Collaborative Filtering Recommender System

University of Piraeus
Department of Digital Systems
Stamatios Orfanos
stamatisorfanos@gmail.com

Abstract

One of the most useful applications in the field of Pattern Recognition are the Recommender Systems. In particular the collaborative filtering (CF) is the process of filtering or evaluating items through the opinions and interactions of other people. In this paper, we present an detailed analysis, overview and challenges, that we faced during the development of a User-User Collaborative Filtering Recommender System.

1 Introduction

The growth of the internet has made it much more difficult to effectively extract useful information from all the available data on the web. With that problem in mind, it was Dave Goldberg and his colleagues at Xerox PARC that firstly came up with this idea of CF recommender system. Of course in the next years, those systems and especially the CF recommender systems were the centre of research in the field of Information Retrieval.

In the paper Social Information Filtering: Algorithms for Automating "Word of Mouth" (U. Shardanand and P. Mayes), social information filtering exploits similarities between the tastes of different users to recommend or advise against items. It relies on the fact that people's tastes are not randomly distributed, but there are general trends and patterns within the taste of a person and as well as between groups of people.

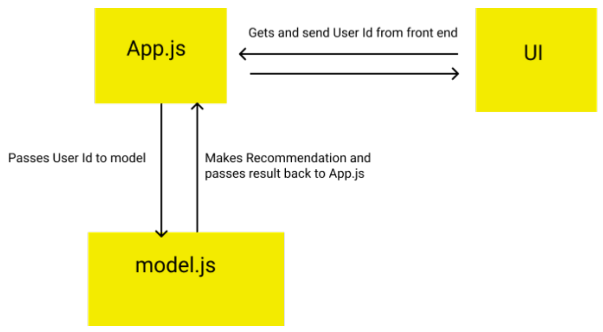
A key advantage of the CF recommender system is that it does not rely on the machine analyzable content, and therefore is capable of accurately making recommendations for complex objects like books, in our case that have multiple characteristics like the genre, the author, the plot, the publishers and other vital data.

We are going to develop a newer implementation of the CF recommender system, that takes advantage of the modern algorithms such as the Neural Network and the embedded layers of data.

2 Methodology

A rough outline of the methodology used to develop the CF recommender system includes the creation of an embedding layer as the input of the model. Furthermore the model consists of two hidden layers before the output layer, in which we get the results of the training. While training we have a validation split of 10%, which is enough given the size of the data set that we are using in this paper.

After the completion of the training of the model, we are going to save the it and then transform it, from a Python model to a JavaScript model. The reason behind the transformation lies in the general architecture of our application, since we use NodeJs, JavaScript and Handlebars.


App.js
Gets and send User Id from front end
UI
Passes User Id to model
Makes Recommendation and passes result back to App.js
model.js

1: Application Architecture

Disclaimer: The application is not production ready, but it is closer to a proof of concept for the usability of the model.

2.1 Data

Initially the data set we are going to be using in this paper comes from the website Kaggle. More specifically we used the goodbooks-10k data set (found https://www.kaggle.com/zygmunt/goodbooks-10k). This data set was quite useful, since it included both information for the training of the model and the development of the application.

The characteristics of the data we are going to be using in the training of the CF recommender model are the identity of each user (userId) and the identity of each book (bookId), from the ratings.csv file. The ratings.csv file has the following data:

Table 1: Ratings.csv

bookId	userId	rating
1	1	4
1	2	3
...	...	...
10000	50.000	2

In the same way we are going to use a subset of characteristics from the books.csv file for the application, like the authors, the title, the image URL of each book.

2.2 Embedding

Firstly we have to analyze the reasons behind the use of the embeddings [1] as an input for our model, instead of using a more well-known method like the Matrix Factorization (MF)[2].

The main reason behind this decision is the growing concern of scientist around deep learning and MF as an effective strategy to tackle the problem of CF recommender systems. [3][4] With that knowledge in mind we tried a different approach with the introduction of an Embedding as input for our model.

Embeddings are a way to represent entities using learned vectors, which is the mapping of a discrete number of objects to a sequence of continuous numbers. Given the data set we used, we had enough to understand the relationship between the userId and the bookId better than other models that used deep learning with MF.

In our case we used an embedding size of 256, which is relatively sensible since we have around 10.000 different books and 53.000 different users in our data set. For both our books and users we have a 1D array as an embedding that consists of 10.000 and 53.000 rows.

It is obvious we have a similar embedding table for the bookId's. The next step was to change the dimensions of those arrays using the Flatten() function with the aim of inserting them as the input layer for our CF recommender system.

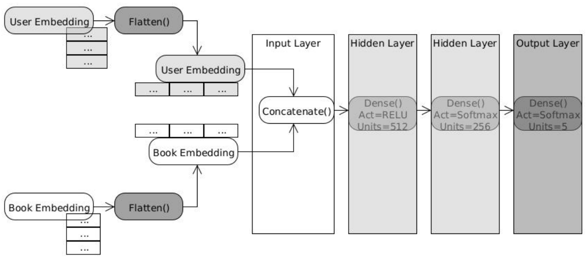
Table 2: Example of Embedding User Table

-0.2	0.24	0.9	...	0.5
0.5	0.34	0.67	...	0.6
-0.5	0.2	0.3	...	0.58
...	...	...	...	...

2.3 Model

Following the completion of the embedding, we use the Concatenate() function adding the two embeddings of the users and the books in an effort to build the input layer.

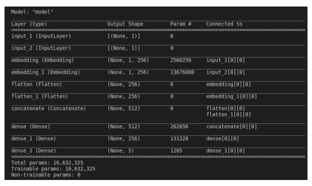
Next was the first hidden layer that used a Dense() architecture, with 512 units and Rectified Linear Unit (relu) as the activation function. The second hidden layer used the same architecture as the previous, but consisted of 256 units and the Softmax activation function.



2.3.1: Model Architecture

Lastly for the output layer we used a Dense() layer with 5 units, since the problem given was a multi-classification one. The goal of the model was to predict the rating between 1-5 from ratings of other users so the Softmax activation function was the most suitable.

The model was formed after experimentation with all the possible different hyper-parameters and a lot of testing with architectures.



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 1, 256)       2560256     input_1[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 256)       13676800    input_2[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 256)          0           embedding[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 256)          0           embedding_1[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 512)          0           flatten[0][0]
                                                                 flatten_1[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 512)          262656      concatenate[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 256)          131328      dense[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 5)            1285        dense_1[0][0]
==================================================================================================
Total params: 16,632,325
Trainable params: 16,632,325
Non-trainable params: 0
```

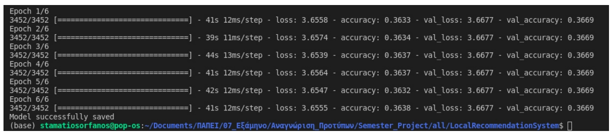
2.3.2: Model Summary

For the training of the CF recommender system, we used the SGD optimizer since the Adam optimizer has some convergence problems that often SGD with momentum can converge better in some cases. The metrics we are tracking are the loss through the Mean Squared Error (MSE) function and the accuracy.

3 Results

The main goal of this paper was to try to solve the main problems of CF recommender systems as a concept, those being the inability to reproduce the results in testing and surpass the 35-40% accuracy.

We trained the model for 6 epochs and with a batch size of 256, since any bigger batch size would create some performance issues, especially with the limited hardware we had in our possession. Unfortunately we estimate that a bigger batch size might have a slightly better accuracy performance, due to the better estimation of the gradient.



```
Epoch 1/6
3452/3452 [==============================] - 41s 12ms/step - loss: 3.6558 - accuracy: 0.3633 - val_loss: 3.6677 - val_accuracy: 0.3669
Epoch 2/6
3452/3452 [==============================] - 39s 11ms/step - loss: 3.6574 - accuracy: 0.3634 - val_loss: 3.6677 - val_accuracy: 0.3669
Epoch 3/6
3452/3452 [==============================] - 44s 13ms/step - loss: 3.6539 - accuracy: 0.3637 - val_loss: 3.6677 - val_accuracy: 0.3669
Epoch 4/6
3452/3452 [==============================] - 41s 12ms/step - loss: 3.6564 - accuracy: 0.3637 - val_loss: 3.6677 - val_accuracy: 0.3669
Epoch 5/6
3452/3452 [==============================] - 42s 12ms/step - loss: 3.6547 - accuracy: 0.3632 - val_loss: 3.6677 - val_accuracy: 0.3669
Epoch 6/6
3452/3452 [==============================] - 41s 12ms/step - loss: 3.6555 - accuracy: 0.3638 - val_loss: 3.6677 - val_accuracy: 0.3669
Model successfully saved
(base) stamatiosorfanos@pop-os:~/Documents/ΠΑΠΕΙ/07_Εξάμηνο/Αναγνώριση_Προτύπων/Semester_Project/all/LocalRecommendationSystem$ 
```

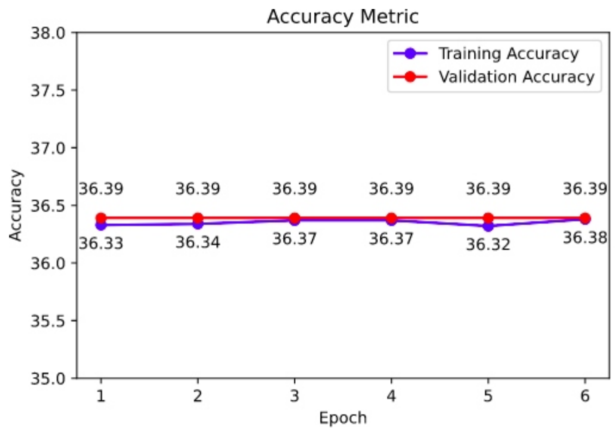
3.1: Training Results

3.1 Accuracy Metric

According to the results of the training we were able to achieve the accuracy goal, since the accuracy fluctuated between the values of 35% and 37%.

Firstly the accuracy of the model increases as the epochs pass, with the exception of the last epoch. After countless experiments we discovered that after 6-8 epochs we had a continuous slight decrease of the accuracy, a sign of over-fitting in our data leading us to the choice of 6 epochs.

Secondly the validation accuracy is constant, as expected since MSE is sensitive to data imbalance. The positive side is the little difference between the training accuracy and the validation accuracy, that ensures no under-fitting or over-fitting are problems of our CF recommender system.



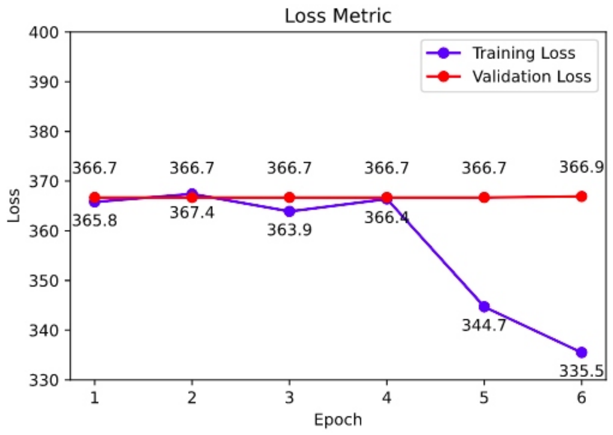
3.1.1: Accuracy Training Results

We have focused on the area of 35%-38% in order to have a clear view of the evolution of the accuracy of the accuracy, since all the values are really close.

3.2 Loss Metric

The loss of the model is pretty high at a first glance, but after some research we figured that it is quite normal given the imbalance data set and the Mean Squared Error (MSE) loss function.

Again there are some positive aspects we can get from our experiment. The positive aspect of our model is the fact that the loss metric is steadily decreasing as the epochs go by.



3.1.2: Loss Training Results

4 Challenges

A collaborative filtering system does not necessarily succeed in automatically matching content to one's preferences. Unless the platform achieves unusually good diversity and independence of opinions, one point of view will always dominate another in a particular community.

As in the personalized recommendation scenario, the introduction of new users or new items can cause the Cold Start problem, as there will be insufficient data on these new entries for the collaborative filtering to work accurately. The collaborative filtering system requires a substantial number of users to rate a new item before that item can be recommended and in our case most of the books had been rated by 1-5 users.

On the technical aspect of this paper, the biggest difficulty we faced was the conversion of the HDF5 or h5 version of the model to the JavaScript format. The main reason for that problem was the lack of documentation on Tensorflow's library and the lack of error messages that could have made the procedure easier to complete by debugging.

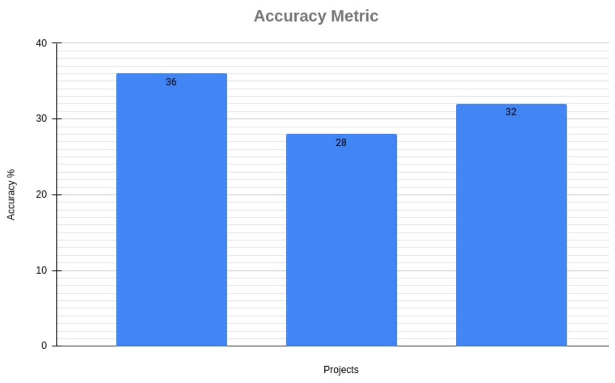
5 Conclusion

Starting this paper our goal was to solve the main problems of the Collaborative-Filtering Recommender Systems. Those problems being the inability to reproduce the results of the training consistently and the extremely low accuracy of those systems.

Unfortunately we were not able to tackle the first major problem of CF recommender systems, since after numerous experiments we were getting various accuracy results, ranging from 20% all the way up to 37%. Although we were able to control the range better than other papers, we did not achieve our ultimate goal of consistent results.

When it comes to the second goal being both the accuracy and the loss metrics we can observe that the samples are quite close, with a slightly better performance for the validation data, which is quite normal given the fact that all the units/nodes of our model are available. Although insufficient, the results pose no surprise since the User-User Collaborative-Filtering Recommender System are not used as a main recommender system, but are mostly used as a part of a Hybrid Recommender System or even a part of an even bigger system for recommendations.

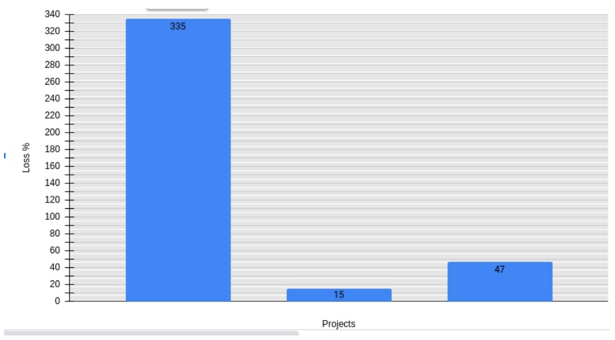
Given other papers that have analyzed the User-User Collaborative-Filtering Recommender Systems, we are able to score good accuracy and in some cases we achieved slightly better accuracy with our embedding approach. After a research for other papers on the same subject we realised that most researchers did not include the accuracy metric for some reasons. The few papers we found that included the accuracy metric, gave us a positive outlook on the results of our work.



5.1: Accuracy Comparison

As we can see our project was the only one of the three to have an accuracy above 35%, which indicates the effectiveness of our model.

When it comes to the loss metric, we did not have as good results as the best performing papers, but we had average results in comparison to all the papers we found.



5.1: Loss Comparison

Table 3: Projects Table

Project
CF Recommender System
[5]
[6]

References

[1] Will Koehrsen Neural Network Embeddings Explained *How deep learning can represent War and Peace as a vector*, October 2018.

[2] Yehuda Koren; Robert Bell; Chris Volinsky *Matrix Factorization Techniques for Recommender Systems.* Volume: 42, Issue: 8, Aug. 2009

[3] Xiangna Liao, Lizi Zhang, Hanwang Nie, Liqiang Hu, Xia Chua, Tat-Seng "Neural Collaborative Filtering". Proceedings of the 26th International Conference on World Wide Web. *International World Wide Web Conferences, ISBN 9781450349130. S2CID 13907106.*,Retrieved 16 October 2019

[4] Rendle, Steffen Krichene, Walid Zhang, Li Anderson, John *"Neural Collaborative Filtering vs. Matrix Factorization Revisited". Fourteenth ACM Conference on Recommender Systems: 240–248. doi:10.1145/3383313.3412488..* 22 September 2020

[5] kth.diva-portal.org/smash/get/diva2:1189454/FULLTI

[6] https://www.mdpi.com/2076-3417/10/7/2441/pdf