UNIVERSITÉ
LIBRE
DE BRUXELLES

Currents trends in Artificial Intelligence
-
Movie Recommendation System with GCNs

Oussama Mifdal, Adrien Zabban

June 2023

Contents

1 Introduction

A recommendation system is an artificial intelligence or AI algorithm, which suggests or recommends additional products to consumers. Recommender systems are highly useful as they help users discover products and services they might otherwise have not found on their own.

Recommender systems are trained to understand the preferences, previous decisions, and characteristics of people and products using data gathered about their interactions. These include impressions, clicks, likes, and purchases. Because of their capability to predict consumer interests and desires on a highly personalized level, recommender systems are a favorite with content and product providers. They can drive consumers to just about any product or service that interests them, from books to videos to health classes to clothing.

In this project, we explore the utilization of Graph Convolutional Networks (GCNs) in a movie recommendation system. We aim to leverage the power of graph-based modeling to improve the accuracy, diversity, and serendipity of movie recommendations. By considering the graph structure and employing message-passing mechanisms, we can capture the collaborative filtering signals, item content features, and user similarities, enabling us to generate personalized and context-aware movie recommendations.

1.1 Advantages of Graph-based Approaches in Recommendation Systems

Graphs are all around us, real world objects are often defined in terms of their connections to other things. A set of objects, and the connections between them, are naturally expressed as a graph. Researchers have developed neural networks that operate on graph data for over a decade. Recent developments have increased their capabilities and expressive power. We are starting to see practical applications in areas such as antibacterial discovery, physics simulations, fake news detection, traffic prediction and recommendation systems.

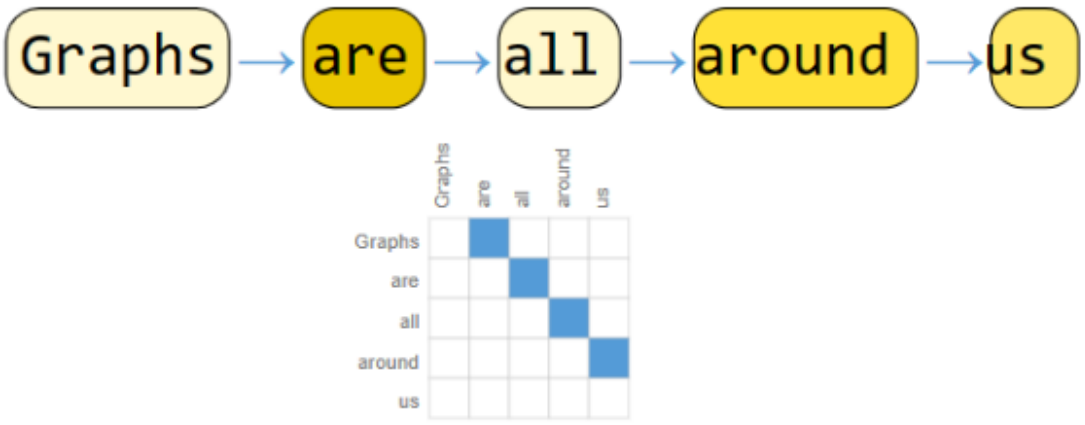

Figure 1: Graph of a text [2]

1.2 Advantages of GCN in Recommendation Systems

The basic intuition is the following: the available data might be better represented in a graph. GNNs can leverage both content information as well as graph structure, whereas typically, traditional models can leverage only one of the two.

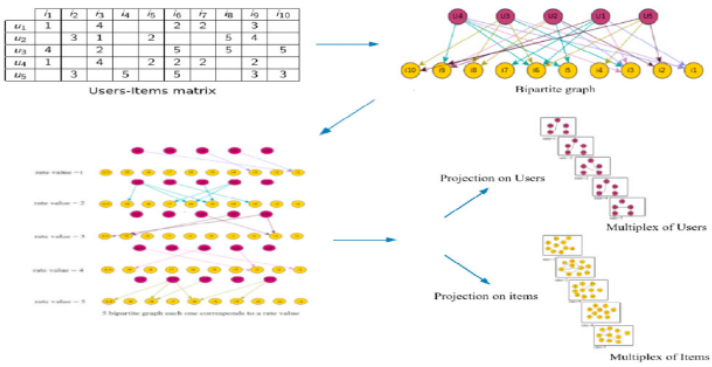

Figure 2: The demonstration of graph learning based recommender systems [5]

1.3 Challenges and Considerations in the Use of GCNs

1. **Graph Construction:** The construction of the graph is crucial in determining the scope and type of information to be propagated. The original bipartite graph consists of user/item nodes and their interactions. There is a choice between applying Graph Neural Networks directly on the heterogeneous bipartite graph or constructing a homogeneous graph based on two-hop neighbors. Computational efficiency is a consideration, and sampling representative neighbors can be used to reduce the computational burden of operating on the full graph.

2. **Neighbor Aggregation:** Aggregating information from neighbor nodes is an important step in graph propagation. Various approaches can be considered, such as differentiating the importance of neighbors, modeling the affinity between the central node and its neighbors, or capturing the interactions among neighbors. This step determines how information from neighbors is combined and utilized.

3. **Information Update:** Integrating the representation of the central node with the aggregated representation of its neighbors is crucial for capturing the overall information. This step involves updating the central node's representation based on the information gathered from its neighbors. The integration process determines how the central node incorporates the collective knowledge of its neighbors.

4. **Final Node Representation:** To predict a user's preference for items, an overall representation of the user/item is required. There is a choice between using the node representation in the last layer of the GNN or combining the representations from all layers. This decision determines the final representation used for making predictions regarding user preferences for items.

2 Data

For this project, we used a dataset we found on the kaggle website (see [6]). You can find the data here. The dataset is composed of 2 csv files. The first contains a list of films associated with an id number. The second gives us access to a list of anonymous users who have watched films and rated them with a score between 1 and 5. 5 being the best score. We also have access to the date on which they watched the film. This information is not included in the project. In all, there are 1682 films, 944 users and 100003 film ratings. Initially, we split this data into three parts to create a training, validation and test database. We cut the data as follows: we took the first 60% of users to put them in the training database, the next 20% for the validation database and the last 20% for the test.

3 Model

In this project we wanted to use convolution graphs to make the best use of our data structure. The aim is to have as input a rating matrix, $R \in \mathcal{M}_{n,m}$ with n the numbers of users and m the numbers of items. In this matrix for which the element i, j contains the rating of user i for film j. This element is zero if the user has not seen the film. This element is zero if the user has not seen the film. The idea is therefore to predict the 0 elements in order to estimate whether the user will like the film or not, and therefore to suggest it or not.

3.1 Model 1

We therefore created an initial model based on [1]. The main idea is to pass the data through layers of graph convolutions and then through a simple layer of neurons to find features of users and films. Then we can have a matrix $U \in \mathcal{M}_{n,k}$ representing users with k the number of features, and a matrix $I \in \mathcal{M}_{m,k}$ representing items (in this case films). We then construct the predicted rating matrix $\hat{R}$ using the following formula: $\hat{R} = U * I^T$. Then to ensure that each element of the predicted rating matrix is between 1 and 5, we tried applying several differential functions (we will call this function 'end function'):

- **a sigmoid:** we apply the function $f : x \mapsto 4\text{sigmoid}(x) + 1$
- **a hyperbolic tangent:** we apply $f : x \mapsto 2\tanh(x) + 3$
- **a clamp:** we apply $f : x \mapsto \max(\min(x, 1), 5)$

However, we have noticed that often the elements of the predicted rating matrix are 0, and just before we do the rating matrix we have a layer of neurons with ReLu. The problem is that the reread function will make a lot of 0, so when we calculate the rating matrix, we end up with a lot of zeros. To avoid this, we also tested other functions such as the sigmoid or the hyperbolic tangent. We call this function the 'middle function'.

3.2 Model 2

Another idea we have had is to put embeddings in front of the model so that it learns about the films and extracts features from them before even going into the convolution

layers. However, the problem with this is that the data then passes into a GCN and each vertex of the graph must have the same number of features. This is a problem because we can't / it's no use embedding on users because they change between training, validation,... There's no point in learning how to place users in a space before knowing which films they've liked. So we came up with the idea of embedding the ids of the films, giving the same number to all the users and putting it in the embedding of the films (we made sure to put a number for the users who weren't already taken for the films). This allows us to add information about the films before passing them on to the GCN. The rest of the structure of this model is the same as before.

3.3 Model 3

The problem with the 2 previous models is that they provide a score in $\mathbb{R}$ (for example 3.2154). We wanted to create a final model that would provide scores in $\mathbb{N}$. So we find a way to manage to have a matrix of size n, m, each element of which is an array with 5 entries (it is in fact a tensor of shape $n, m, 5$), which represents the probabilities of each note. To do this, we used model 2 once with the $\hat{R}$ matrix, transformed it into a vector of size $n * m$ and passed it through a layer of neurons to obtain a matrix of shape $n * m, 5$. We then put it back into a tensor of shape $n, m, 5$ and applied the softmax function.

4 Loss

As the aim is for our predicted rating matrix to resemble our initial rating matrix as closely as possible (without counting the 0 elements in the initial matrix), we use unsupervised learning.

4.1 Masked MSE Loss

We have then created our differential losses. The first is used for models 1 and 2. It consists of calculating the MSE (Mean Square Error) on the known data of the initial matrix (we don't look at the 0 elements). It is given by the following formula:

$$L(\hat{R}, R) = \sqrt{\frac{1}{N} \sum_{i=1}^{n} \sum_{j=1}^{m} (\hat{R}_{i,j} - R_{i,j})^2}$$

where N is the number of non-zero elements of R

4.2 Masked CrossEntropy Loss

Due to another release, we had to change the loss for model 3. We adapted the previous loss by replacing the MSE with the Cross Entroy Loss.

5 Training

To do our training and find the best parameters to get the best results, we implemented a random search (see ref [3]). The principle is to have a list of possibilities for each parameter and to run a training on randomly chosen parameters in the range of possibilities. We ran a random search on the following parameters:

- learning rate (LR), which can take values between 0.01 and 0.001
- embedding size (Embed), which can take value between 0 and 32. If embedding size is 0, so we use the first model.
- hidden layers 1 (hl1), which can take value between 4 and 64.
- hidden layers 2 (hl2), which can take value between 8 and 32.
- hidden layers 3 (hl3), which can take value between 1 and 8.
- droupout probability (droupout), which can take value between 0 and 0.15.
- middle function (middle), which can be ReLu, sigmoid, hyperbolic tangent(tanh) or nothing.
- end function (end), witch can be sigmoid, tanh, clamp, or softmax. If it's softmax and the embedding size is not 0, so we use the third model.

The Figure 3 shows the results of the random search. It shows the best parameters. The table shows the best results based on the best validation loss.

Experiement_path	Best_loss_value	Learning_rate	Embedding_di	Hidden_laye	Hidden_lay	Hidden_layer_3	Dropout	Middle_fun	End_functi
logs\random_search\experiment_47	0.06	0.01	32	4	8	4	0.05	None	tanh
logs\random_search\experiment_43	0.06	0.01	8	64	34	2	0	None	tanh
logs\random_search\experiment_17	0.07	0	32	8	34	1	0.1	relu	sigmoid
logs\random_search\experiment_9	0.07	0.01	8	16	16	8	0.05	None	tanh
logs\random_search\experiment_15	0.07	0.01	32	16	16	1	0.15	relu	sigmoid
logs\random_search\experiment_14	0.07	0.01	16	16	8	2	0.05	relu	clamp
logs\random_search\experiment_27	0.07	0	32	34	16	8	0.15	None	sigmoid
logs\random_search\experiment_35	0.07	0	0	16	16	4	0	tanh	sigmoid
logs\random_search\experiment_18	0.07	0.01	0	16	16	1	0.15	tanh	sigmoid
logs\random_search\experiment_30	0.07	0.01	0	16	8	8	0	sigmoid	sigmoid
logs\random_search\experiment_1	0.07	0.01	0	64	34	8	0	relu	sigmoid
logs\random_search\experiment_31	0.07	0.01	0	64	16	8	0.1	relu	clamp
logs\random_search\experiment_48	0.07	0	0	64	16	4	0.1	None	tanh
logs\random_search\experiment_3	0.07	0.01	0	4	8	4	0.1	sigmoid	clamp
logs\random_search\experiment_8	0.07	0.01	32	4	16	2	0.1	relu	None
logs\random_search\experiment_37	0.07	0.01	0	16	16	8	0.15	None	softmax
logs\random_search\experiment_6	0.07	0.01	0	64	34	4	0.05	tanh	softmax
logs\random_search\experiment_19	0.07	0.01	0	4	34	4	0.15	tanh	softmax
logs\random_search\experiment_44	0.08	0	32	8	16	2	0.1	tanh	None

Figure 3: Result of a random search

We can see that the best results come from model 2 and the 8th from model 1. The best result from model 3 is ranked 30th out of 50. This may not seem like much, but you have to bear in mind that you can't compare them like that because model 3 uses a different loss than models 1 and 2. Now we're going to look at the best results for each model, see Table 1.

Model	Val Loss	LR	Embed	hl1	hl2	hl3	Dropout	middle	end
1	0.067	0.001	0	16	16	4	0	tanh	sigmoid
2	0.065	0.005	32	4	8	4	0.05	None	tanh
3	0.084	0.01	16	4	16	4	0.15	None	softmax

Table 1: Parameters and loss of validation of the best-performing experiment for each model

6 Test

These 3 models were then evaluated on the test data. In order to compare the results of model 3 with those of models 1 and 2, we applied the argmax function for the prediction of model 3, and then applied the loss of the first models. Table 2 shows the results of the evaluation.

Model	Masked MSE Loss	Masked CrossEntropy Loss
1	0.1978	-
2	0.0684	-
3	0.0884	0.0897

Table 2: Evaluation of the different model

Table 2 shows that model 2 is the best, followed by model 3. We notice that model 1 has a very high test loss, which is odd given its validation loss (0.067). This may be due to a poor distribution of training, validation and test data.

7 Prediction

To make a prediction, you first have to enter a list of the films a person has seen and write them down. This can be a long and tedious process. To simplify this task, we made two python programs that generate a user interface. The first program displays a window scrolling through the names of the films, with 2 yes/no buttons at the bottom. You then have to say whether or not you've seen these films (see Figure 4). The results are then stored in a csv file. Next, we run the second program, which scrolls through all the films we've seen and asks us to rate them between 1 and 5 (see Figure 5). This automatically saves everything in another csv file, directly in a good format so that the algorithm can make a prediction.

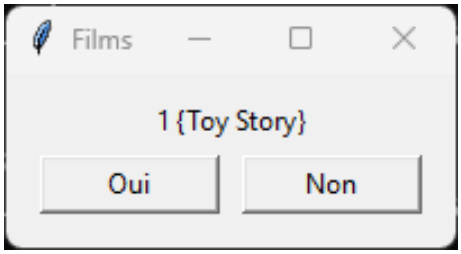

Figure 4: user interface to tell if you've seen the films or not

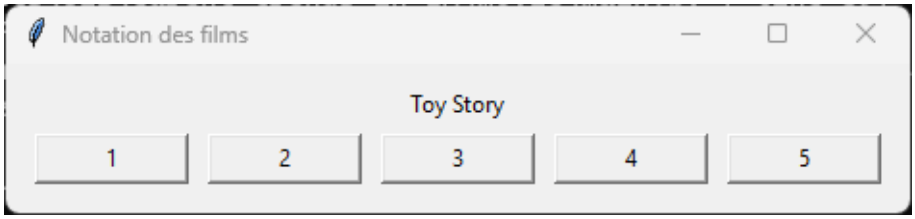

Figure 5: user interface for noting the films you've seen

Then simply run the prediction by choosing the desired model. This will create a new csv file containing all the films and the predicted ratings. The 5 best films according to the algorithm will be displayed in the terminal (films that the user has not already watched). The Figure 6 shows the prediction of the 5 best films to watch for each model.

```
Best film to see for the user: 0
      user_id  rating                    title  year
0           0     5.0                Toy Story  1995
1043        0     5.0            Pallbearer The  1996
1119        0     5.0  Escape to Witch Mountain  1975
1118        0     5.0         Feeling Minnesota  1996
1117        0     5.0                      Safe  1995
```

(a) prediction of model 1

```
Best film to see for the user: 0
      user_id    rating                    title  year
1162        0  3.494605   Welcome To Sarajevo  1997
877         0  3.437329                  Bent  1997
385         0  3.429744  Three Musketeers The  1993
721         0  3.401876          Philadelphia  1993
371         0  3.376557             True Lies  1994
```

(b) prediction of model 2

```
Best film to see for the user: 0
      user_id  rating                    title  year
0           0       3                Toy Story  1995
1288        0       3          Late Bloomers  1996
657         0       3  Bride of Frankenstein  1935
659         0       3             Cape Fear  1962
660         0       3           Cat People  1982
```

(c) prediction of model 3

Figure 6: Prediction of the 5 best films to watch for each model

8 Model defaults

You can see this a little in the predictions for the 5 best films, but it's even more obvious in the predictions for all the films. The values of the scores are very close. Table 3 shows the mean and standard deviation (std) of the predictions for all the films for each model.

Model	**Mean**	**Std**
1	5.0	0
2	3.08	0.08
3	2.43	0.49

Table 3: Evaluation of the different model

We can see that model 1 is catastrophic because it predicts 5 everywhere. Models 2 and 3 are not very good either, as their standard deviations are quite low.

9 Improvements

We could improve our models and therefore our performance by finding other output functions to avoid having too low a standard deviation. Or, more generally, we could incorporate user-user and/or movie-movie connections into our model, we can further enhance its performance and provide even more personalized recommendations.

User-User connection : By considering the social connections between users, such as their friendships or interactions on social media platforms, we can enrich the recommendation process. Users who have similar tastes and preferences often have a higher likelihood of enjoying similar movies. Incorporating user-user connections allows us to capture these relationships and recommend movies based on the preferences of users with similar social connections.

Movie-Movie connection : Movie-movie connections can be established based on various factors such as shared actors, directors, genres, or similar plot themes. When two movies share these attributes, there is a higher chance that users who enjoy one movie will also appreciate the other. By incorporating movie-movie connections, our model can identify and recommend movies that are similar in terms of various aspects, thus expanding the recommendations beyond user-specific preferences.

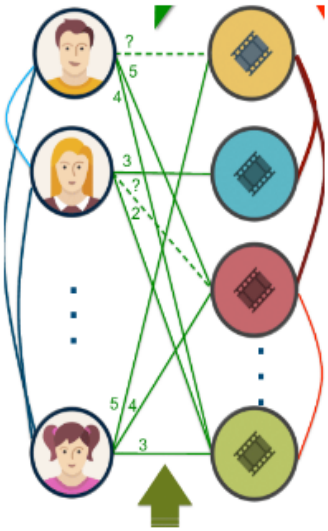

Figure 7: Model with users-users and items-items connection [4]

10 Conclusion

In conclusion, there is a rapid development of graph neural network models in the research field of recommender systems. This paper provides an extensive survey systematically presenting the challenges and the methods. Our recommendation system for movies using Graph Convectional Networks (GCN) offers a promising approach to providing personalized and accurate movie recommendations. By leveraging the power of graph-based

modeling, we can capture the complex relationships between users, movies, and their attributes, leading to improved recommendation performance.

Since we came from different horizons and formations, this project was a human journey, when we have to improve our capacities to work in group on a project, to familiarize ourselves with methodologies we were not used to and to adapt to new environments. It has permitted us to apply for an organization of shared work, using GitHub and a configuration system to launch our training sessions (see our code and README).

11 References

[1] Mariam Zomorodi and Ismail Ghodsollahee and Pawel Plawiak and U. Rajendra Acharya, "RECOMMED: A Comprehensive Pharmaceutical Recommendation System", 2022, https://arxiv.org/abs/2301.00280

[2] Sanchez-Lengeling, Benjamin, et al. « A Gentle Introduction to Graph Neural Networks ». Distill, vol. 6, no 9, septembre 2021, p. e33. distill.pub, https://doi.org/10.23915/distill.00033.

[3] Petro Liashchynskyi and Pavlo Liashchynskyi, "Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS", 2019, https://arxiv.org/abs/1912.06059

[4] HyPER: A Flexible and Extensible Probabilistic Framework for Hybrid Recommender Systems - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Example-recommendation-graph_fig1_283469392 [accessed 11 Jun, 2023]

[5] Graph Learning based Recommender Systems: A Review - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/The-demonstration-of-graph-learning-based-recommender-systems_fig1_350979695 [accessed 11 Jun, 2023]

[6] Movie Recommendation System. https://www.kaggle.com/datasets/dev0914sharma/dataset. Consulté le 11 juin 2023.