AI - Recommendation System

Bedirhan Gergin, Osman Bulut

1 Introduction

Recommendation systems are designed to help recommend an item to a user based on their past interactions with other things. They play a crucial role in many services today and help involve and interact with their user base. Also, they help to connect people with the right product and industries. Lately, online platforms such as Youtube, Spotify, and Netflix are increasing their popularity with their successful implementation of recommendation systems. According to [2] 80% percent of movies watched on Netflix and 60% percent of video clicks on YouTube came from recommendations. Having a recommendation system that is fast and accurate can increase the profit of a company and increase the satisfaction of its users.

Matrix Factorization (MF) is one of the most used Collaborative Filtering techniques. But it has some downsides with the increase in data size. Neural networks are getting popular and used widely in every area. Recently recommendation systems are also benefiting from many features of Neural networks. While it adds non linearity to the system, it is also helps to add a lot of features by neural networks.

In this project, we will use the Collaborative Filtering method on a movie dataset called MovieLens[4], which consists of 5 million movie ratings. First, we will transform explicit data into implicit data, use a Neural Network model to make movie recommendations, and evaluate our result with Hit Ratio @10 Protocol (HR10) method.

2 Related Work

Recommendation systems are one of the biggest application fields in big data and artificial intelligence. As seen in Fig. 1, the recommender system methods can be divided into three main categories. Collaboratives filtering methods benefit from user activity like ratings on items or buying patterns. In contrast, content based filtering methods are based on user activity attributes such as keywords during the search or their profiles. On the other hand, Hybrid filtering techniques use both above methods to have a better model to overcome their limitations and improve performance [9].

Another critical issue in the field is the input types that systems use on the models. Implicit Feedback and Explicit Feedback are two different feedback

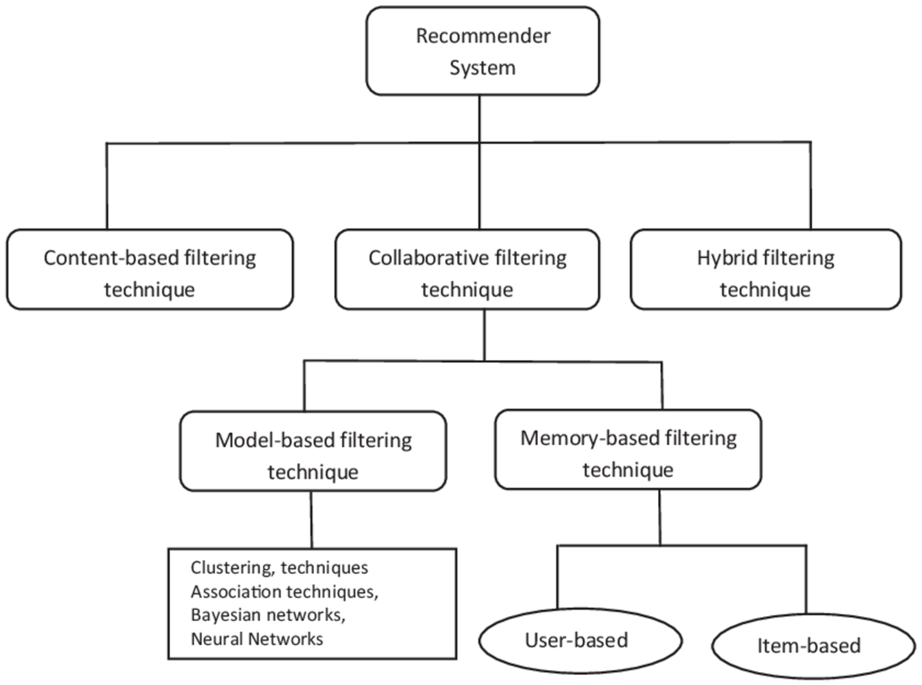
Recommender System
Content-based filtering technique
Collaborative filtering technique
Hybrid filtering technique
Model-based filtering technique
Memory-based filtering technique
Clustering, techniques
Association techniques,
Bayesian networks,
Neural Networks
User-based
Item-based

Figure 1: Categorization of Recommendation Systems

types generally used in recommendation systems. In particular, explicit feedback is collected directly from users' intentional feedback. Reviews, comments, and likes are all explicit feedback[8]. For example, IMDB is a movie rating website based on explicit feedback. It allows people to rate movies on a scale from 1 to 10 and make comments about movies. This is useful for other users to see if a film might interest them. The downside of explicit feedback is that most users don't submit much of it. Think about the number of videos on Youtube you've watched compared to the number of videos on Youtube that you've liked.

Implicit feedback, on the other hand, is not collected directly from users but indirectly from their interactions. For example, even if a user does not rate a video on Netflix, merely watching the video creates an exchange that is used as feedback. This allows us to obtain a considerable amount of interaction data since each click is an interaction. On top of that, if you are a company like YouTube, these interactions are real-time. However, a crucial disadvantage of implicit feedback is being unable to capture negative feedback. Only charging interactions mean only capturing positive feedback. This means we have to employ alternative techniques to have negative samples.

Matrix Factorization (MF) is one of the most used Collaborative Filtering techniques due to its ability to deal with a substantial user-item rating matrix. It pairs each user-item interaction with a real-valued vector of latent features[5]. Latent space is the result of mapping users and products in the same dimension;

the dot product or the cosine of the angle in latent vectors gives us the similarity between two people. Then it also benefits from a second point by the Jaccard coefficient, which helps MF to calculate the similarity between the users without losing generality between them[11]. However, when the rating matrix sparseness increases, its performance deteriorates[10]. To overcome this issue, researchers such as [6] and [7] used side-information of users and items to improve general recommendation performance and weigh the missing data. Another method and the mainly used choice to create recommendation systems is autoencoders such as AutoRec[18]. It uses hidden patterns capable of reconstructing a user's ratings with the inputs of historical ratings is called user-based.

Some recent research, such as [3] has analyzed recommendation systems on implicit feedback (IF) using deep learning models. It can be seen that they mainly used deep neural networks (DNNs) to have additional information like text description of the items, properties of the music, which deals with the behavior of users across multiple domains, and much content about the knowledge base. Recently, Google also published its deep neural network models [1] that they are using for its products. They are usually based on Multi-Layer Perceptron architecture which shows significant improvement in the area.

3 Method

Data

MovieLens[4] is one of the most widely used datasets for evaluating collaborative filtering algorithms. Different versions of the data have been published so far. The data consist of more than 20 million movie ratings from 1995. We used 30 percent of the data with 6 million movie ratings in our project. The data has a user id column, movie id column, rating column, and timestamp column. Movies are rated by users from 1 to 5 in the rating column. And timestamp indicates which movie is rated when. This information will be so useful for us to do a particular way of train-test spit step that we will be explaining in the Data Preprocessing step.

Data Preprocessing

First, we added a new column to the dataset to rank the data based on their view time. If its rank is one, it's the last movie the corresponding user watched. We did that because we wanted to use the latest movies that users watched as a test set and all the previous movies that users watched as a training set. So the model will try to predict the last movies a user watched by learning from the earlier movies that the user watched. This was our method for splitting data as train set and test set, called leave one methodology.

Second, we converted the data from an explicit to an implicit format. The data we used is actually in explicit form. It has user movie ratings that range from 1 to 5. We first converted this detailed data to implicit data, which means we counted all ratings as interactions. Therefore, all ratings will have a value of 1, which indicates interaction. The downside of explicit feedback is it's so rare in the real world compared to implicit feedback. The number of interactions or clicks is much more than the number of reviews, comments, or likes. Therefore, real-world prediction models should be ready to learn from implicit data, which occupy most of the data space. We mainly wanted to work on developing this kind of model. Therefore, even though the raw data is detailed, we assumed we did not know the movie ratings and only known interactions. The question we want to answer is whether we can still develop a model if we have only interactions.

The third important thing we did before training our model was negative sampling. After converting explicit data into implicit data, we obtained data that consisted of interactions only. In other words, we know who did prefer the watch which movie in our data. But we need to find out who did not choose to watch which movie. Therefore, we also need negative samples to train our model. To solve this problem, we needed to do negative sampling. We used different ratios, such as 3 to 1, 4 to 1, and 5 to 1. For example, a 4 to 1 negative sample means we assigned four pieces of zeros to each row from the movies the corresponding user did not watch.

Model

In our project, we used Neural Collaborative Filtering (NCF) [5] Model. Model Framework can be seen in Fig. 2. The model starts with the input layer fed by the user and items, which are binarized sparse vectors with one-hot encoding embeddings. Embedding is a technique that lets you compress high dimension vector to an equivalent but lower dimension vector[8]. This is useful because it can decrease the number of input our model needs while preserving the information it conveys. After the first layer, they are provided with a multi-layer neural architecture called neural collaborative filtering layers. As a result of this layer, we get prediction scores.

NCF is beneficial for us in several ways. NCF allows for computation using implicit feedback rather than explicit feedback. NCF is also highly generic, in this project we will be using it with a database of movies. In order to recommend a movie successfully you need two pieces of information, first you need data about what a given user likes, and you also need information about how similar other movies are to the users preferences. That's a lot of data that is difficult to acquire, particularly information about similarities between movies.

NCF allows us to utilize only information about user interactions with movies to generate those links between other users and movies. This is easy to see on an intuitive level. Consider two users A and B who both watched the same set of 100 movies, in addition A has watched one additional movie. It's very likely that since A and B seem to consume the same kinds of movies that B would also like to watch the one movie A has seen but B has not. This is the principle

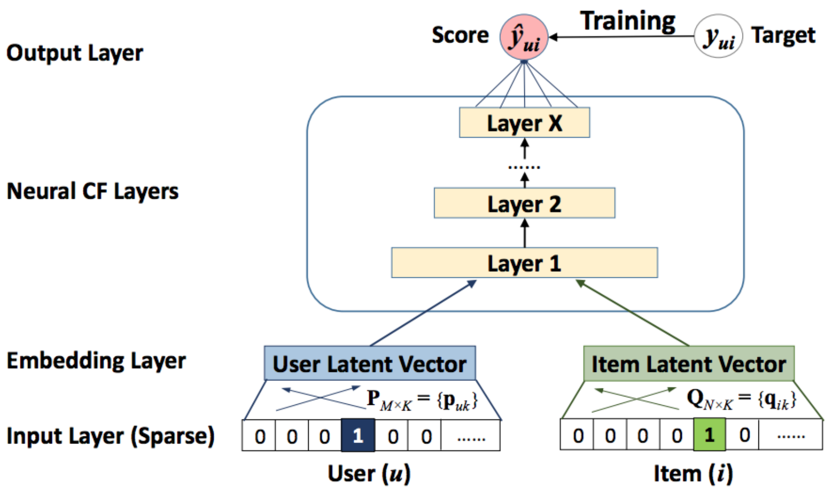

Figure 2: Neural collaborative filtering framework [5]

this system operates on. We don't need to know what kinds of movies they are or the connections between them.

On a broader level, the system will consider hundreds of users and observe the overlaps between them. By training and testing our model to fine tune it we can result in a fast and effective system that can accurately predict movies a user would like to watch.

Evaluation

We used the Hit Ratio @10 Protocol (HR10) to test our model's effectiveness. This protocol involves selecting a random user and a movie they interacted with. Then we chose 99 random movies the user didn't interact with. Combining them, we have a set of 100 random movies. Next, we ask the model for the top 10 movies recommended for the selected user. If the movie we know the user interacted with is in the ten recommended movies, this test result was a hit. Therefore, a high hit to miss ratio is desired. This informs us of the models' accuracy. If all goes as planned, the final output of our model will be a single function that will take in a given user and produce a list of movie recommendations that accurately reflects movies that the user would be interested in watching.

After we preprocess our data set and convert it into an implicit set, we will select the most recent interaction from each user and add it to our testing set. All previous interactions will be used for our training set. This should give us a large collection of both training and testing data. Additionally, after our model is trained and testing begins, we may look into how the weights of the internal nodes affect our results. We may take steps to expedite the process of selecting weights that give a high hit rate.

4 Results

Results were run in the Google Colab Pro with GPU. We did several experiments and analyzed different metrics to determine our system's optimum parameters: Neuron Size, Number of Layers, Activation Function, Maximum Epoch, and Batch Size. We kept the parameters 512 batch size, 5 epochs, 1 layer, and 16 neuron size. We changed the parameters we wanted to observe while keeping all other parameters the same for all the experiments.

First of all, we started with observing the epoch size, as known one Epoch is passing forward and backward of the entire dataset through the neural network once. Results are as seen in Fig. 3. Starting from epoch size 2 with a resulting 0.80, then with the increase in the epoch size, we see an increase in the result until the 5th Epoch. In the 5th Epoch, we see 0.86; then impact is not changing much and continues with almost identical.

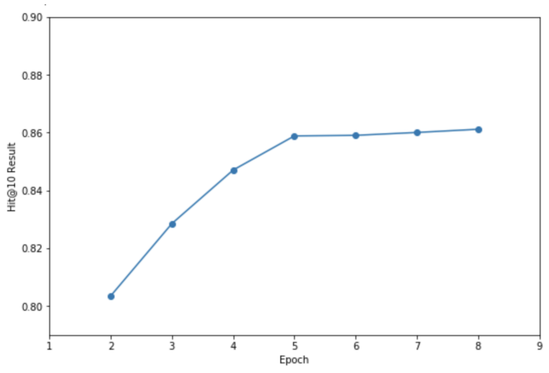

Figure 3: HitRatio@10 Results According to Change in Epoch Size

Secondly, we are looking at the batch size as seen in Fig. 4 with batch sizes 256,512 and 1024. We can see that it starts with 0.85 with batch size 256. Then we can see a little increase with batch size 512, approximately 0.86. Then in the last batch size, it again drops to 0.85. So we can say 512 batch size was optimal with very close results.

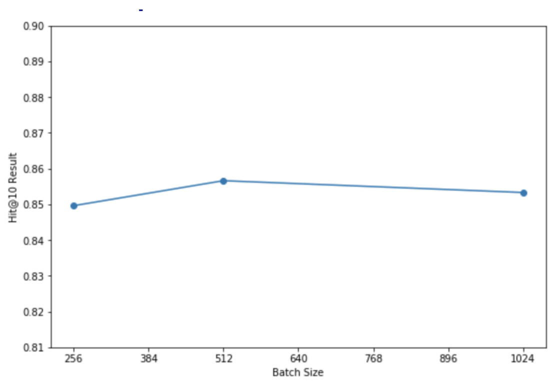

Figure 4: HitRatio@10 Results According to Change in Batch Size

We started with one layer and increased the neuron size to 16,32,64, and 128. As seen in Fig. 5, we get 0.85 with neuron size 16 and then .86 with neuron sizes 32,64, and 128. Although there is a slight increase after neuron size 16, later, there is not much change in the result. From this example, we can say that changing the neuron size didn't affect the system very much.

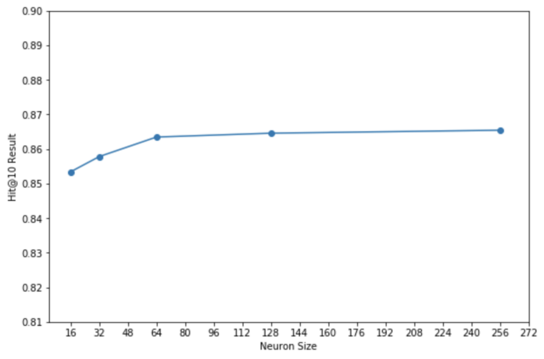

Figure 5: HitRatio@10 Results According to Change in Neuron Size

For the last experiment, we observed the effect of layer size. We started with layer size 1 and increased it to layer size 5. As seen in Fig. 6, results fluctuate around 0.86, and there is no significant change in the system.

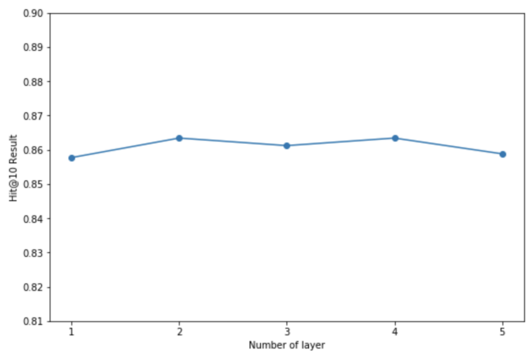

Figure 6: HitRatio@10 Results According to Change in Layer Size

5 Conclusion

Recommendation systems play a crucial role in many services today and help involve and interact with their user base. Also, they help to connect people with the right product and industries. In this project, we aimed to make a recommendation system with the Neural Collaborative Filtering method. We used one of the most popular datasets, MovieLens Dataset, transformed the movie ratings as interactions, and fed them into our system. After creating the design and understanding the whole model, we started to see the effects of different parameters in the system. We analyzed the system in the case of 4 other parameters: Neuron Size, Number of Layers, Activation Function, Maximum Epoch, and Batch Size. We observed that increasing the epoch until a point affected the system positively. Then in the batch size, we found the optimal batch size as 512, while other sizes are also very close to optimal. Increasing the neuron size also affected the system positively until 64 neuron size. On the other hand, we couldn't see a significant effect of layer size in the system. As a result, we got the optimal development of **0.86**.

References

[1] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide amp; deep learning for recommender systems, 2016.

[2] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. The youtube video recommendation system. In *Fourth ACM conference on Recommender systems*, 2010.

[3] Ali Mamdouh Elkahky, Yang Song, and Xiaodong He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. WWW '15, page 278–288, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[4] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), dec 2015.

[5] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering, 2017.

[6] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback, 2017.

[7] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert L. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[8] James Loy. Deep learning based recommender systems. kaggel.com, 2020.

[9] Nishanth Reddy Pinnapareddy. Deep learning based recommendation systems. master's projects. 2018.

[10] Haiyang Zhang, Ivan Ganchev, Nikola S. Nikolov, Zhanlin Ji, and Máirtín O'Droma. Featuremf: An item feature enriched matrix factorization model for item recommendation. *IEEE Access*, 9:65266–65276, 2021.

[11] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. Discrete collaborative filtering. SIGIR '16, page 325–334, New York, NY, USA, 2016. Association for Computing Machinery.