Applied Machine Learning at Scale: build a recommender system from scratch

Onintsoa Anjara Rakotondranisa [1]

Abstract

Recommender systems rely on data that connect users and items. MovieLens data are used to create our model. We use the matrix factorization method which ultimately aims to build a model using alternating least squares. We experimented with different hyperparameters in order to find the appropriate values for our model to perform reasonably well. We tracked the progress of the model by looking at the negative log likelihood and RMSE, generally showing the expected results, but our model seems to be overfitting. We obtain a good recommendation for some movies, which means that our model performs well in some cases. However, some predictions are not logical. The model could still be improved for better performance.

1. Introduction

A recommender system is a tool allowing a user to interact in a personalized context with large and complex information. There are two basic principles. It is personalised for every user, and it is also intended to help the user select specific elements among large discrete options (Burke et al., 2011).

The most commonly used recommender systems are based on numeric data informing about the ratings that one user gives to an item (Al-Ghuribi & Mohd Noah, 2021). Item is the term that is used to define the object that the system recommends to the users. It could be movies, games, video, news, songs, books, CDs, PCs, travel, jobs... Personalised recommendations are shown in a form of a list of items ranked according to a rating. Based on the information provided to the system, it tries to predict the preferences of a user.

[1]African Institute for Mathematical Sciences (AIMS) South Africa, 6 Melrose Road, Muizenberg 7975, Cape Town, South Africa. Correspondence to: Onintsoa Anjara Rakotondranisa <onintsoa@aims.ac.za>.

AIMS | African Institute for Mathematical Sciences SOUTH AFRICA

The main motivation to build a recommmender system is to increase the number of items sold, sell more diverse items, increase the satisfaction of users, and better understand what the users want (Ricci et al., 2010).

The typical way that a recommender system works includes three phases: modelling, prediction and recommendation. The details are as follows (Al-Ghuribi & Mohd Noah, 2021).

- Modelling refers to the phase where the available data are used to build a rating matrix containing users as records and item as attributes, with the corresponding ratings as the elements of a matrix's cell.
- Prediction, as the name implies, refers to the part where the prediction happens. This is where a rating is predicted for an unseen item for a specific user, depending on the information extracted form the modelling phase.
- Recommendation is the final step where different methods can be used to influence a user's decision regarding the choice of item.

Two primary types of recommender systems can be applied: content filtering and collaborative filtering. On one hand, content filtering is a method using item features to recommend similar items. It is based on past interactions of items with a particular user. On the other hand, collaborative filtering is based on similarity between users for item recommendations[1].

The second strategy (collaborative filtering) is chosen for this work. It examines the correlation between users and the interdependencies among movies to identify new user-item relation. What makes it very interesting is the fact that it can offer some aspects almost hard to obtain with content -based filtering. Collaborative filtering also provides more accurate predictions. Nevertheless, it still encounters a main problem, which is the incorporation of new users and new movies in the system.

Collaborative filtering can be approached in two ways. The first one is the neighbourhood method which focuses on the relationship between items or users. The main idea is that, for one user, the preference regarding an item is assessed by

[1] https://www.ibm.com/topics/collaborative-filtering

referring to other ratings for neighbouring items. Neighbour means having movies having same ratings given by one user. The second method is called latent factor model. This approach considers the ratings by characterizing both items and users on a given number of factors that is derived from the observed patterns in the item ratings (Koren et al., 2009).

This report aims to build a model for a recommender system based on collaborative filtering. To that end, we will first describe the data that are used for this work, followed by a brief description of the methodology to build the model. The main results will then be presented and discussed.

2. Methodology

2.1. Data

Data from the MovieLens website[2] collected by GroupLens Research, are used. There are various available datasets describing 5-star rating and free-tagging activity. The first experiments were conducted with a small dataset containing 100 000 ratings and 3 600 tag applications applied to 9 000 movies by 600 users. But the dataset that we focus on contains 25 000 095 ratings and 1 093 360 tag applications across 62 423 movies. This particular dataset was collected from 162541 users between January 09, 1995 and November 21, 2019.

To be included in this dataset, randomly selected users have to rate at least 20 movies. The MovieLens users are anonymous, and the only information about them is their ids. As for the movies, only those with at least one rating or tag are included in the dataset.

There are three files containing the data: `movies.csv`, `ratings.csv` and `tags.csv`. Those files are written in a comma-separated values format with a header in the first row.

Most of the task were performed on the ratings file, including the creation of a data structure to have an easier access to the data. The movies file is necessary when making prediction. The tags file can be used to explore the data.

In our case, we are mostly interested in the movie ratings. Figure 1 shows the ratings distributions.

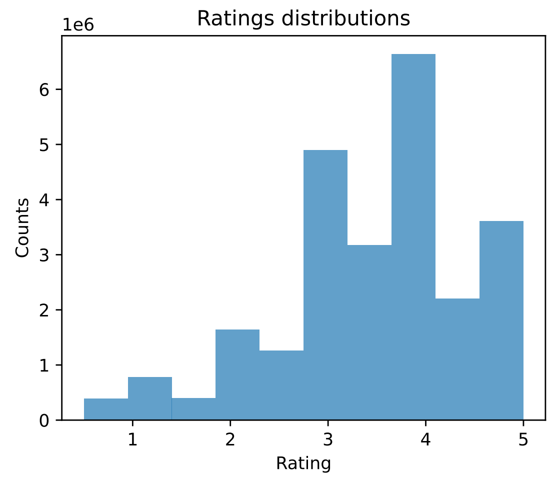

Figure 1. Distribution of the ratings given by users for all the movies they watched.

Data structure: In order to easily explore the data, we create a structure such that we have two lists. The first list of users contains sublists. Each sublist corresponds to one user. Then, every sublist contains tuples of item and rating that are assigned to one user. We proceed in the same way with the items for the second list. Thus, the latter has sublists, where one sublist represents one item. This sublist, which has tuples of user and rating, contains all the users who have watched and rated that item.

In other words, we create a sparse matrix of users and items. As the name indicates, the matrix is sparse because there are a lot of empty elements in the matrix. This is normal since not all users have seen and rated all the movies.

For simplicity, users are mapped to their indexes and movies are also mapped to their indexes, giving four dictionaries that can be accessed when needed.

In neural network terms, we want each user to be represented by a one hot vector and each item to be embedded as a low dimensional vector. The items or users can then be considered as nodes. And every node has k links or degree. The incoming and outcoming degree distributions, for user and movie respectively, are shown in Figure 2.

The distribution of items often follows a power-law distribution. Popular movies, which have been rated by many users, are exponentially greater than less popular ones, which have not been rated by many users (Koenigstein et al., 2012). This pattern is also seen with our data, as seen in Figure 2.

Because we have a power-law distribution, the degree of a randomly chosen node can be significantly different from $<k>$. Consequently, $<k>$ does not serve as an intrinsic scale, meaning that the data is scale-free (Barabási, 2013). If the data were truncated in some way, the power-law that we see would be skewed. We can see this in Figure 3 where only movies with a rating greater than 4 are selected.

Training and test data: To evaluate the performance of the model, the dataset is split into training and test sets. The most common way to achieve this is by doing a random split. The training set is used to fit, train the model, and learn the parameters. As for the test set, it is used to provide

[2] https://grouplens.org/datasets/movielens/

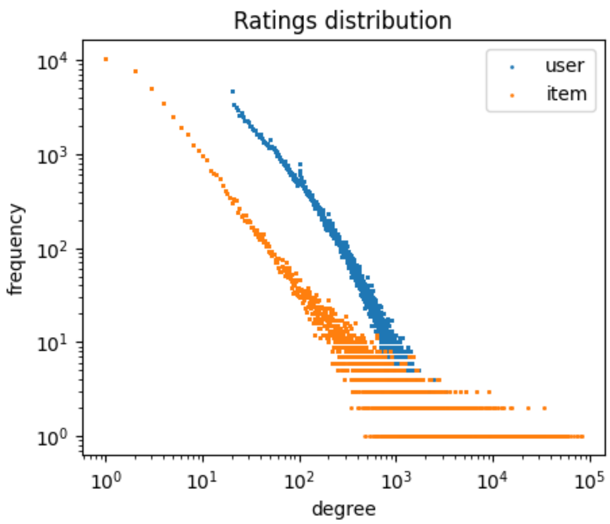

Figure 2. Degree distributions for user and item, as indicated. The x axis shows the degree, which refers to the number of connections each user or item has. The y axis represents the frequency of each degree.

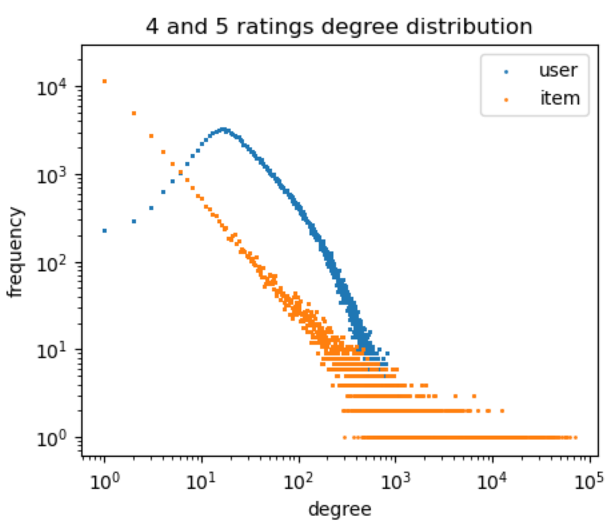

Figure 3. Same as 2 but for items with ratings higher than 4, along with the users who assigned those ratings.

an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The test set is also used to validate the model after training. The data is not just split directly. The number of elements in the training and test set should be the same. The difference is that the test set has more empty lists.

To split the dataset, we proceed as follows. A coin is flipped randomly following a normal distribution. If the resulting value is less than a threshold value, the sample is sent to the test set. Otherwise, the sample is added to the training set.

2.2. Matrix factorization

The matrix factorization is a sophisticated machine learning technique. It is a class of collaborative filtering method which characterizes both items and users by vectors of factors inferred from item rating patterns. In other words, this method decomposes the user-item interaction matrix into the product of two lower dimensionality rectangular matrices (Koren et al., 2009).

The basic model for matrix factorization requires a mapping of both users and items to a joint latent factor space of dimensionality k. The inner product in that two-dimensional space describes the interaction between user and item, which means all users and items are embedded.

Let us assume that each item n is associated with a vector $\mathbf{v}_n$, and each user m is associated with a vector $\mathbf{u}_m$, where n and m are the indexes of items and users respectively. The elements of the item vector $\mathbf{v}_n$ indicate how much factor an item has. The same goes for the user vector $\mathbf{u}_m$ where its elements measure how much interest the user gave to items with high factors. The dot product $\mathbf{u}_m{}^T\mathbf{v}_n$ gives an approximation of the ratings $\mathbf{r}_{mn}$ from a user m to an item n.

The problem can be written in a simple likelihood defined by Equation 1

$$p(r_{mn}|\mathbf{u}_m, \mathbf{v}_n, b_m^{(u)}, b_n^{(i)}) = \mathcal{N}(r_{mn}; \mathbf{u}_m{}^T\mathbf{v}_n + b_m^{(u)} + b_n^{(i)}, \lambda^{-1}) \tag{1}$$

where $b_m^{(u)}$ and $b_n^{(i)}$ refer to the user and item biases respectively. Biases are effects related to either the users or items, causing variations in the ratings, independently of any interactions.

Equation 1 states that the posterior probability of getting a rating r_{mn} given the user vector, item vector, user bias and item bias, is drawn from a Gaussian distribution centred on $\mathbf{u}_m{}^T\mathbf{v}_n + b_m^{(u)} + b_n^{(i)}$, with a standard deviation λ^{-1}.

For simplification, the biases in Equation 1 are removed, at first . So we end up with

$$p(r_{mn}|\mathbf{u}_m, \mathbf{v}_n) = \mathcal{N}(r_{mn}; \mathbf{u}_m{}^T\mathbf{v}_n, \lambda^{-1}) \tag{2}$$

Now, if we write in terms of log likelihood, we have

$$\log p(\mathbf{R}|\mathbf{U}, \mathbf{V}) = -\frac{\lambda}{2}\sum_{m=1}^{M}\sum_{n\in\Omega(m)}(r_{mn} - \mathbf{u}_m{}^T\mathbf{v}_n)^2 + \text{const} \tag{3}$$

This is summed over all the users, but it can be summed over all the items as well.

A regularised log likelihood is then:

$$\begin{aligned}\mathcal{L} &= \log p(\mathbf{R}|\mathbf{U},\mathbf{V}) + \log p(\mathbf{U}) + \log p(\mathbf{V}) \\ &= -\frac{\lambda}{2}\sum_{mn}(r_{mn} - \mathbf{u}_m{}^T\mathbf{v}_n)^2 \\ &\quad - \frac{\tau}{2}\sum_m \mathbf{u}_m{}^T\mathbf{u}_m - \frac{\tau}{2}\sum_n \mathbf{v}_n{}^T\mathbf{v}_n + \text{const}\end{aligned} \tag{4}$$

Then, an algorithm is performed in order to minimise this likelihood. At the end, we obtain the user vector $\mathbf{u}_m$ defined by Equation 5 (and subsequently $\mathbf{v}_n$).

$$\mathbf{u}_m = \left(\lambda \sum_{n\in\Omega(n)} \mathbf{v}_n\mathbf{v}_n{}^T + \tau\mathbf{I}\right)^{-1}\left(\lambda \sum_{n\in\Omega(n)} r_{mn}\mathbf{v}_n\right) \tag{5}$$

(The biases terms are still missing here.)

Alternating least squares (ALS)

We aim to find the optimal matrices that minimises the least squares error. To that end, an optimization algorithm is applied. Equation 4 has two unknowns, namely the user and item vectors $\mathbf{u}_m$ and $\mathbf{v}_n$. The alternating least square technique allows for a rotation between fixing $\mathbf{u}_m$ and $\mathbf{v}_n$ alternatively. It also ensures the gradient is descending. After fixing one vector, the other one is recomputed by solving a least-squares problem. And then, the other vector is calculated the same way.

The ultimate goal is to build a model that finds the maximum likelihood estimate for user and item biases and vectors ($\mathbf{u}_m, \mathbf{v}_n, b_m^{(u)}, b_n^{(i)}$) with alternating least squares.

Root mean squared error (RMSE)

As a metric to assess the performance of the model, we use the root mean squared error (RMSE) defined by Equation 6. It basically measures the amount of variance explained by the model, looking at the difference between the real values and the predictions. Hence, a low value is better.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2} \tag{6}$$

Initialisation

For the biases and latent vectors to be updated, they first need to be initialised. The biases can be set to zero, at first. As for the latent vectors, they cannot be zero. We opt to initialise them randomly from a normal distribution centred on 0 with a standard deviation related to the number of latent factor space k $\left(\mathcal{N}\left(0, \frac{1}{\sqrt{k}}\right)\right)$.

Hyperparameters

There are five hyperparameters in our model, namely n (epochs), λ, γ, τ, k. For the bias only model, which is basically used as a test at the very beginning, τ and k are not considered. Once the algorithm works, we implement the model with embeddings.

For hyperparameter tuning, we experimented with different values, by looking at the evolution of the loss and the RMSE over all the epochs with the training and test sets.

2.3. Model

2.3.1. With biases only

We first start with a model with only the biases ($b_m^{(u)}$ and $b_n^{(i)}$). That means we do not consider the user and item vectors $\mathbf{u}_m, \mathbf{v}_n$ in Equation 4. In order to get the likelihood that we aim to minimise, we need the biases, which are updated with Equation 7 for the user biases (a simple change of indexes for the item biases).

$$b_m^{(u)} = \frac{\lambda \sum_{n\in\Omega(m)}\left(r_{mn} - b_n^{(i)}\right)}{\lambda|\Omega(m)| + \gamma} \tag{7}$$

The corresponding likelihood is given by

$$\mathcal{L} = -\frac{\lambda}{2}\sum_{mn}\left(r_{mn} - (b_m^{(u)} + b_n^{(i)})\right)^2 - \frac{\gamma}{2}\left(b_m^{(u)2} + b_n^{(i)2}\right) \tag{8}$$

Algorithm 1 is followed to update the user biases with ALS. We proceed in a similar way for the item biases.

Algorithm 1 ALS: bias only update

Input: user_data x, user_item y, hyperparameters, epoch
Initialize $b_m^{(u)}, b_n^{(i)}$
repeat
  for $m = 0$ **to** $m - 1$ **do**
    for (n, r) **in** $x[m]$ **do**
      Sum bias terms
    end for
    Update $b_m^{(u)}$
  end for
until epoch is reached

The RMSE is defined by

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(r_{mn} - (b_m^{(u)} + b_n^{(i)})\right)^2} \tag{9}$$

2.3.2. With latent vectors

In this case, we take into account the latent vectors ($\mathbf{u}_m$ and $\mathbf{v}_n$). Consequently, other terms are added to update the biases.

$$b_m^{(u)} = \frac{\lambda \sum_{n\in\Omega(m)} \left(r_{mn} - (\mathbf{u}_m{}^T \mathbf{v}_n + b_n^{(i)}) \right)}{\lambda|\Omega(m)| + \gamma} \tag{10}$$

And the latent vectors also need updating.

$$\mathbf{u}_m = \left(\lambda \sum_{n\in\Omega(n)} \mathbf{v}_n \mathbf{v}_n{}^T + \tau \mathbf{I} \right)^{-1} \left(\lambda \sum_{n\in\Omega(n)} \mathbf{v}_n (r_{mn} - b_m^{(u)} + b_n^{(i)}) \right) \tag{11}$$

Equations 10 and 11 are only for the user biases and vector, the item biases and vector can be written in a similar way with the corresponding indexes.

The likelihood with the embedding vectors is defined as

$$\begin{aligned} \mathcal{L} = &-\frac{\lambda}{2} \sum_{mn} \left(r_{mn} - (\mathbf{u}_m{}^T \mathbf{v}_n + b_m^{(u)} + b_n^{(i)}) \right)^2 \\ &- \frac{\tau}{2} \Big(\sum_m \mathbf{u}_m{}^T \mathbf{u}_m + \sum_n \mathbf{v}_n{}^T \mathbf{v}_n \Big) \\ &- \frac{\gamma}{2} \Big(b_m^{(u)2} + b_n^{(i)2} \Big) \end{aligned} \tag{12}$$

Algorithm 2 shows the steps to follow in order to update the user biases and the user latent vector. We can proceed in a similar way for the item biases and vector.

Algorithm 2 ALS: latent vector update

Input: user_data x, user_item y, hyperparameters, epoch
Initialize $\mathbf{u}_m, \mathbf{v}_n, b_m^{(u)}, b_n^{(i)}$
repeat
  for $m = 0$ **to** $m - 1$ **do**
    for (n, r) **in** $x[m]$ **do**
      Sum bias terms
    end for
    Update $b_m^{(u)}$
    for (n, r) **in** $x[m]$ **do**
      Sum vector terms
    end for
    Update $\mathbf{u}_m$
  end for
until epoch is reached

The RMSE is given by

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(r_{mn} - (\mathbf{u}_m{}^T \mathbf{v}_n + b_m^{(u)} + b_n^{(i)}) \right)^2} \tag{13}$$

2.4. Predictions

Once the model is trained with the latent vectors, we only need to store the user and item vectors and biases, load one user's vector at a runtime and finally compute predictions over all items.

The details are as follows.

- Create a dummy user that gives a 5-star rating to a single movie, which will be used to update the user vector $\mathbf{u}_m$, the same way as during training.
- Use the $\mathbf{v}_n, b_m^{(u)}, b_n^{(i)}$ obtained from training to update $\mathbf{u}_m$.
- Make the prediction using the inner product between the user and item vectors, and add the bias (only the item bias $\mathbf{v}_n$ because the recommendation score for a user is independent of the user bias $\mathbf{u}_m$).
- Get the movies corresponding to the top scores from the prediction.

To avoid high bias, the personalisation component should be increased. In other words, the contribution of the item bias needs to be downplayed.

Once we have the top ranking, we can assess whether the recommendations make sense.

3. Results and Discussion

After experimenting with different values, the hyperparameters for our model are the following: $\lambda = 0.01, \gamma = 0.0001, \tau = 0.9$. For the epochs n, we started with just $n = 10$ before increasing to $n = 30$, when using all the 25 million data at once. The latent factor k was also tested with different values, as we will see later in this section.

3.1. Model with biases only

Here, we aim to see the general behaviour of the ALS model, considering only the biases. The loss function is shown in Figures 4 and 5, and the RMSE is presented in Figure 6. We have a monotonically decreasing loss, which means that the model is performing well. The difference of the loss values between the training and test sets is also expected, since the loss is summed over all epochs. Regarding RMSE, the test set has lower values than the training set, suggesting that the model does not perform well on the test set. This is one of the reasons why it is important to regularise the error for RMSE. Therefore, the ALS model definitely must include the latent vectors.

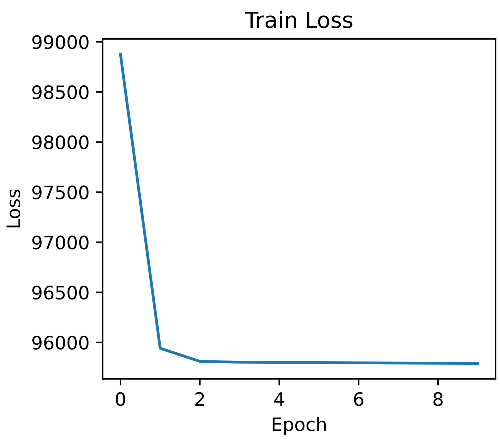

Figure 4. Evolution of the loss function (log likelihood) with the ALS model with biases only, after each of the 10 epochs with the 25 millions dataset split for the training set.

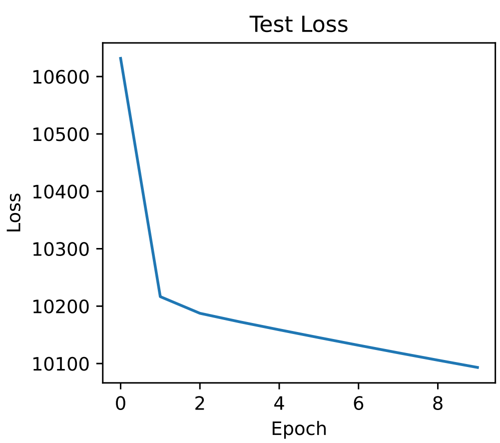

Figure 5. Same as Figure 4 but for the test set.

3.2. Model with latent vectors

We experimented with different hyperparameters. The final results are presented below.

Loss and RMSE evolution

Looking at the evolution of the loss function over all the epochs is important. It helps tracking the training progress. In other words, seeing how the loss changes over epochs can determine if the model is making good progress or if it is stuck or diverging.

The loss evolution with our best hyperparameters ($n = 10, \lambda = 0.01, \gamma = 0.0001, \tau = 0.9, k = 10$) are illustrated in Figures 7 and 8. The expectation with the evolution of the loss function over all epochs is a monotonic decrease. We have this expected trend with both the training and test sets. The values of the loss come from the fact that it is summed over all users (or items). Hence, the high values.

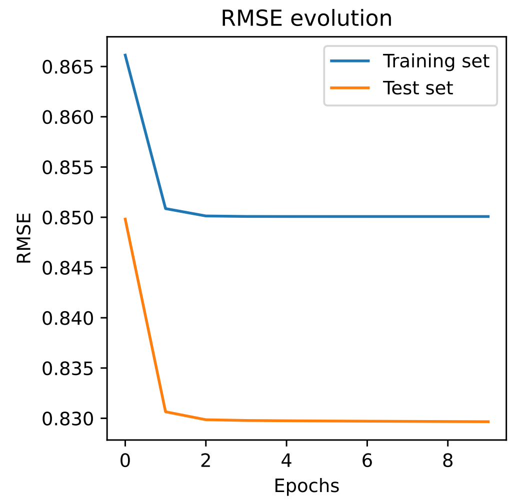

Figure 6. Evolution of the RMSE after each of the 10 epochs with the 25 millions dataset split into training and test sets, as indicated. The model uses biases only.

We notice that the loss converges quite earlier with the test set than with the training set.

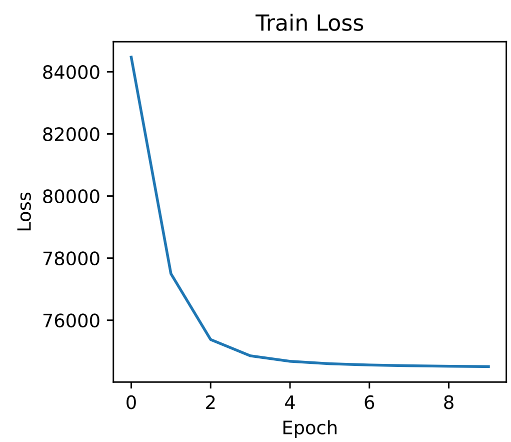

Figure 7. Evolution of the loss function (log likelihood) after each of the 10 epochs with the 25 millions dataset for the training set. The loss is monotonically decreasing.

Figure 9 shows the RMSE over training iterations for the training and test sets. The RMSE for the test set converges very early at ~ 0.83, while for the training set, it only converges at ~ 0.73 after several more iterations. So, both curves decrease and converge at some point. But there is quite a difference between the value at which convergence occurs for the training and test sets. This could indicate that the model has learnt, but did not generalise very well on the test set. Another reason why we have those results is that the test and training set might be different. The most simple way to deal with this is to increase the regularisation term when computing the RMSE. But in our case, $\tau = 0.9$

Figure 8. Same as Figure 7 but for the test set.

is quite high already.

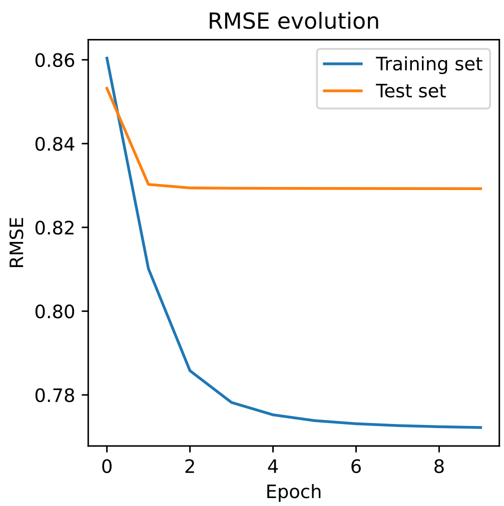

Figure 9. Evolution of the RMSE after each of the 10 epochs with the 25 millions dataset split into training and test sets, as indicated.

Figure 10 compares RMSE over 10 iterations for different values of k. In this case, the data are not split. We clearly see RMSE getting lower with higher values of k.

We also experimented with a split of the data into training and test sets, resulting in an almost stable convergence of the RMSE value for the test set, but decreasing value where the RMSE converges, for the training set. A higher value of k decreases the RMSE in the training set, but it is not generalised on the test set. We can conclude that overfitting is more pronounced with higher k.

The model is always overfitting with the value of the latent factor k on which we trained our model. However, with a lower value of k, some movies might not be well represented in the latent vector space. Hence, we have to find the right balance so that the choice of k captures the embedding, but

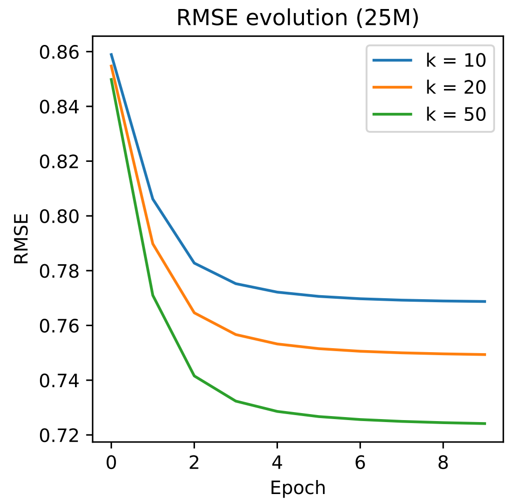

Figure 10. Evolution of the RMSE after each of the 10 epochs with the 25 millions dataset with different latent factor k.

at the same time the overfitting is tolerable. The value of the latent factor that we choose for the prediction is $k = 10$.

3.3. Embedding

Figure 11 shows the 2D embedding of the item trait vectors v_n. For this part, we still used the item vector with the latent factor of dimension $k = 10$ because this is the value that were used for the recommendation. In order to plot the 2D embedding, we reduced the dimensionality using principal component analysis (PCA). A hundred movies (blue points) are scattered in the latent 2D space. Some movie titles are presented as examples. We can see that some points are gathered in the middle, they represent movies with multiple genres.

We have similar movies embedded together, namely children movies like Toy Story (some musical movies are also in the same region). Thriller/crime/horror movies (The Godfather) are placed on a different side. We also see the first two movies of Back to the Future series close together on a different region. These suggest that the embedding is well captured, at least for the few examples. Nevertheless, there is one war movie (Schindler's list) that is placed in the same region as the children and musical movies. This is just one example, but there are probably more of that kind. It could be due to several reasons. As we mentioned earlier, the model is overfitting. Hence, any small correlation or similarities between movies could be exaggerated, resulting in a misplaced movie in the embedding space. For example, some users might have given 5-star ratings to a kids movie and a war movie. Additionaly, the model could consider similarity between genres. In the data, movies can indeed have multiple genres. The dimensionality reduction might also affect the embedding space by distorting distances and losing meaningful separations.

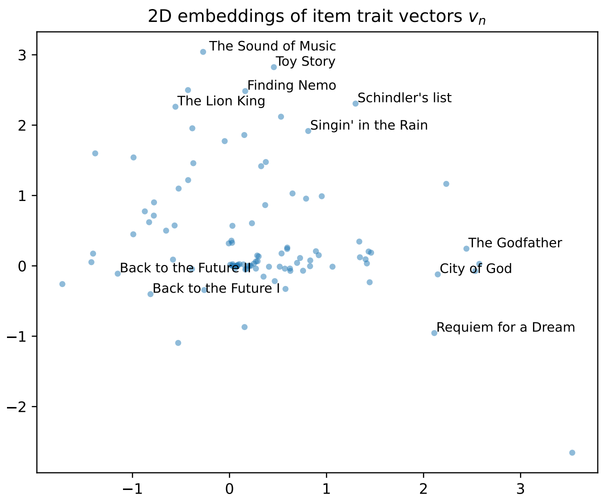

Figure 11. 2D embeddings of the item trait vectors v_n for some items of the data.

3.4. Prediction / Recommendation

To assess the performance of the model, we make some predictions by selecting one movie at a time and see the corresponding movie recommendations, which should be similar movies. Some predictions are listed below.

- Harry Potter and the Sorcerer's Stone
  - Harry Potter and the Chamber of Secrets
  - Harry Potter and the Prisoner of Azkaban
  - Harry Potter and the Goblet of Fire
  - Harry Potter and the Order of Phoenix
  - Harry Potter and the Half-Blood Prince
  - Harry Potter and the Deathly Hallows: Part 1
- Toy Story
  - The Lion King
  - Aladdin
  - A Bug's Life
  - Toy Story 2
  - Shrek
  - Monsters, Inc.
- The Godfather (1972)
  - The Usual Suspects
  - Casablanca
  - Goodfellas
  - The Godfather: Part II
  - The Thorn
  - Anybody's Son Will Do
- Schindler's List (1993)
  - Braveheart
  - The Shawshank Redemption
  - Forrest Gump
  - Dances with Wolves
  - The Godfather
  - Saving Private Ryan
- The Sound of Music (1965)
  - The Lion King
  - Aladdin
  - Beauty and the Beast
  - The Wizard of Oz
  - Titanic
  - Anybody's Son Will Do
- Back to the Future Part II (1989)
  - Forrest Gump
  - Jurassik Park
  - Terminator 2: The Judgment Day
  - Back to the Future
  - Back to the Future Part II
  - Ghostbusters (a.k.a. Ghost Busters)
- Saving Private Ryan
  - Braveheart
  - Apollo 13
  - Forrest Gump
  - Schindler's List
  - Gladiator
  - Anybody's Son Will Do
- La La Land (2016)
  - The Good Mother
  - The Thorn
  - Anybody's So Will Do
  - The Viking
  - Evening's Civil Twilight in Empire of Tin
  - Amori che sanno stare al mondo

With some popular movies like Harry Potter, we have really good recommendations. The model recommends all Harry Potter movies. For Toy Story, we do not obtain the other Toy Story movies as the top recommendations, but we still have Disney or Pixar movies, which is expected.

With other predictions, we get movies of similar genres, actors, directors in the top recommendations (e.g. Schindler's List, The Sound of Music). These results indicate that some users might have watched and rated movies of the same genres, with the same actors or by the same directors. Thus, it is possible to obtain recommendations of movies of different genres given that other aspects (actors or directors) might have been considered. This behaviour is captured by our model leading to our recommendations.

However, for other movies, the recommendations have absolutely nothing to do with the actual movies. This could be due the fact that some movies might have very few ratings (not seen by many users), leading to poorly trained embeddings, giving unreliable predictions.

The cold star problem might also have consequences on the predictions. We have a new (dummy) user, with no previous interaction yet. Therefore, getting a specific recommendation for this user might be complicated. In our case, we rely on collaborative filtering, but coupling this method with content-based filtering could help with this problem.

In addition, the consequence of overfitting is clearly visible in our predictions. For some movies, the top recommendations are exactly the same, which does not make sense. As we already mentionned, the model performs well on some (popular) movies, but that result is not generalised. Here, the model overfits on some movies, leading to repetitive predictions.

Moreover, some outliers are also seen in the predictions. Those movies (e.g. Inside Out (1991), Anybody's Son Will Do) appear in several predictions.

The model could be improved with more hyperparameter tuning. Unfortunately, due to time constraints and limited computer resources, it was challenging to find adequate hyperparameters that would give the best results and predictions. Using a jit compiler like Numba or Jax, we could optimize our code at runtime, and allow for parallelism.

4. Summary

We developed a movie recommendation system for personalised movie suggestions. Our data mainly rely on movie ratings given by users. We created a sparse matrix of users and item that we explored throughout this work. The matrix factorization technique was used for the recommendation in order to get the matrix of lower dimension that captures what the model has learnt about the user and the item during training. We applied an optimization algorithm to find the optimal matrices. Our model does not perform well on every movie due to problems encountered during training or due to the data itself. Nevertheless, we obtain coherent recommendations for popular movies in our dataset. Improvement and optimization can still be made for a better performance. This work has given valuable insights into how a recommendation system can be created using collaborative filtering, matrix factorization or alternating least squares, highlighting how machine learning can be applied to internet-scale systems.

Code and data

Data[3] and code[4] are both publicly available.

References

Al-Ghuribi, S. and Mohd Noah, S. A. A comprehensive overview of recommender system and sentiment analysis, 09 2021.

Barabási, A.-L. Network science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1987):20120375, 2013.

Burke, R., Felfernig, A., and Göker, M. H. Recommender systems: An overview. *AI Magazine*, 32(3):13–18, Jun. 2011. doi: 10.1609/aimag.v32i3.2361. URL `https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2361`.

Koenigstein, N., Nice, N., Paquet, U., and Schleyen, N. The xbox recommender system. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pp. 281–284, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312707. doi: 10.1145/2365952.2366015. URL `https://doi.org/10.1145/2365952.2366015`.

Koren, Y., Bell, R. M., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer*, 42, 2009. URL `https://api.semanticscholar.org/CorpusID:58370896`.

Ricci, F., Rokach, L., and Shapira, B. *Recommender Systems Handbook*, volume 1-35, pp. 1–35. 10 2010. ISBN 978-0-387-85819-7. doi: 10.1007/978-0-387-85820-3_1.

[3] `https://grouplens.org/datasets/movielens/`

[4] `https://github.com/onintsoaarkt/applied-ml-at-scale-report-onintsoa.git`