Ukrainian Catholic University

Faculty of Applied Sciences

Computer Science & Business Analytics Programmes

Collaborative filtering and matrix factorization approaches in movie recommendation systems

Linear Algebra final project report

Authors:
Zoryana Herman
Andriy Yanovskyi
Halyna Koziak

12 May 2021

Abstract

Recommendation systems can be implemented using various approaches and techniques. It is also an evident application of matrix factorization that has been proven to severely increase the efficiency of such systems. There are a lot of papers and researches regarding choosing the best approach. In our project, we review these approaches and with the provided reasoning try to implement and look into algorithms behind the chosen one. We discuss the work of it on the real data, and its pros/cons implementation-wise.

The code is available on https://github.com/zoriankaH/collaborative_filtering

1 Introduction

1.1 Field

We use recommendation systems every day, when we do online shopping, listen to music or watch movies, so we can't argue with the importance of its implementation and accuracy of results. To limit the researched field, we will further discuss movie recommendation systems.

After a part of our research, we decided to focus on collaborative filtering that uses the data collected from the users with the similar interests to create a list of movies that the user can potentially enjoy. The use of this technique has shown severe improvements in recommender systems use, in the contrary for such techniques as content-based filtering in which we recommend movies only based on the user's history and do so based entirely on the movies features, the shortcomes of it being the absence of human factors, strong limitation on the recommendation lists etc. A lot of such techniques have a very visible impact in real life. For example, in the history of Netflix, the first recommender systems that were based on content-based filtering created a lot of inconvenience for users and not only produced relatively bad results. To solve the problem of a user with low ratings history, which occurs with the use of such technique, during the registration they were asked to rate the genres which they enjoy the most. Not only was it inconvenient but also dealt poorly as users are actually not that good in rating their own genres preferences, turns out it is a task that is the best to solve with the help of other users.

1.2 Problem setting

There is a huge amount of approaches in recommender systems, starting with filtering techniques to choosing the concrete algorithms from the field for implementation and testing. There are also a lot of choices to be made like the representation of raw data, the problems to emphasize on during implementation (such as sparse data or overfitting), what will be recommended to the new users etc. We researched the literature and tried to choose the best approach. During our work, we tried to address which approaches deal better with which type of data and upcoming problems, how do they do it. We also discuss whether it's hard to implement such algorithms, its efficiency and time-wise representation. Our main goal is to understand in-depth the chosen algorithm and test its implementation.

2 Overview of approaches

When considering collaborative filtering approach, we know that we are given a matrix with the corresponding user-movie ratings, and our main goal is to find a correlation between either movies or users to estimate the missing values (ungiven movie ratings), or "train" our model. Two approaches that we can use for this estimation are memory-based and model-based techniques.

In memory-based approach, we directly take our data matrix and perform calculations to either find users with similar taste (to do so usually functions like cosine similarity are applied to the matrix rows) and then estimate the values by taking weighted average from this user group, or we apply same functions to matrix columns and find previously mentioned movies correlation. This approach is relatively simple, and doesn't have any weaknesses towards changing data sets, however, because the computations are direct does depend on the size of the data and also deals poorly with sparse (meaning poor users or movie ratings history) datasets.

Model-based method implies that instead of directly using the data we use various machine learning algorithms (the four general approaches are clustering(non-parametric approach), classification, latent model, MDP, matrix factorization) to create prediction models from transaction data. We will further emphasize on matrix factorization techniques, nevertheless, the general idea of model-based CF is to apply those algorithms and use ML techniques that are often centred around guessing and calculating the error, or, in our case, it would be the accuracy of the predicted rating. Model-based approach comparatively is harder to implement and is not adaptive to the changing data. Because of the ML techniques it partially solves the problem of sparse data with the use of such approaches as reduction of the mean absolute error. [1]

Various testings show that a model-based approach usually predicts items with higher accuracy and is better timewise when tested on big data, as we predicted before. In the suggested research [2], both analytic and real life user testing studies were conducted on a real data recommendation system using memory-based and model-based approaches and compared by accuracy of predictions.

Matrix factorization approaches of model-based CF:

- Unconstrained matrix factorization, where we hold no constraints on the factor matrices U and V meaning the entries can be both positive and negative. Such algorithms as Stochastic gradient descent, Weighted Alternating Least Squares, Incremental Latent Component Training are of that kind.

- SVD - decomposes the original matrix into three matrices therefore creating clearer relations and predicts values by approximation. SVD usually works better for predicting explicit data (and that's what we have).

- Non-negative matrix factorization, our matrices have to have non-negative values, and we achieve that by using a specific minimization formula. We can also use such factorization to perform clustering, and use the results to generate features. In the study [5], this approach was compared to the latent factor and Bayesian probabilistic approaches by using the MAE. It was shown that it gives higher accuracy than the other two.

We decided to use SVD as our main collaborative filtering approach. it is considered to be a CF algorithm of higher quality. For example, when the decomposition is performed, as a result we would usually get only two matrices, but SVD also provides us with the diagonal matrix S. The iteration methods in SVD are also a bit different from algorithms like NNMF. SVD performs iteration until it produces a successively higher-rank factorization, while NNMF has an ability for you to choose good enough value regardless of the factorization rank. Steps like that add the extra efficiency but do require more additional work, for example more iteration steps. SVD allows us to create out-of-sample recommendations and that is a huge disadvantage of approaches like memory-based CF since they do not provide it. SVD solution also works well with dense matrices and for that some algorithms like Divide-and-conquer approach or general rectangular approach can be used. There are also additional algorithms for sparse matrices such as stochastic SVD, and although regular SVD deals with them relatively well computations can take longer for huge sparse matrices if no additional algorithms are used. Regardless of that, SVD is still considered to be one of the best approaches for model-based recommender systems and is widely used.

3 Algorithm analysis

In the study [4], different SVD approaches were tested. It was shown that SVD for item-based similarity predicts better than for user-based similarity. It was also shown that if the data is beforehand categorized, the predictions increased in their accuracy and the computations improved because of the decreasing of the size of decomposed matrices. This experiment produced better results for categorizing the movies (with different genres) than for categorizing the users (with gender etc), but the best results were shown when both were categorized. The categorization approach can be often seen in real life usages of recommendation systems, an example being movies genres categorization on Netflix.

To perform SVD, firstly, we build a mxn matrix R that consists of users as rows, movies as columns and the entries will be ratings (SVD allows both positive and negative values). As was noted before, most of the ratings are missing. We have to predict/find these unknown values. But sometimes this can become a problem, because not all MF algorithms are fitted to work well with missing values. One way to solve this is to simply impute the values, but it's a bad practise since then we will not be able to distinguish them from observed ones. For that, we use factorization and error minimization. We apply SVD algorithm to the matrix R and as a result of decomposition we get:

1. U - mxr matrix of users and their latent factors. Here, latent factors can be also interpreted as the features of movies that attract a certain user, it can be somewhat equated to the preferences in different genres etc, but still not quite as these features are not evidently defined. This gives us the opportunity to create such factors as favorite actors or even plot twists, which are much more thought out than, for example, previously defined movie genres. On this step our goal is to form a connection between latent factors and users, as well as factors and movies.

2. S - rxr diagonal matrix with positive values sorted in the decreasing order, it represents the features and how strong they (and their relation) are, meaning later we would preferably recommend a movie with feature that has more strength, given the same user-feature relation.

3. V - rxn matrix of movies and the same latent factors or features, so called "movie-to-concept" matrix.

SVD is a constrained algorithm and it states that U and V have to be mutually orthogonal. The three computed matrices are always unique with U and V having orthonormal columns. Also, from the dimensions of the matrices we can tell that SVD reduces the original dimension and maps all users and all movies into r-dimensional latent space.

It is generally performed with the use of an eigendecomposition method. Let's recall that we have our matrix R and it would be equal:

$$R = U * S * V^T$$

Matrices U and V consist of k largest eigenvectors of $R * R^T$ and $R^T * R$ respectively. Matrix S consists of square roots of the k largest eigenvalues of one of the matrices along its diagonal. Using the orthogonality properties, and knowing V and S we can also compute U by projecting A onto V:

$$U = R * V * S^{-1}$$

Using this same equation we can also find the A^T and see that we basically just have to swipe U and V matrices from the original equation.

After we applied SVD, we have to compute the ratings. To do so, we take our vectors u_i that represent the items and vectors v_j that represent users and calculate our rating r_{ij} as the dot product of two:

$$r_{ij} = u_i * v_j \qquad (1)$$

To find these values, we would naturally minimize the squared error, however, if we do so by the simplest approach as so, it could lead to overfitting:

$$min \sum_{i,j} (r_{ij} - u_i * v_j)^2$$

To avoid it, we add a regularization constant term and use the equation:

$$min \sum_{i,j} (r_{ij} - u_i * v_j)^2 + \beta(||u_i||^2 + ||v_j||^2) \qquad (2)$$

And for the final step of error reduction often a bias term is used by computing the average given ratings given by user and received by item. We perform these iterative steps of substituting missing values and execute them until we reach convergence.

4 Implementation

In our implementation, we decided to compare SVD with the NNMF algorithm while evaluating both on the same data. In our opinion, such comparison will best represent the advantages of SVD. Whilst it is questionless that SVD has advantages over memory-based models, NNMF is actually very similar to SVD in the theoretical sense behind it.

Most of the algorithms were implemented using Scikit-learn[7] - ML-oriented python library. **TruncatedSVD** function that we use is implemented in a way that does not include data centering therefore it works well with sparse matrices. And **NMF** function that is implemented with the use of Frobenius and Elementwise norms.

When we apply the SVD function to our data, we have to set the k dimension (which is 2 by default) to which we reduce our vectors. As a result we get our output vectors and compute the rating in cycle:

```
Data: mxn matrix R, dimension k, stopping criteria $\alpha$
Result: Matrix with predicted values
TruncatedSVD(R,k);
repeat
    For each user i assign a k-vector $u_i$ and for each item j a k-vector $v_j$;
    Calculate $r_{ij}$ using (1);
    Substitute missing entry in R;
    if (2) $\leq \alpha$ then
        i += 1 or j += 1, i = 0;
        break;
    else
        repeat;
    end
until for all random pairs $i < j$;
```

Algorithm 1: Iterative use of SVD

NMF algorithm decomposes ratings matrix into two matrices from which we get our correlation vectors. Let us detonate matrix R:

$$R = W * H$$

Then our algorithm will be:

```
Data: mxn matrix R, maximum number of steps n, regularization parameter $\lambda$,
      error threshold $\beta$, learning rate $x$*
Result: Matrices W and H
Initialize W and H with random small values;
k = 0;
while $k < n$ do
    repeat
        if $R_{ij} > 0$ then
            $err = W_i * H_j - R_{ij}$;
            $W_i = W_i - x * (err * H_j^T + *W_i)$;
            $H_j = H_j - x * (err * W_i^T + *H_j)$;
            Substitute $W_i$ and $H_j$ into matrices;
        else
            go to step (1)
        end
        Calculate total error;
        if total error $< \beta$ then
            break
        else
            go to step (1)
        end
    until for all random pairs $i < j$ step (1);
    k + = 1;
end
```

Algorithm 2: Iterative NMF

*given parameters are not essential, however they improve the work of the algorithm

After we compute our decomposition, we just have to multiply vectors corresponding to desired user and movie to get the prediction value, similar to such process with SVD.

To test the accuracy of both algorithms we also evaluated RMSE.

$$\text{err} = \sqrt{\sum_{i,j} (\hat{r}_{ij} - r_{ij})^2}.$$

This gives us the idea of how well our algorithms works, and as a result we got:

RMSE for SVD:

```
Out[213]: 3.0964271533374723
```

RMSE for NMF:

```
Out[220]: 2.5349280506087943
```

These values are relatively not bad considering the sparsity of the data on which we conduct the testing.

5 Experiments

5.1 Data

We decided to use the Kaggle dataset[8] which contains both relatively huge and small datasets of movie ratings that can be from 1 to 5, with 0 meaning absent value. Data is presented with csv files. It also offers a variety of additional movie features. We used the additional library functions to compute a compressed sparse matrix, which is good for our case of experiment.

5.2 Evaluation metrics

To compare two algorithms we used two accuracy metrics:

- MAP - Mean Average Precision

  MAP helps us to understand how relevant are the recommended items. It also partially implies that we have a preference or rank of our recommendation list - and it is certainly true, because we need to recommend only 10 movies or so to each user. The main idea behind MAP is to judge how good were the recommended items and also if we had more desirable items in the beginning of our list. Two terms we will need to use with MAP are precision and recall both at cutoff k, the first one stands for rate of liked items (correct positive) to predicted items and second - rate of liked items to all possible relevant items. And to make them at cutoff k and consider the importance of the ordering, we will be calculating these values firstly considering only the first one, then first to and so on to k elements.

  If we consider n recommended items, we got m relevant items, p(k) calculates precision of kth item and r(k) is the function that is equal to 1 if item is relevant and 0 otherwise, than we compute mean average precision using formula:

$$\frac{1}{m}\sum_{k=1}^{n} p(k) * r(k)$$

- NDCG - Normalized Discounted Cumulative Gain

  As well as MAP, NDCG rates according to ranking and how well its computed. This algorithm uses cumulative gain, which is just the sum of all relevance scores. And to apply positioning matter to that we compute discounted cumulative gain:

  $$\sum_{i=1}^{n} \frac{(relevance_i)}{log_2(i+1)}$$

  To achieve better testing, we also normalize DCG, and we do so using DCG of recommender list and DCG of the ideal ordered list, then:

  $$NDCG = \frac{DCG}{IDCG}$$

5.3 Results

We applied these metrics to both algorithms results, and got:

NDCG for SVD:

```
Out[214]: 0.978169480322486
```

NDCG for NMF:

```
Out[221]: 0.9826583668972942
```

MAP for SVD:

```
0.9603315348888394
```

MAP for NMF:

```
0.974300163004246
```

These values were abducted on the whole ratings set, it would, however, be more precise if we were to test it on the set of recommendations. Nevertheless, the values we obtained were very good, which means that our models are trained well.

We can also see that the values for NMF showed better results, both in this metrics and with RMSE testing.

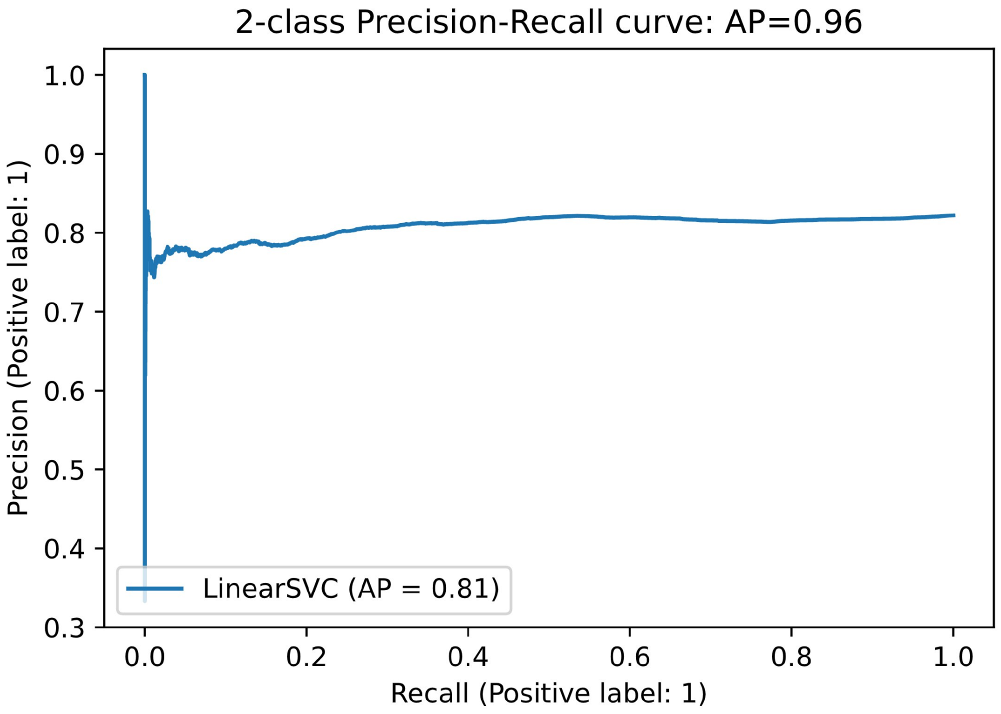
In our implementation we also provided the computed recommendations for given users sorted in the decreasing order.

Example of such recommendations using SVD:

```
[(1.02, 296),
 (0.98, 356),
 (0.96, 318),
 (0.94, 260),
 (0.93, 593),
 (0.87, 2571),
 (0.87, 1196),
 (0.84, 1270),
 (0.83, 1198),
 (0.8, 527)]
```

Here we printed out the predicted rating and the corresponding movie id from the dataset.

The last thing we did, we build a precision classifier, using the relations of marking all the ratings higher than 2.5 as 1 and the other zero, to represent the hit/miss rate. Using that, we plotted a curve that helps us understand the work of our system. Basically the higher values are on the curve - the better, it shows that we both got high precision and recall (both of those values were mentioned above), and the higher those values are - the better are our recommendations.



Our curve shows that we have pretty stable values of both kind, and represents the good obtained results.

6 Conclusions

Surprisingly, we were proven wrong in some of our assumptions. We were expecting that SVD algorithm will show better metrics results and therefore justify the complexity of implementation, however, all results were a little better for NMF algorithm. There are although a few things that NMF failed relatively to SVD in our implementation. First of all, NMF was much slower in work compared to SVD that was very quick for the data size.

To deal with this problem we tried to use such additional parameters as maximum number of steps when using NMF, but even then it did not show much approvement. One of the reasons can be the scale of our recommender systems. Usually such recommendations are computed with the millions, not thousands like we had, data records. On this scale, NMF would probably become not at all suitable considering the fact that we would also continiously update our model and train again - in this implementation it would be impossible to use NMF. SVD, however seems very suitable and it's important to say that even if the recommendations were a little less accurate, this is still a very good result for metrics of that kind.

References

[1] ttps://anjs.edu.iq/index.php/anjs/article/view/1736/1551

[2] ttps://qoribmunajat.github.io/files/comparative-analysis-memory-based-model-based-recommendation-systems.pdf

[3] ttps://www.diva-portal.org/smash/get/diva2:1214255/FULLTEXT01.pdf

[4] ttps://etd.lib.metu.edu.tr/upload/12612129/index.pdf

[5] Non-Negative Matrix Factorization for Recommender Systems Based on Dynamic Bias

[6] haru C. Aggarwal, Recommender Systems

[7] ttps://medium.com/swlh/rank-aware-recsys-evaluation-metrics-5191bba16832

[8] ttps://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.231.5583rep=rep1type=pdf

[9] ttps://proceedings.neurips.cc/paper/1993/file/49c9adb18e44be0711a94e827042f630-Paper.pdf