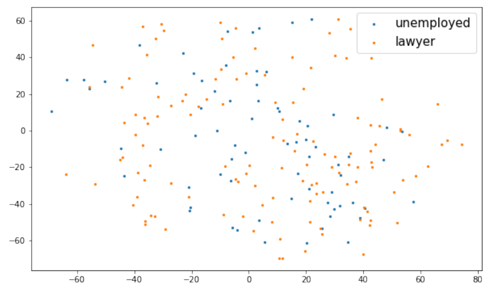


(c) T-SNE

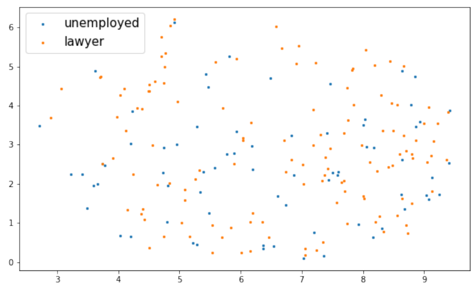


(d) UMAP

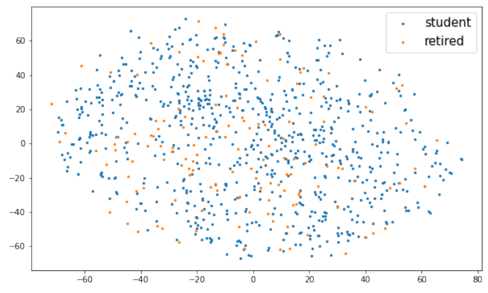
Figure 10: Clusterings of various occupations with the T-SNE (left panel) and UMAP (right panel) algorithms. The colours indicate different occupations as labeled.



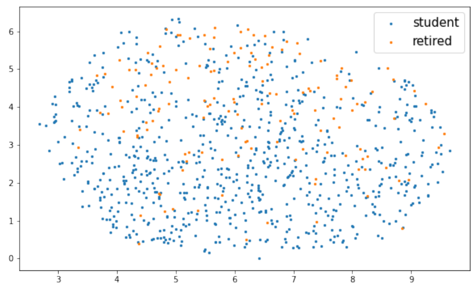
(a) T-SNE



(b) UMAP



(c) T-SNE



(d) UMAP

Figure 11: Clusterings of various occupations with the T-SNE (left panel) and UMAP (right panel) algorithms. The colours indicate different occupations as labeled.