华南理工大学

South China University of Technology

The Experiment Report of *Machine Learning*

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Haishan Ao
Shen Fu

Supervisor:
Qingyao Wu

Student ID:
201720145051
201720144955

Grade:
Graduate

December 28, 2017

Recommender System Based on Matrix Decomposition

***Abstract*—Recommendations can be generated by a wide range of algorithms. While user-based or item-based collaborative filtering methods are simple and intuitive, matrix factorization techniques are usually more effective because they allow us to discover the latent features underlying the interactions between users and items. In our experiment, we constructed a recommendation system based on Matrix Decomposition under small-scale dataset. The result on real world examples have shown that matrix factorization models are superior to classic nearest-neighbor techniques for producing product recommendations, allowing the incorporation of additional information such as implicit feedback, temporal effects, and confidence levels.**

I. Introduction

THERE is probably no need to say that there is too much information on the Web nowadays. Search engines help us a little bit. What is better is to have something interesting recommended to us automatically without asking. Indeed, from as simple as a list of the most popular bookmarks on Delicious, to some more personalized recommendations we received on Amazon, we are usually offered recommendations on the Web.

Recommendations can be generated by a wide range of algorithms. While user-based or item-based collaborative filtering methods are simple and intuitive, matrix factorization techniques are usually more effective because they allow us to discover the latent features underlying the interactions between users and items. Of course, matrix factorization is simply a mathematical tool for playing around with matrices, and is therefore applicable in many scenarios where one would like to find out something hidden under the data.

This paper deals with a recommendation system based on Matrix Decomposition on the MovieLens 100K benchmark. The result on real world examples have shown that matrix factorization models are superior to classic nearest-neighbor techniques for producing product recommendations, allowing the incorporation of additional information such as implicit feedback, temporal effects, and confidence levels.

The rest of the paper is organized as follows. In Section II, we introduce recommender system strategies, matrix factorization methods, a basic matrix factorization model and the details of learning algorithms. The details of our experiments and evaluation are presented in Section III and Section IV concludes the paper.

II. Methods and theory

Personalized recommendations systems are particularly useful for entertainment products such as movies, music, and TV shows. Many customers will view the same movie, and each customer is likely to view numerous different movies. Customers have proven willing to indicate their level of satisfaction with particular movies, so a huge volume of data is available about which movies appeal to which customers. Companies can analyze this data to recommend movies to particular customers.

Matrix factorization techniques are usually more effective because they allow us to discover the latent features underlying the interactions between users and items. Of course, matrix factorization is simply a mathematical tool for playing around with matrices, and is therefore applicable in many scenarios where one would like to find out something hidden under the data [1].

A. Matrix Factorization Methods

Some of the most successful realizations of latent factor models are based on matrix factorization. The intuition behind using matrix factorization to solve this problem is that there should be some latent features that determine how a user rates an item. For example, two users would give high ratings to a certain movie if they both like the actors/actresses of the movie, or if the movie is an action movie, which is a genre preferred by both users. Hence, if we can discover these latent features, we should be able to predict a rating with respect to a certain user and a certain item, because the features associated with the user should match with the features associated with the item. In its basic form, matrix factorization characterizes both items and users by vectors of factors inferred from item rating patterns. High correspondence between item and user factors leads to a recommendation.

These methods have become popular in recent years by combining good scalability with predictive accuracy. In addition, they offer much flexibility for modeling various real-life situations. Recommender systems rely on different types of input data, which are often placed in a matrix with one dimension representing users and the other dimension representing items of interest. The most convenient data is high-quality explicit feedback, which includes explicit input by users regarding their interest in products.

We refer to explicit user feedback as ratings. Usually, explicit feedback comprises a sparse matrix, since any single user is likely to have rated only a small percentage of possible items. One strength of matrix factorization is that it allows incorporation of additional information. When explicit feedback is not available, recommender systems can infer user preferences using implicit feedback, which indirectly reflects opinion by observing user behavior including purchase history, browsing history, search patterns, or even mouse movements. Implicit feedback usually denotes the presence or absence of an event, so it is typically represented by a densely filled matrix.

B. A Basic Matrix Factorization Model

Just as its name suggests, matrix factorization is to, obviously, factorize a matrix, i.e. to find out two (or more) matrices such that when you multiply them you will get back the original matrix.

As mentioned above, from an application point of view, matrix factorization can be used to discover latent features underlying the interactions between two different kinds of entities. (Of course, you can consider more than two kinds of entities and you will be dealing with tensor factorization, which would be more complicated.) And one obvious application is to predict ratings in collaborative filtering.

Matrix factorization models map both users and items to a joint latent factor space of dimensionality f, such that user-item interactions are modeled as inner products in that space. Each item i is associated with a vector $q_i \in \mathbb{R}^f$, and each user u is associated with a vector $p_u \in \mathbb{R}^f$. For a given item i, the elements of q_i measure the extent to which the item possesses those factors, positive or negative. For a given user u, the elements of p_u measure the extent of interest the user has in items that high on the corresponding factors, again, positive or negative. The resulting dot product, $q_i^T p_u$, captures the interaction between user u and item i – the user's overall interest in the item's characteristics. This approximates user u's rating of item i, which is denoted by r_{ui}, leading to the estimate

$$\hat{r_{ri}} = q_i^T p_u. \tag{1}$$

The major challenge is computing the mapping of each item and user to factor vectors $q_i, p_u \in \mathbb{R}^f$. After the recommender system completes this mapping, it can easily estimate the rating a user will give to any item by using Equation 2.

Such a model is closely related to singular value decomposition (SVD), a well-established technique for identifying latent semantic factors in information retrieval. Applying SVD in the collaborative filtering domain requires factoring the user-item rating matrix. This often raises difficulties due to the high portion of missing values caused by sparseness in the user-item ratings matrix. Conventional SVD is undefined when knowledge about the matrix is incomplete. Moreover, carelessly addressing only the relatively few known entries is highly prone to overfitting.

To learn the factor vectors (p_u and q_i), the system minimizes the regularized squared error on the set of known ratings:

$$\min_{q^*,p^*} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2 + \lambda(\parallel q_i \parallel^2 + \parallel p_u \parallel^2) \tag{2}$$

where K is the set of the (u,i) pairs for which r_{ui} is known (the training set). The constant λ controls the extent of regularization and is usually determined by cross-validation.

C. Learning Algorithms

Two approaches to minimizing Equation 2 are stochastic gradient descent (SGD) and alternating least squares (ALS).

1) Stochastic gradient descent: For each given training case, the system predicts r_{ui} and computes the associated prediction error

$$e_{ui} \stackrel{def}{=} r_{ui} - q_i^T p_u.$$

Then the parameters update by a magnitude proportional to γ in the opposite direction of the gradient, yielding:

- $q_i \leftarrow q_i + \gamma \cdot (e_{ui} p_u - \lambda \cdot q_i)$
- $p_u \leftarrow p_u + \gamma \cdot (e_{ui} q_i - \lambda \cdot p_u)$

.

This approach ease with a relatively fast running time. Yet, in some cases, it is beneficial to use ALS optimization.

2) Alternating least squares: ALS techniques rotate between fixing the q_i's and fixing the p_u's. When all p_u's are fixed, the system recomputes the q_i's by solving a least-squares problem, and vice versa. This ensures that each step decreases Equation 2 until convergence.

D. Recommender System Strategies

Broadly speaking, *Recommender systems* are based on one of two strategies C through content-based filtering (also known as the personality-based approach) or through collaborative filtering.

The content filtering approach creates a profile for each user or product to characterize its nature in order to recommend additional items with similar properties. For example, a movie profile could include attributes regarding its genre, the participating actors, its box office popularity, and so forth. User profiles might include demographic information or answers provided on a suitable questionnaire. The profiles allow programs to associate users with matching products. Of course, content-based strategies require gathering external information that might not be available or easy to collect.

An alternative to content filtering relies only on past user behaviorłfor example, previous transactions or product ratingsł without requiring the creation of explicit profiles. This approach is known as *collaborative filtering*. Collaborative filtering approaches build a model from a user's past behaviour (items previously purchased or selected and/or numerical ratings given to those items) as well as similar decisions made by other users.

The two primary areas of collaborative filtering are the neighborhood methods and latent factor models. Neighborhood methods are centered on computing the relationships between items or, alternatively, between users. The item oriented approach evaluates a users preference for an item based on ratings of neighboring items by the same user. A products neighbors are other products that tend to get similar ratings when rated by the same user.

Latent factor models are an alternative approach that tries to explain the ratings by characterizing both items and users on, say, 20 to 100 factors inferred from the ratings patterns. In a sense, such factors comprise a computerized alternative to the aforementioned humancreated song genes. For movies,

the discovered factors might measure obvious dimensions such as comedy versus drama, amount of action, or orientation to children; less well-defined dimensions such as depth of character development or quirkiness; or completely uninterpretable dimensions. For users, each factor measures how much the user likes movies that score high on the corresponding movie factor.

Here we present the schematic diagram of the architecture of the GroupLens Research collaborative filtering engine in Fig.1. The user interacts with a Web interface. The Web server software communicates with the recommender system to choose products to suggest to the user. The recommender system, in this case a collaborative filtering system, uses its database of ratings of products to form neighborhoods and make recommendations.

A major appeal of collaborative filtering is that it is domain free, yet it can address data aspects that are often elusive and difficult to profile using content filtering. While generally more accurate than content-based techniques, collaborative filtering suffers from what is called the cold start problem, due to its inability to address the systems new products and users. In this aspect, content filtering is superior.

III. Experiments

Though recognizing individual digits is only one of many problems invoved in designing a practical recognition system, it is an excellent benchmark for comparing shape recognition methods. In order to illustrate the performance of Convolutional neural network, we setup experiments on the MNIST handwritten digit benchmark.

A. Dataset

In this experiment, we use MovieLens 100K Dataset. It is a stable benchmark dataset. MovieLens data sets were collected by the GroupLens Research Project at the University of Minnesota. MovieLens 100K dataset is available at https://grouplens.org/datasets/movielens/100k/.

This data set consists of: * 100,000 ratings (1-5) from 943 users on 1682 movies. * Each user has rated at least 20 movies. * Simple demographic info for the users (age, gender, occupation, zip)

The data was collected through the MovieLens web site (movielens.umn.edu) during the seven-month period from September 19th, 1997 through April 22nd, 1998. This data has been cleaned up - users who had less than 20 ratings or did not have complete demographic information were removed from this data set.

u.data – Consisting 10,000 comments from 943 users out of 1682 movies. At least, each user comment 20 videos. Users and movies are numbered consecutively from number 1 respectively. The data is sorted randomly.

u1.base / u1.test are train set and validation set respectively, separated from dataset u.data with proportion of 80% and 20%. It also make sense to train set and validation set from u1.base / u1.test to u5.base / u5.test.

B. Loss Function

The loss function in the experiment is modified squared error

$$e_{ij}^2 = (r_{ij} - \sum_{K=1}^{K} p_{ik}q_{kj})^2 + \frac{\beta}{2}\sum_{K=1}^{K}(\|P\|^2 + \|Q\|^2) \quad (3)$$

In other words, the new parameter β is used to control the magnitudes of the user-feature and item-feature vectors such that P and Q would give a good approximation of R without having to contain large numbers. In practice, β is set to some values in the range of 0.02. The new update rules for this squared error can be obtained by a procedure similar to the one described above. The new update rules are as follows.

$$p'_{ik} = p_{ik} + \alpha\frac{\partial}{\partial p_{ik}}e_{ij}^2 = p_{ik} + \alpha(2e_{ij}q_{kj} - \beta p_{ik}) \quad (4)$$

$$q'_{kj} = q_{kj} + \alpha\frac{\partial}{\partial q_{kj}}e_{ij}^2 = q_{kj} + \alpha(2e_{ij}p_{ik} - \beta q_{kj}) \quad (5)$$

C. Experiment Setup

In our experiment, u1.based and u1.test are chosen to be the training dataset and validation dataset respectively.

We use gradient descent method to complete the experiment, taking into account that the dataset is very sparse. We setup a experiment use jupyter according to steps shown as follow.

1) Read the data set and divide it (or use u1.base / u1.test to u5.base / u5.test directly). Populate the original scoring matrix against the raw data, and fill 0 for null values.
2) Initialize the user factor matrix and the item (movie) factor matrix , where is the number of potential features.
3) Determine the loss function and hyperparameter learning rate and the penalty factor.
4) Use the gradient descent method to decompose the sparse user score matrix, get the user factor matrix and item (movie) factor matrix:
   4.1 Calculate all sample's loss gradient of specific row(column) of user factor matrix and item factor matrix;
   4.2 Use GD to update the specific row(column) of user factor matrix and item (movie) factor matrix;
   4.3 Calculate the on the validation set, comparing with the of the previous iteration to determine if it has converged.
5) Repeat step 4. several times, get a satisfactory user factor matrix and an item factor matrix, Draw a curve with varying iterations.
6) The final score prediction matrix is obtained by multiplying the user factor matrix and the transpose of the item factor matrix.

.

D. Result and Analysis

The hyperparameters we chose in this experiment are shown as follow.

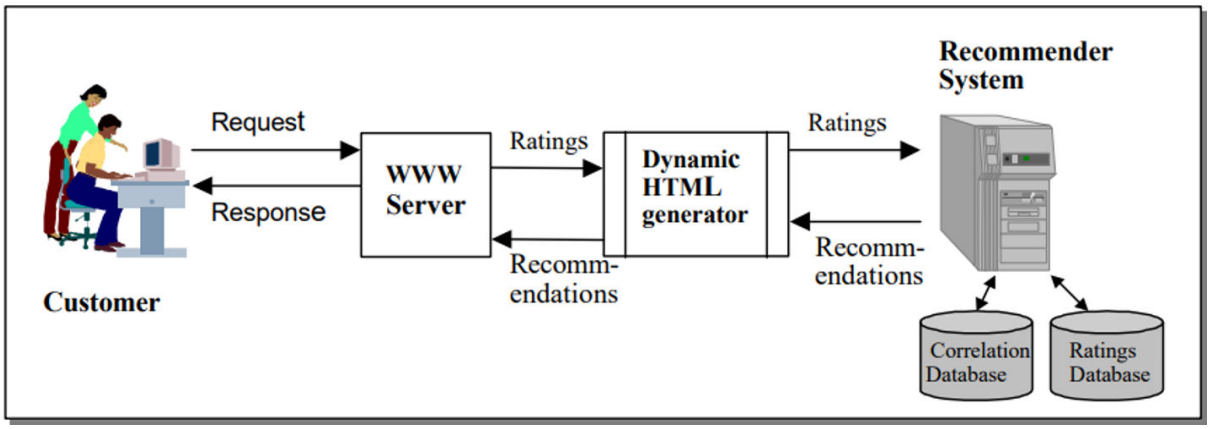

Fig. 1: The architecture of a Recommender System.

1) Potential feature K = 30
2) Max iteration = 200
3) Learning rate = 0.002
4) Penalty factor = 0.02

The result on MovieLens 100K Dataset is shown in Fig.2

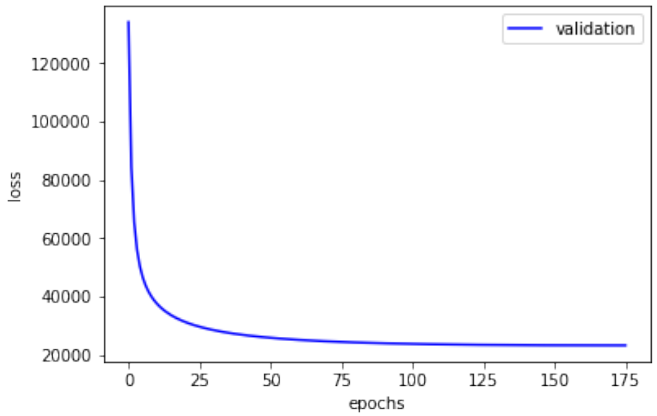

Fig. 2: Result on MovieLens 100K Dataset.

As the result shown above, we can see that the loss of validation set decreased after some iterations. The loss of validation dataset achieve 23244.3449006 finally. The best result of this experiment indicates that Matrix Decomposition have a good performance in recommender system.

IV. Conclusion

This paper deals with a Matrix Decomposition based recommender system on the ML-100K benchmark. Through experiments, we have shown that low MSE could be achieved using a Matrix Decomposition based recommender system for ML-100K dataset. Matrix Decomposition based recommender system has been shown to obtain the promising performance in recommendation area, but there still are some problems, for example it can not deal with the one-class data and sparse data very well. Meanwhile, they are faced by the problem of big data recommendation because of high computational complexity.

References

[1] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, 2009.