Recommender system

Machine Learning project
Team Denner2.0
Automn 2018

Arnaud Duvieusart
Msc in DS, EPFL Lausanne
arnaud.duvieusart@epfl.ch

Henry Declety
Msc in DS, EPFL Lausanne
henry.declety@epfl.ch

***Abstract*—This paper describes our experiments and results for a recommender system. Recommender systems refers to a system that predicts how a set of users is going to rate a set of items. Different approaches have been explored such as collaborative filtering, latent factor analysis and neural network computing.**

I. Introduction

With the abundance and availability of online data, it is crucial to present to users appropriate content. In this context, recommender systems imposed themselves as a famous real-world machine learning problem. Indeed they have a great commercial purpose and change the experience of the users. One example was the lead Netflix took in this field with their notorious challenge in 2009 where they rewarded with 1 000 000$ a team that improved their predictions by 10%

II. Data Description

Our dataset is composed of 1176952 pairs of user and items (u, i). And for each of those, a rating r an integer between 1 and 5. There are $n = 10000$ different items and $m = 1000$ different users. On average, each user has rated 1177 different items and each of them has been rated on average 117 times. Figure 1 shows high sparsity : 1324 items have been rated les than 50 times. Similarly 10 users have rated less than 50 items. This will yield more difficult predictions for pairs containing these users and items. We are given a similar set of pairs of users and items and our task is to predict the rating that would be given.

III. Collaborative fillterinG

Our fist idea was to implement a user-user recommender system. For this, we have to compute the average rating of each user b_u and each item b_i and for each pair of user, item (u, i) we have to compute the similarity between user u all the users who have rated items i over commonly rated items. The similarity between two users $u1, u2$ is given as

$$sim(u1, u2) = \frac{\sum_{j \in C} (r_{u1,j} - b_{u1})(r_{u2,j} - b_{u2})}{\sqrt{\sum_{j \in C} (r_{u1,j} - b_{u1})^2} \sqrt{\sum_{j \in C} (r_{u1,j} - b_{u1})^2}}$$

with C being the set of commonly rated items by $u1$ and $u2$ and $r_{u,i}$ the rating given by user u on item i. Thus the prediction for a pair (u, i) becomes

$$\frac{\sum_{v \in V} (r_{v,i}) sim(u, v)}{\sum_{v \in V} sim(u, v)}$$

with V being the set of users who have given a rating for item i. Similarly, one could do an item-item based recommendation. Unfortunatly our matrix is too sparse to use one of this techniques, therfore we ha to use another approach

IV. Latent factor recommendation

A. Models

There are various existing algorithms that capture the dimensionality of data. We implemented different versions and tested them before picking the optimal for our problem. The results are summarized in Figure 2

B. Singular Value Decomposition

The Singular Value Decomposition (SVD) is a matrix factorization algorithm relying on the eighenvalues. It is the one yielding the best results for our model. The aim is to decompose each item as a composition of latent factors. For items such as movies this factors could be science-fiction/Historical or Action/Romance, and each movie would have a weight for every factor. Similarly, every user would

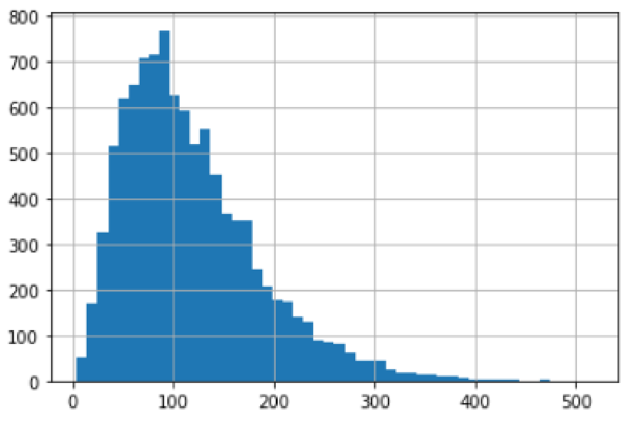


Fig. 1. Repartition of the number of ratings per person

Model	Co-clustering	NMF	KNN	SVD
RMSE	1.006929	1.005606	0.994666	0.984953

Fig. 2. Results for different models

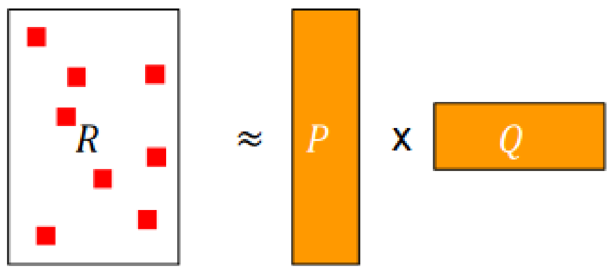


Fig. 3. Scheme of the SVD algorithm

have a weight for every factors representing his sensibility to this kind of movie. Formally, the decomposition is as follow :

$$R = PQ$$

with R being our m x n ranking matrix, P is a m x k matrix respresenting the relationship between users and factors and similarly Q is a k x n matrix representing the relationship between items and factors. In some sense, the SVD algorithm reduces dimensionality of our data by extratcting its latent factors. Figure 3 shows the idea behind SVD.

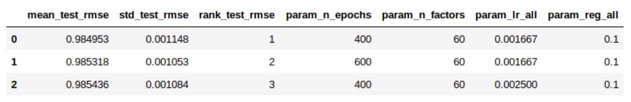
C. Execution

As our matrix has empty entries i.e. we only have 11% of all values, the SVD algorithm is not applicable. We used a variance of this algorithm which does not take into account missing values in the calculation.

To estimate the matrices P and Q, we have to minimize

$$\sum_{(u,i)} (r_{u,i} - p_u q_i)^2 + \lambda(\| p_u \| + \| q_i \|)^2$$

where we added a regularization factor to reduce overfitting. We proceed to a gradient descent on a 5 fold validation for this minimization and we used a grid search to estimate our parameters such as the learning rate, the regression rate, the number of epoch and the number of factors. Figure 4 shows a fraction of our grid search.



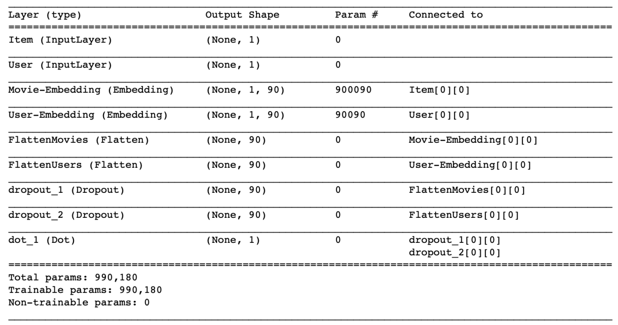
	mean_test_rmse	std_test_rmse	rank_test_rmse	param_n_epochs	param_n_factors	param_lr_all	param_reg_all
0	0.984953	0.001148	1	400	60	0.001667	0.1
1	0.985318	0.001053	2	600	60	0.001667	0.1
2	0.985436	0.001084	3	400	60	0.002500	0.1

Fig. 4. Snapchot of the results of the grid search

V. Neural Network

Even though we had good enough results with SVD, we wanted to give a try to neural networks.

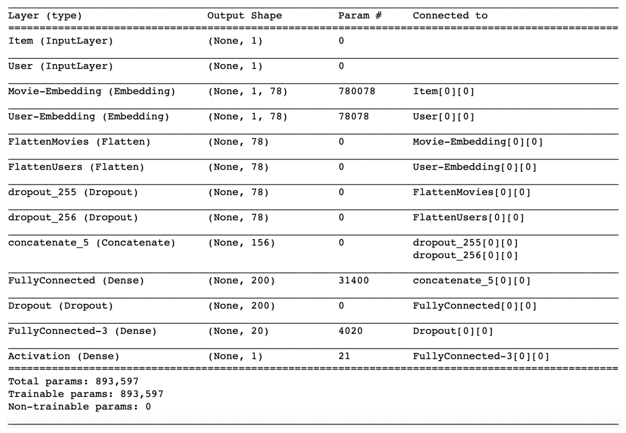
We started with the following implementation : We used embedding layers to transform each movie and each user to a vector and then applied a dot product to them to get the corresponding rating (Figure 5). This model was working but the RAE on a test set of 20% was a bit disapointing (0.91).



```
Layer (type)                     Output Shape         Param #     Connected to
==================================================================================================
Item (InputLayer)                (None, 1)            0
__________________________________________________________________________________________________
User (InputLayer)                (None, 1)            0
__________________________________________________________________________________________________
Movie-Embedding (Embedding)      (None, 1, 90)        900090      Item[0][0]
__________________________________________________________________________________________________
User-Embedding (Embedding)       (None, 1, 90)        90090       User[0][0]
__________________________________________________________________________________________________
FlattenMovies (Flatten)          (None, 90)           0           Movie-Embedding[0][0]
__________________________________________________________________________________________________
FlattenUsers (Flatten)           (None, 90)           0           User-Embedding[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 90)           0           FlattenMovies[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 90)           0           FlattenUsers[0][0]
__________________________________________________________________________________________________
dot_1 (Dot)                      (None, 1)            0           dropout_1[0][0]
                                                                  dropout_2[0][0]
==================================================================================================
Total params: 990,180
Trainable params: 990,180
Non-trainable params: 0
__________________________________________________________________________________________________
```

Fig. 5. First Neural Network Model

While trying to get better results we came up with what we thought would be a better solution (indeed was, we got 0.77 RAE on test our set) since it would allow the user vector and the movie vector to exploit non-linear solutions to produce the rating. This network is described in Figure 6 but the main difference with the first one is that instead of computing dot product we concatenate those vectors and then use all entries as features for the next layers.



```
Layer (type)                     Output Shape         Param #     Connected to
==================================================================================================
Item (InputLayer)                (None, 1)            0
__________________________________________________________________________________________________
User (InputLayer)                (None, 1)            0
__________________________________________________________________________________________________
Movie-Embedding (Embedding)      (None, 1, 78)        780078      Item[0][0]
__________________________________________________________________________________________________
User-Embedding (Embedding)       (None, 1, 78)        78078       User[0][0]
__________________________________________________________________________________________________
FlattenMovies (Flatten)          (None, 78)           0           Movie-Embedding[0][0]
__________________________________________________________________________________________________
FlattenUsers (Flatten)           (None, 78)           0           User-Embedding[0][0]
__________________________________________________________________________________________________
dropout_255 (Dropout)            (None, 78)           0           FlattenMovies[0][0]
__________________________________________________________________________________________________
dropout_256 (Dropout)            (None, 78)           0           FlattenUsers[0][0]
__________________________________________________________________________________________________
concatenate_5 (Concatenate)      (None, 156)          0           dropout_255[0][0]
                                                                  dropout_256[0][0]
__________________________________________________________________________________________________
FullyConnected (Dense)           (None, 200)          31400       concatenate_5[0][0]
__________________________________________________________________________________________________
Dropout (Dropout)                (None, 200)          0           FullyConnected[0][0]
__________________________________________________________________________________________________
FullyConnected-3 (Dense)         (None, 20)           4020        Dropout[0][0]
__________________________________________________________________________________________________
Activation (Dense)               (None, 1)            21          FullyConnected-3[0][0]
==================================================================================================
Total params: 893,597
Trainable params: 893,597
Non-trainable params: 0
__________________________________________________________________________________________________
```

Fig. 6. Second Neural Network Model

With the model getting more complex we face a small issue: overfitting. Even though we were using early stopping, we wanted to delay that overfit to see if it could produce better results. We tried two things mainly to reduce overfitting: regularizers and dropout layers. We went through some grid search to find out what worked best in our specific model. Dropout layers with rate 0.2 ended up working best.

VI. Results

Our best results were obtained with the latent factor method. With a training on the full set and the best parameters determined by our grid search, we reached a final score of 1.025 on crowdAI, giving us a ranking of 40 on this competition.

VII. Conclusions

Recommender systems are a clear yet complex problem. Through our different attemps we got a better insight on the difficulties yielded by such a challenge. It seems like the room for improvement for our latent factor model is quiet small but this project could be perfected with a more complex neural network. However a tradeoff has to be made between precision and computation time.