Recommendation Systems Using Collaborative Filtering

Kevin Zhang, Qinyuan Sun, and Austin Mease

March 21, 2016

1 Introduction

With millions of products on the retail market today, many companies can benefit from recommendation systems to increase user satisfaction by quickly matching customers with the products that they will be the most interested in. This area of machine learning and pattern recognition has recently emerged as a very hot and lucrative topic. In fact, Netflix held a competition in which they awarded 1 million dollars to the team that could increase the accuracy of their recommendation system by 10 percent or more.

Recommendation systems utilize data processing techniques to give accurate predictions about a user's interest in a particular item. One of the most popular techniques for designing these systems is collaborative filtering which relies on previously attained preferences in order to predict items certain users will enjoy. In this lab, we will explore the design of recommendation systems and introduce some common, effective methods of collaborative filtering.

2 Overview

2.1 Collaborative Filtering

Collaborative Filtering(CF) is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). By analyzing the past preferences and behaviors of users in the system, collaborative filtering can look at relationships between users and products to identify potential user-item interactions. After establishing these associations, the system can easily produce tailor made suggestions for each user.

The two most common techniques in collaborative filtering are memory based and model based methods. A memory based approach, exemplified by the neighborhood method, estimates unknown ratings based on the known ratings of similar users or similar items (local phenomena). A model based approach uses matrix factorization to uncover latent factors in the data (global phenomena) that explain known ratings.

2.2 Sparsity of Data Sets

Data sets used in recommendation systems come in the form of a user (n) by item (m) ratings matrix, where each entry contains a rating by a user for an item. Items that a user has not rated are empty and often times, these data sets are very sparse. Take the MovieLens data set, which has 21 million ratings with 30,000 movies and 230,000 users. A person is unlikely to see even one percent of all of the possible movies in the data set. As a result, recommendation systems must deal with this inherit sparsity problem, where the majority of the entries in the matrix are not filled.

2.3 Memory Based Collaborative Filtering

To estimate a rating for a particular user and item, a memory based model can either compare similarities between items or between users. For instance, a user-based neighborhood method picks the top-L most similar users from the training set based on a similarity measure. There are different similarity measures, including cosine-similarity and the Pearson Correlation measure, and using different measures might give you different results. We will be using the Pearson Correlation measure in this lab to determine how similar two users are based on movies that both have rated. Since many ratings in the matrix are unknown, only users that have at least k ratings in common with the target user will be considered in the similarity measure calculation. To make a prediction for a user's rating of a particular item i, the system takes the weighted average of the L most similar user's ratings for item i, as the formula shows.

$$\hat{r}_{ui} = \frac{1}{k} \sum_{u' \epsilon U} simil(u, u') r_{u',i} \tag{1}$$

Where

- $\hat{r}_{ui}$ represents a prediction of a rating that user u gives to item i
- k represents a normalization factor, defined by $\sum_{u' \in U} |simil(u, u')|$.
- $simil(u, u')$ represents the similarity measure. In this lab, it is the Pearson Correlation measure.
- u represents current user.
- U represents the set of L users that are the most similar to user u.
- u' represents a user in the set U.

Benefits and Drawbacks of Memory Based Methods

Memory based recommendation systems leverage local information. For instance, consider an item-item based approach. If a particular user gives the first Lord of the Rings movie a high rating, he is likely to give the second and third movies a high rating because all three movies are similar. By comparing similar movies or similar users, a recommendation system can identify these types of localized relationships between users or items. Memory based approaches are also intuitive and have results that can be interpreted easily. Since memory

based systems utilize similarity measures, they can explain that a user received a recommendation for a particular item because he gave a high rating to a similar item. However, such systems will struggle to find good predictions for users who report similar tastes to few or no other users (ie. low similarity measure values). Other notable issues include long computation times, heavy memory usage, and a lack of consideration of global structure in the data set.

2.4 Model Based Collaborative Filtering

Model based collaborative filtering techniques use matrix factorization to decompose the user-item ratings matrix into separate user-feature and item-feature matrices [2]. This decomposition yields a mapping between users and items onto a latent feature space. Latent features may not have obvious meanings but can characterize some underlying properties of the users and items, revealing abstract characteristics of the data that may not be able to be extracted through other methods. For example, a factor may indicate the level of action that a user prefers in a movie, measure abstract features like level of character development, or indicate something completely uninterpretable. In this latent feature space, a higher measurement value for a particular feature shows a higher preference for that feature. For instance, a movie that measures a 10 for a feature that indicates comedic value is likely to be considered funnier than a movie that measures 5.6. Similarly, a user who measures a 5 for a feature that indicates level of action will likely enjoy the Die Hard movies more than a user who measures a 0.5. Remember that these latent feature values are learned through factorization and are not previously obtained values through experimentation or observation. Also, note that the original ratings are positive values, though the features obtained in the factorization can be negative to indicate a user's level of dislike towards a particular factor.

After mapping users and items to a latent feature space, a particular rating $\hat{r}_{ui}$ can be approximated by taking the dot product of a user feature vector p_u with an item feature vector q_i. The optimal number of latent features, f, must be obtained through cross validation.

$$\hat{r}_{ui} = q_i^T p_u \qquad p_u, q_i \in \mathbb{R}^f \tag{2}$$

This matrix factorization can be obtained by minimizing the squared error on the known ratings. A regularization term is added to avoid over fitting on the training data.

$$min_{p,q} \sum_{u,i} (r_{ui} - \hat{r}_{ui})^2 + \lambda(||p_u||^2 + ||q_i||^2) \tag{3}$$

Where

- r_{ui} represents the true rating for user u at item i
- q_i represents factor vector for item i
- p_u represents factor vector for user u
- λ represents the regularization parameter

Incorporating Bias

There are user and item biases in recommendation systems that are independent of any user-item interaction. For example, some users may be more critical than others or public opinion may drive the perception of a particular item. To deal with this, a system may attribute some portion of a rating to biases. The system can then correct for these biases so that it only considers the portion of the rating that contributes to a user-item interaction. A simple calculation of bias in equation 4 considers global average rating (μ), item bias (b_i), and user bias (b_u). This baseline estimate b_{ui} of the user's rating can then be added to the portion of the rating explained by the user-item interaction ($q_i^T p_u$) to obtain a new rating prediction in equation 5. Finally, we can modify the previous objective function (equation 3) to obtain a new function where the rating prediction and regularization term considers the baseline estimate and bias terms (equation 6).

$$b_{ui} = \mu + b_i + b_u \tag{4}$$

$$\hat{r}_{ui} = b_{ui} + q_i^T p_u \qquad b_u \in \mathbb{R}^n, b_i \in \mathbb{R}^m \tag{5}$$

$$min_{p,q,b} \sum_{u,i} (r_{ui} - \hat{r}_{ui})^2 + \lambda(||p_u||^2 + ||q_i||^2 + b_u^2 + b_i^2) \tag{6}$$

Where

- μ represents the overall average rating
- b_i represents deviations (bias) from μ for a given item
- b_u represents deviations (bias) from μ for a given user
- b_{ui} represents the baseline estimate for user u's rating on item i independent of any user-item interaction.

Implicit Data and the Cold Start Problem

A major issue collaborative filters face is the cold start problem. When a new user joins the system, there is little or no explicit data for their past choices. As a result, collaborative filtering systems will have difficulty drawing inferences for these users. However, some systems may leverage implicit feedback, in which the system deduces user preferences indirectly through additional related data like search, browsing, and purchase history. Although explicit ratings data is the primary information used in collaborative filtering, systems may also supplement this information with implicit data to make suggestions more accurate.

In this lab, we do not have access to implicit data like user browsing history or viewing history. However, we can consider the movies that user has chosen to rate as a kind of implicit data. By taking the time to give a rating, the user has implicitly given a preference (good/bad/neutral) for a movie regardless of whether they rated it low or high. To utilize this kind of implicit data, we can create an n x m binary matrix N where an entry ui is filled (1) if the user u has previously rated (ie. given a preference for) an item i.

SVD++

To incorporate implicit data, we can modify the previous objective function and prediction equation to obtain a model commonly called SVD++ [1]. Like our previous models, this model continues to use matrix factorization to generate a mapping between users/items and a latent feature space and bears little resemblance to the traditional singular value decomposition (SVD) which factorizes a matrix into two orthonormal matrices and a diagonal matrix of singular values. Specifically, SVD++ involves modeling the user as a linear combination of his latent features p_u with the features of the items which he implicitly prefers. In this sense, the model requires a separate set of feature vectors y_j for each item j from the set of items for which the user u has previously provided an implicit preference ($N(u)$).

By extending equation 3, we can generate equation 7 to incorporate this new user model into the formula for $\hat{r}_{ui}$. Note that as the amount of implicit data increases, its role is increasingly emphasized in our model by scaling the y_j vectors by the term $|N(u)|^{-1/2}$. Finally, we can modify the objective function in equation 6 to obtain a function which incorporates the baseline estimate as well as implicit user data given by equation 8.

$$\hat{r}_{ui} = b_{ui} + q_i^T(p_u + \frac{1}{\sqrt{|N(u)|}} \sum_{j \epsilon N(u)} y_j) \qquad y_j \in \mathbb{R}^f \tag{7}$$

$$min_{p,q,b,y} \sum_{u,i} (r_{ui} - \hat{r}_{ui})^2 + \lambda(||p_u||^2 + ||q_i||^2 + b_u^2 + b_i^2 + \sum_{j \epsilon N(u)} ||y_j||^2) \tag{8}$$

Where

- $N(u)$ represents the set of items for which a user u has previously given an implicit preference. In this lab, N(u) will represent a binary vector of items where an entry is filled (1) if the user has previously rated the item.
- y_j represents a factor vector for an item j in the set $N(u)$.

Benefits and Drawbacks of Model Based Methods

Unlike neighborhood methods, model based approaches consider global structure in the data set. While both approaches suffer from the cold start problem, model based approaches tend to avoid over fitting to a sparse ratings matrix by including implicit data as well as a regularization term in the objective function. Model based methods also typically scale better than memory based methods, as online algorithms like SVD++ can efficiently handle the addition of new users to the system. However, since the latent features that model based methods uncover typically do not have straightforward, real-world interpretations, these systems may have difficulty explaining why users receive certain recommendations.

Iterative Methods for Matrix Factorization

Iterative processes are used to allow model-based methods to compute matrix factorization efficiently. In this lab, we will use Stochastic Gradient Descent

(SGD), a common technique used to minimize objective functions that can be expressed as a sum of differentiable functions. At each iteration, the algorithm performs an update of the target variables by computing the subgradient at a single training example. For example, the update rule for q is given below.

$$q_i = q_i - \frac{\gamma}{2}\nabla_q W() \tag{9}$$

Where

- w_t represents the target iterate
- γ is the step size
- $\nabla_q W()$ is the gradient of the objective function W at a single training example with respect to the target variable, q

3 Warm-up

1. What are the problems with using traditional SVD on a recommendation system data set?
2. What are the advantages and disadvantages of memory-based methods?
3. What are the advantages and disadvantages of model-based methods?
4. Which method do you expect will perform better, Memory-based (Neighborhood Method) or Model-based (SVD++)? Explain your reasoning.
5. Using equation 8, derive explicit formulas for p_u and q_i for each iteration in stochastic gradient descent for the objective function in equation 3.
6. Derive explicit formulas for p_u, q_i, b_u and b_i for each iteration in stochastic gradient descent for the objective function in equation 6.
7. Derive explicit formulas for p_u, q_i, b_u, b_i, and y_j for each iteration in stochastic gradient descent for the objective function in equation 8.

4 Lab

In this lab, we will use a small subset of the MovieLens data set with 943 users (n), 1682 movies (m), and approximately 100,000 ratings. The n x m ratings matrix is provided in *ratings.m*. You will implement the neighborhood method as well as latent factor models using stochastic gradient descent and SVD++. Code skeletons have been provided, for which the bulk of the code has been completed.

4.1 Implementing the Neighborhood method

1. Complete the code in the function *neighborhood.m* to find the top 40 (L) users that are the most similar to a target user. Compute the Pearson correlation coefficient for each of user in the training set using the MatLab command *corr()*. Only users that have 60 (k) ratings in common with the

target user should be considered in the similarity measure calculation (see the MatLab function *intersect()*). In the provided code skeleton, L and k have been added as constants.

2. Using equation 1 from the overview, implement the prediction equation in *main.m*.

4.2 Implementing the Latent Factor Model and SVD++

3. Using the formulas derived in question 6 in the warm-up, complete the code for updating q_i, p_u, b_u, and b_i in each iteration of stochastic gradient descent in *sgd_bias.m*. Also complete the code for calculating $\hat{r}_{ui}$.

4. Using the formulas derived in question 7 of the warm up, complete the code for updating q_i, p_u, b_u, b_i, and each y_j in *svdpp.m*. Also complete the code for $\hat{r}_{ui}$, for which you will need to calculate $|N(u)|$ as well as the sum of y_j's that the user has rated.

4.3 Evaluation

In order to evaluate the performance of the algorithms, 30% of the ratings for each user in the testing set are blanked out. The remaining 70% of each of those user's ratings are used to establish a baseline for user behavior. The code for this has already been implemented in *main.m* and will output the root mean squared error (RMSE) for each method.

4.4 Analysis

5. Run *main.m* to find the RMSE for the neighborhood method, SGD with bias, and SVD++. Compare the results using a random 100 rows, 400 rows, and 800 rows. Use a testing set of size 130 rows. In *main.m*, you can change the size of the training set in the variable n_train and the size of the testing set in the variable n_test. Also analyze the actual run time (using timeit or some other means) and place your results in the table below. Note that when the training size is 800, *main.m* will take about 10-15 minutes to complete.

6. Which method performed the best? Did your results match your expectations?

7. To simulate the cold start problem, change the percentage of blanked out ratings on the testing data to 80%. Compare the results for each method using a training set of size 1000 and testing set of size 200. What conclusions can you draw from your results?

8. Change the percentage of blanked out ratings back to 30%. Then rerun the code for SGD with bias but instead of using f=20, use f=50. What happens to the error rate in SGD iterations and the error rate on the testing set? Can you explain your observations?

9. Is it easier to add new users or new items (with existing ratings) to a live recommendation system?

Table for Analysis						
set size	SGD Bias run time	SGD Bias RMS error	SVD++ run time	SVD++ RMS error	Neighborhood run time	Neighborhood RMS error
100						
400						
800						

5 Appendix

An Integrated Approach [1]

In this section, we introduce a method of collaborative filtering that combines both model and memory based approaches. As previously explained, matrix factorization models find a global structure in the data set while neighborhood methods consider local information by comparing similar users or items. By integrating both techniques into a single method, we can leverage both global and local phenomena in the data to further increase prediction accuracy. A model proposed by Yehuda Koren for the Netflix Prize Competition [1] combines SVD++ with an item based neighborhood method and can be learned by solving the following regularized least squares problem.

$$\begin{aligned}\hat{r}_{ui} = b_{ui} + q_i^T(p_u + \frac{1}{\sqrt{|N(u)|}}\sum_{j\epsilon N(u)} y_j) \\ + \frac{1}{\sqrt{|R^L(i;u)|}}\sum_{j\epsilon R^L(i;u)}(r_{uj} - b_{uj})w_{ij} \\ + \frac{1}{\sqrt{|N^L(i;u)|}}\sum_{j\epsilon N^L(i;u)} c_{ij}\end{aligned} \tag{10}$$

$$\begin{aligned}min_{p,q,b,y,w,c}\sum_{u,i}(r_{ui} - \hat{r}_{ui})^2 + \lambda(||p_u||^2 + ||q_i||^2 + b_u^2 + b_i^2 \\ + \sum_{j\epsilon N(u)} ||y_j||^2 + \sum_{j\epsilon R^L(i;u)} w_{ij}^2 + \sum_{j\epsilon N^L(i;u)} c_{ij}^2)\end{aligned} \tag{11}$$

Where

- $R^L(i;u)$ represents the set of L items that are the most similar to item i, where user u has previously rated i.
- $N^L(i;u)$ represents the set of L items that are the most similar to item i where user u has previously provided an implicit preference for i.
- b_{uj} represents the baseline estimate for user u's rating of item j.

- $w_{ij} \in \mathbb{R}$ and $c_{ij} \in \mathbb{R}$ represent the individual weights from item j to item i for explicit and implicit feedback.

Notice that the first half of the equation for $\hat{r}_{ui}$ is the SVD++ model presented in section 2.4. The second half is a different flavor of the neighborhood model, which uses optimization to learn the similarity values (ie. the weights w_{ij}) rather than obtaining them directly through comparison. While a traditional item based neighborhood method seeks to calculate similarity values for the most similar items to the ones that the target user has rated, the learned w_{ij}'s in this method remain independent of any user-item interaction and operate on a more global scale.

In this sense, it is useful to adopt a new view of the weights as offsets to the baseline estimate b_{ui} of a user's rating for a particular item. For instance, if a user's rating for a similar item j deviates from his expected rating on that item (ie. $|r_{uj} - b_{uj}|$ is non-zero), then we add a weighted portion of $(r_{uj} - b_{uj})$ to the estimate for that user's rating for the target item i. Similarly, if the user's rating for j does not deviate from expectation (ie. $|r_{uj} - b_{uj}|$ is close to zero) or a preference for j does not highly indicate a preference for i (ie. w_{ij} is small) then our estimate for the user's rating on i will remain relatively the same. Defining the model in this way also allows us to incorporate implicit data into the neighborhood model as well and we can adopt a similar interpretation of the weights on the implicit data, c_{ij} [1].

The integrated model provides the global perspective of SVD++ while offering adjustments to estimated ratings using a neighborhood approach. For example, a system implementing pure SVD++ might estimate that Bob would give a rating of 4.5 to Star Wars, based solely upon the learned latent factors that it assigned to Bob and Star Wars. However, such a system would be oblivious to important deviations like Bob's 1.5 rating of Star Trek, which has a high similarity to Star Wars. Through the added neighborhood model, an integrated approach would account for these idiosyncrasies in user behavior and appropriately adjust the estimate for Bob's rating of Star Wars to 4.

While the integrated model may provide noticeable performance gains, we will not implement this method as tuning the parameters and learning the model have significantly longer computation times than the other methods we covered. However, students may differentiate the objective function to obtain update rules for stochastic gradient descent and extend the SVD++ code if they wish to create an implementation.

6 Resources

6.1 Original Data Set

MovieLens dataset: http://grouplens.org/datasets/movielens/

References

[1] Yehuda Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD Inter-*

national Conference on Knowledge Discovery and Data Mining, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.

[2] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009.