Machine Learning Recommender System

Emery Sébastien, Canton Diego, Sergi Guido
Machine Learning CS-433, Project 2
Department of Computer Science, EPFL, Switzerland

***Abstract*—Recommender sytems aim to personalize products or services recommendation to the user. One successful class of such systems is called Collaborative Filtering (CF), which establishes connections between users and items based on the previous products ratings. The two main models of CF are latent model factor, which characterizes users and items by vectors of factors, and nearest neighbors which analyzes similarities between users and items. We use both algorithms and neural networks on the provided data set, using different Python libraries. The performance is assessed using RMSE metric.**

I. Introduction

Recommendation systems are used in a wide variety of industries, with e-commerce and streaming being the main ones. The increasing number of players entering the streaming industry (Netflix, Amazon, Disney, ...) makes providing relevant movies recommendations to users an essential aspect to gain market share. An accurate recommendation systems can represent a significant competitive advantage over other players and can help differentiate one's streaming platform. Recommenders are based on two general strategies, Content Filtering (CB) and Collaborative Filtering (CF). CB systems require to identify the factors which describes the products or items and to represent them in terms of those factors. On the contrary, CF systems model interactions between users and items as a product of latent factors based on the previous ratings. Therefore, CB approaches require extensive data collection which is not always possible and CF latent factors technique allow to caption complex patterns that are difficult to profile with CB. Our data set only consisting of 1'176'952 ratings given by 10'000 users for 1'000 items (movies), so we use CF approaches. The primary CF algorithms used are neighborhood methods and latent factor models. Neighborhood methods analyze similarities between users and items, the item based approach evaluates a user's preference for an item based on the ratings of the neighboring items. Latent factors models characterize users and items by a vector of factors inferred from the past ratings, matrix factorization is a well established method to do so where the dot product of a user and an item vector gives the rating of the item i by user u. We built a recommender starting from a general baseline model and we improve it by using both CF algorithms described above using the Surprise, FastAI and spotlight packages. Neural networks are also implemented using the last two ones. The models' performance is assessed using the Root-mean-square-error (RMSE) with K-fold cross validation.

II. Models and Methods

A. Baseline models

We used the following models as baselines:

$$\begin{aligned}\text{Global mean: } \hat{x} &:= \tfrac{1}{|\Omega|}\textstyle\sum_{(u,i)\in\Omega} x_{ui}\\ \text{User mean: } \hat{x}_u &:= \tfrac{1}{|\Omega_{u:}|}\textstyle\sum_{i\in\Omega_{u:}} x_{ui}\\ \text{Item mean: } \hat{x}_i &:= \tfrac{1}{|\Omega_{:i}|}\textstyle\sum_{u\in\Omega_{:i}} x_{ui}\end{aligned}$$

Where Ω is the set of non-zero rating indices *(u,i)* in the training data matrix, $\Omega_{u:}$ is the set of movies rated by the *u*-th user, and $\Omega_{:i}$ is the set of users who have rated the *i*-th item. [1]

B. Surprise package

We decided to use the Python library Surprise [2], which already provides ready-to-use CF prediction algorithms for recommender system that handle explicit ratings. The package provides model selection modules which allow to tune the hyper parameters and cross validate the results. In the following section we describe the mathematical models used and the methods to estimate the parameters.

Baseline

A baseline estimate for an unknown rating r_{ui} is denoted by b_{ui} and accounts for the user and item effects:

$$\hat{r}_{ui} = b_{ui} = \mu + b_u + b_i$$

The parameters b_u and b_i indicate the observed deviations of user u and item i, respectively, from the average [3].
The following regularized cost function is minimized using Stochastic gradient descend (SGD) to estimate the parameters:

$$\min_{b_u,b_i} \sum_{r_{ui}\in\Omega} (r_{ui} - (\mu + b_u + b_i))^2 + \lambda(b_u^2 + b_i^2)$$

Nearest Neighbors inspired

These algorithm provide predictions derived from the nearest-neighbors model. They are memory based models which make predictions for a user/item based on what similar users have rated or what ratings similar items have received. We used *KNNBasic()* and *KNNBaseline()* algorithms to predict ratings.
K-NNBasic(): The prediction item-based $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \frac{\sum\limits_{j\in N_u^k(i)} sim(i,j)\cdot r_{uj}}{\sum\limits_{j\in N_u^k(i)} sim(i,j)}$$

K-NNBaseline(): The prediction item-based $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum\limits_{j \in N_u^k(i)} sim(i,j) \cdot (r_{uj} - b_{uj})}{\sum\limits_{j \in N_u^k(i)} sim(i,j)}$$

where b_{ui} is the baseline rating of user *u* for item *i*, N_u^k, the k neighbors of u and sim(i,j) is a similarity measure between users *i* and *j*. The Surprise library provides multiple choices to compute the similarity (e.g. cosine distance, mean square error). For user-based, the formula is equivalent but the similarity is calculated on a pair of users (u,v) with $v \in N_i^k(u)$. The algorithm first estimate bias terms using ALS (for KNNBaseline) and then compute the similarity matrix to make predictions using above formula.

Matrix factorization based algorithms

These algorithms are based on the famous singular value decomposition (SVD) matrix factorization. The $(I \times U)$ dimensional input (ratings) is mapped into a joint dimensional space K linearly. Each item i and user u are associated with a vector of features q_i and $p_u \in R^k$ respectively. The ratings are now represented by the inner product of that space $\hat{r}_{ui} = q_i^T \cdot p_u$. Such algorithms cannot be applied on typical user-item matrix which are usually sparse. Therefore, the training is done on the set of available ratings : $(i,u) \in \Omega$ which was popularized by Simon Funk during the Netflix challenge. We used the two following algorithm : SVD() and SVDpp() which are provided by the surprise package.

SVD(): The prediction item-based $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = b_{ui} + q_i^T p_u = \mu + b_u + b_i + q_i^T p_u$$

where b_{ui} is the bias involved in rating $\hat{r}_{ui}$ as described in the baseline section above.
The following regularized cost functions is minimized using SGD to estimate the parameters :

$$\min \sum_{r_{ui} \in \Omega} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_u^2 + b_i^2 + \|q_i\|^2 + \|p_u\|^2)$$

SVDpp(): The prediction item-based $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T(p_u + |I_u^{-\frac{1}{1}}| + \sum_{j \in\ I_u} y_j)$$

where I_u is the set of all items rated by user u, implicit feedback, and y_j captures the implicit ratings which means user u rated item j regardless of the rating. A regularized square loss similar to SVD using SGD is used to optimize the parameters.

C. *Fastai package*

In last sections, we applied methods based on nearest-neighbors and matrix factorization. Now, we decided to try neural networks-based models. Despite neural networks make interpretation more difficult, they could help to learn new features from the data. Therefore, we choose the Fastai library [4] which provides ready-to-use methods for fast and accurate training of neural networks suitable for collaborative filtering. The library need PyTorch to run. The methods we used are the following:

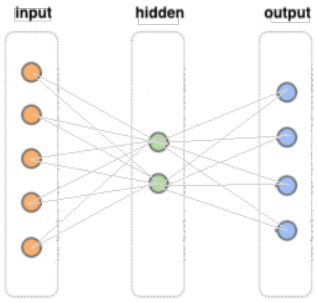

Fig. 1: Representation of EmbeddingDotBias algorithm [5]

EmbeddingDotBias()

This algorithm create a simple model based on matrix factorization. It takes the dot products of the embedding vectors of size K for an user and an item and add the bias terms to predict the rating. Its also apply a regularizer term to avoid overfitting. The predictions are then compared to the real values (test set generated directly by the algorithm) and then, the model is gradually adjusted. What is added in this model compared to SVD earlier, is that after calculating a prediction the model applies an activation function (here, a sigmoid function) to map the result between 1 and 5. It could be compared to a simple neural network with one hidden layer as in figure 1. The algorithm only needs 3 parameters to run, the size of the embedding vectors = K, the range of predictions = [1,5] and the regularization parameter.

EmbeddingNN()

This algorithm creates a more deeper neural network. We can choose the number of hidden layer (not just one as before) and the size of embedding vectors for user and item separately.

More details about how to use these methods are given in the `read.me` and in the `fastai_package.py` files.

D. *Spotlight package*

We now try to implement shallow factorization representations using Spotlight library [6]. Spotlight, like Fastai, is based on PyTorch. The difference is that Spotlight, like Surprise, aims to be a tool for rapid exploration and prototyping of new recommender models. The models available in Spotlight fall under two main categories: sequence models and factorization models. Given the structure of the training data we have (user id, movie id, rating), we used the Explicit Factorization Model.

Explicit Factorization Model

This is an explicit feedback matrix factorization model. It uses a matrix factorization approach [7], with latent vectors used to represent both users and items. The model implementation provides different parameters that allow for tuning. The main ones are *loss*: loss function to use, *embedding_dim*: number of embedding dimensions to use for users and items, *n_iter*:

number of iterations to run, *batch_size*: minibatch size, *l2*: L2 loss penalty, *learning_rate* : initial learning rate.

E. Exploratory Data Analysis and Pre-Processing

The dataset consist of integer ratings between 1 to 5 stars of movies from users. No particular pre-processing was done except the loading of the dataset on the different packages which is a done by the following files: *suprise_package.py,fastai_package.py and data_spotlight.py*.

F. Model selection and Validation

We used RMSE metric and 5-fold cross-validation procedure to compare our models and select the best one, the function cross_validate() from surprise and the file CVspot.py were used to do so. The function GridSearchCV() allow us to tune the hyper parameters by doing a grid search procedure on the range of parameters given as an input and comparing them using CV.

III. Results

Method	Local RMSE	AICROWD
Baseline		
General Mean	1.122	1.127
User Mean	1.033	1.117
Item Mean	1.096	1.070
Surprise		
Baseline	0.999	1.040
SlopeOne	1.000	1.040
KNN Basics	1.028	1.067
KNN Baseline	0.995	1.038
CoClustering	1.003	1.042
SVD	0.986	1.027
SVD++	0.992	1.039
FastAI		
EmbeddingDotBias	0.990	1.030
EmbeddingNN	0.996	1.037
Spotlight		
ExplicitFactModel	0.988	1.025

Table 1: Best local RMSE and AICROWD score for each method

A. Baseline models

The results from baseline models gave us a baseline to start with. The best result we got was from the baseline method based on user mean, thanks to which we achieved a RMSE of 1.033. (Table 1) By using the baseline method available in Surprise library we increased the accuracy and obtained a RMSE of 0.999. (Table 1)

B. Nearest Neighbors based methods

From all the K-NNs algorithms in the surprise library, we got the best RMSE with KNNBaseline() (see table 1). With k=40, the number of neighbors, MSD as similarity measure and using the item-based option.

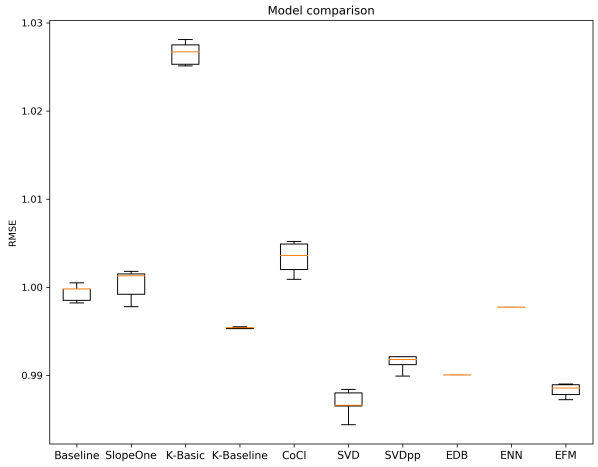

Fig. 2: Model Comparison

C. Matrix factorization based methods

SVD() / SVD++()

By using the SVD() function available in Surprise we managed to obtain a RMSE of 0.986, which is our best result overall. In order to obtain this result we used the following tuning of hyper-parameters: *lr_bu=0.007, lr_bi=0.0008, lr_pu=0.01, lr_qi=0.02, reg_bu=0.1, reg_bi=0.0001, reg_pu=0.8, reg_qi=0.01*. The best result we got by using the SVDpp() function is a RMSE of 0.992. In order to get this result we used the following hyper-parameters tuning: *lr_bu: 0.005, lr_bi: 0.0001, lr_pu: 0.01, lr_qi: 0.01, reg_bu: 0.01, reg_bi: 0.01, reg_pu: 0.5, reg_qi: 0.01 , n_factors: 200.*

EmbeddingDotBias()

The best RMSE we could reach is 0.990 with K=200, the size of embedding vectors and λ=0.05, the regularizer parameter. This result is nearly similar as the results we have for SVD. Since both are based on matrix factorization that was expected. (Table 1)

ExplicitFactorizationModel()

Using the ExplicitFactorizationModel() from spotlight we managed to obtain a RMSE of 0.988 by using the following tuning: *loss='regression', embedding_dim=8, n_iter=200, batch_size=4096, l2=5e-5, learning_rate=2e-3*.For reproducibility we added the optional parameter *random_state=np.random.RandomState(42)* in order to obtain the same predictions with every execution. We observed that increasing the size of embedding_dim leads to overfitting of our model.

D. Other collaborative methods

Surprise library provides other methods such as CoClustering() and SlopeOne(). We used these methods and included the results in Table 1. As can be seen from the table, the results

obtained were similar to Surprise baseline method and given the little room available for tuning of hyper-parameters, we decided to not investigate these methods further.

E. Neural Network based methods

We use EmbeddingNN() from Fastai which yield an RMSE of 0.996, which is a little higher than the one obtained with matrix factorization based method, therefore no further optimization was done. We obtained this result with the following parameters: use_NN = true (goes into neural netork mode), embedding size: user=100, item=100, layers=[32, 16] (the number or layers) and y_range=[1,5] (the range of ratings).

IV. Discussion

A. Baseline

Starting from the most basic model which is predicting an unrated movie as the global mean of the rated movie, we can see that taking into account user and item specific effects can already improve our results. In that case taking the user or item mean for a specific user u or item i respectively decreases the RMSE (Table 1). The next step is to combined the two effects together. Using the baseline method from surprise, we model the rating as the global mean plus two bias, one for the user and one for the movie. This can be interpreted as the tendency of some user/movie to get higher or lower ratings than the average. This approach improved our result by having a RMSE of 0.999.

B. Collaborative Filtering

To improve our baseline model, we use CF based systems which allow to capture the connections between users and item to predict a rating.

Nearest Neighbors

We started with a basic model,KNN-Basic() (see Model and methods), which compute the rating as an average of it's nearest neighbor. However, the performance of the algorithm is significantly lower than the Baseline (Table 1), so we tried to combine both effects (KNN Baseline) and the RMSE was slightly higher (0.995) than for the Baseline only (0.999). Therefore Nearest neighbor models,which are good in capturing localized effects, do not capture much of the variation in ratings in our model compare to the interaction free parameters (bias).

Latent Factors

We tried three different latent factors models from the three library we used in order to model the ratings as overall effects of user item interactions unlike the previous approach. In each case, we direclty include baseline in our model as our first experiments suggested they are an important part of the rating variation. By doing so we improved the local RMSE compare to the KNN Baseline (Table1). Using the function ExplicitFactorizationModel() from Spotlight yield the best submission RMSE (1.025). However, they are all all based on the same principle and as expected they yield comparable results (Table1). The main difference is that spotlight allow to use mini-batch SGD, different loss functions and pyTorch optimizers.

Therefore, latent factors model are able to explain in higher proportion the variation in ratings than the nearest neighbor model.

C. Neural Network

With the neural network (NN) approach we wanted to see if it was possible to overpass the cold start problem that we had with MF. Which is that for items/users with low number of ratings its impossible to accurately make predictions, since users and items are modeled in the same latent space and ratings are predicted by linear operation [8]. One way to limit this problem was to increase the size of embedding vector (dimensionality of the latent space), but at a certain point this overfit the model. So the idea with the NN-based method was to insert non-linearity and hopefully insert new features in the model. Unfortunately, we did not get better result. For further work, we could try to combine MF-based methods and NN-based method to take advantages of both and maybe get a more efficient model.

V. Summary

The raise of streaming industry makes it necessary to have recommender systems that are as accurate as possible. Starting from our baseline estimate we were able to improve the accuracy of our recommender system by using the two main models of Collaborative Filtering combined with baseline, first nearest neighbors and then latent model factor. In order to do so, we used different Python libraries, like, Surprise, FastAI and Spotlight. Then neural nets approach did not outperformed our matrix factorization algorithm. However, as combining models yields better performances from our results, a model combining baseline neighbor and latent factors models in a neural nets could increase accuracy a little bit, but was not implemented in this project. In conclusion, our result suggest that latent factors models are superior to the others algorithms in recommending explicit feedback data like ours as in the Netflix challenge. It can certainly be improved by furthermore optimization and gathering more data, especially implicit feedback data such as likes, purchase history,search patterns, etc... and temporal dynamics.

REFERENCES

[1] EPFLMLTeachingTeam, "Lab 10," 2019.

[2] N. Hug, "Home." [Online]. Available: http://surpriselib.com/

[3] Y. Koren, "Factor in the neighbors: Scalable and accurate collaborative filtering," *ACM Trans. Knowl. Discov. Data*, vol. 4, no. 1, pp. 1:1–1:24, Jan. 2010. [Online]. Available: http://doi.acm.org/10.1145/1644873.1644874

[4] "Fastai collab documentation." [Online]. Available: https://docs.fast.ai/collab.html

[5] M. Cimini, "Deep learning for collaborative filtering (using fastai)," Aug 2019. [Online]. Available: https://medium.com/quantyca/deep-learning-for-collaborative-filtering-using-fastai-b28e197ccd59

[6] M. Kula, "Spotlight," https://github.com/maciejkula/spotlight, 2017.

[7] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

[8] Victor, "Collaborative filtering using deep neural networks (in tensorflow)," Jun 2019. [Online]. Available: https://medium.com/@victorkohler/collaborative-filtering-using-deep-neural-networks-in-tensorflow-96e5d41a39a1

[9] Y. Koren, "Factorization meets the neighborhood," *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*, 2008.

[10] B. Rocca, "Introduction to recommender systems," Jun 2019. [Online]. Available: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada

[11] G. Tanner, "Collaborative filtering with fastai." [Online]. Available: https://gilberttanner.com/blog/collaborative-filtering-with-fastai

[12] D. Vasani, "Collaborative filtering using fastai," Sep 2019. [Online]. Available: https://becominghuman.ai/collaborative-filtering-using-fastai-a2ec5a2a4049

[13] Y. Koren, "Factor in the neighbors," *ACM Transactions on Knowledge Discovery from Data*, vol. 4, no. 1, p. 1–24, Jan 2010.