Recommendation Systems Using Collaborative Filtering

Kevin Zhang, Qinyuan Sun, and Austin Mease

December 7, 2015

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 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, exemplified by SVD++, 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. Furthermore, 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 also suffer from 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. 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

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

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 considers global average rating (μ), item bias (b_i), user bias (b_u), and user-item interaction. To incorporate bias, we can modify the previous objective function and prediction equation.

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u \qquad b_u \in \mathbb{R}^n, b_i \in \mathbb{R}^m \tag{4}$$

$$min_{p,q,b} \sum_{u,i} (r_{ui} - \hat{r}_{ui})^2 + \lambda(||p_u||^2 + ||q_i||^2 + b_u^2 + b_i^2) \tag{5}$$

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

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. This model is commonly called SVD++ [1].

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T(p_u + \frac{1}{\sqrt{|N(u)|}} \sum_{j \epsilon N(u)} y_j) \qquad y_j \in \mathbb{R}^f \tag{6}$$

$$min_{p,q,b} \sum_{u,i} (r_{ui} - \hat{r}_{ui})^2 + \lambda(||p_u||^2 + ||q_i||^2 + b_u^2 + b_i^2 + \sum_{j \epsilon N(u)} ||y_j||^2) \tag{7}$$

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.

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. 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 5 (HINT: p_u, q_i should be the same).
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 7 (HINT: p_u, b_i, b_u should be the same).

4 Lab

In this lab, we will use a subset of the MovieLens data set with 6040 users (n), 3952 movies (m), and approximately 1 million 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 in the appendix, 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 20 (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 5 (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 10 rows, 100 rows, and 1000 rows. Use a testing set of size 200 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 1000, *main.m* will take about 20-30 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. Rerun the code for SGD with bias but instead of using f=10, 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. If the run time for the SVD++ depends on only one dimension, is it easier to add new users or new items in the real world? What advantages would having the reverse setup have and what would be different in the interpretation of the result?

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
10						
100						
1000						

5 Code

main.m

```
load ratings.mat

[num_usr, num_itm] = size(ratings);

% Number of users for testing
n_test = 200;

% Number of users for training
% The line below will use the remaining full dataset for training
% n_train = num_usr - n_test;
n_train = 1000;

% Matrix factorization and SVD++ parameters
f = 20;
lambda = 0.01;
step_size = 0.005;

% Neighborhood method parameters
K = 5;
L = 20;

% split ratings matrix into testing and training datasets
train_idx = randperm(num_usr, n_train);
test_idxidx = randperm(n_test);
test_idxset = setdiff([1:num_usr],train_idx);
test_idx = test_idxset(test_idxidx(1:n_test));

test_data = ratings(test_idx, :);
train_data = ratings(train_idx, :);

% To test predictions, blank out 'perc_blank' percent of the ratings in
% the testing data set. The remaining '1 - perc_blank' percent of
% the ratings will be included in the full training dataset to establish
% a baseline for the testing users.
perc_blank = 0.3;
test_data_blanked = zeros(n_test, num_itm);
% the result indicies of the randperm for each user
user2itmratedidx = zeros(n_test, num_itm);
for i = 1:n_test
    u = test_idx(i);
    num_itm_rated = itm_rated_4_user(u, 1);
    itm_rated = itm_rated_4_user(u, 2:num_itm_rated+1);

    itm_rated_idx = randperm(num_itm_rated);
    new_len = floor((1.0-perc_blank) * num_itm_rated);
    user2itmratedidx(i, 1:num_itm_rated) = itm_rated_idx;
    itm_rated_idx = itm_rated_idx(1:new_len);

    test_data_blanked(i, itm_rated(itm_rated_idx)) = ratings(u, itm_rated(itm_rated_idx));
end

% neighborhood method
[id_sim_usr, coeff_sim_usr] = neighborhood(train_data, test_data_blanked, K, L);

% ---------------------------------------------------------------------------
% TODO:
% 1. Using the lookup tables returned from the neighborhood function, find
% the item rating predictions for each user in the testing set.
%
% 2. Place your results in the neighbor_predictions matrix.
%
% Predictions are the normalized weighted sum of the ratings of the L most
% similar users to a target user in the testing data. See the overview of
% the lab for the formula.
% ---------------------------------------------------------------------------
neighbor_predictions = zeros(n_test,num_itm);
% PLACE YOUR CODE HERE


% new training set with the blanked out users
augment_train_data = [test_data_blanked; train_data];

% find matrix factorization and biases using sgd
[q, p, bu, bi, mu] = sgd_bias(augment_train_data, f, lambda, step_size);

% create lookup table of item ids that a user has rated for augmented
% training set for SVD++ speedup
[~,max_itms_rated] = size(itm_rated_4_user);
lookup_tbl = [zeros(n_test, max_itms_rated); itm_rated_4_user(train_idx, :)];
for i=1:n_test
    [row, col] = find(test_data_blanked(i, :));
     col_size = nnz(test_data_blanked(i, :));
     lookup_tbl(i, 1:(col_size+1)) = [col_size, col];
end
% SVD++
[qpp, ppp, bupp, bipp, mupp, y] = ...
    svdpp(augment train data, lookup tbl, f, lambda, step size);


% Compute predictions and errors for each method
total_err_bias = 0;
total_err_neighbor = 0;
total_err_svdpp = 0;
sum_blanked = 0;
sum_neighbor = 0;
for i = 1:n_test % i is the user index in the augmented training matrix

    u = test_idx(i); % u is the user index in the original ratings matrix
    num_itm_rated = itm_rated_4_user(u, 1);
    itm_rated = itm_rated_4_user(u, 2:num_itm_rated+1);
    itm_rated_idx = user2itmratedidx(i, 1:num_itm_rated);
    % find indicies of blanked out ratings
    new_len = floor((1.0-perc_blank) * num_itm_rated);

    % the last 'perc_blank' percent of indexes in itm_rated_idx were
    % blanked out
    blanked_idx = itm_rated(itm_rated_idx(new_len+1:end));
    blanked_len = length(blanked_idx);

    % track number of predictions for final error calculation
    sum_blanked = sum_blanked + blanked_len;

    actual = ratings(u, blanked_idx);

    % evaluate neighborhood method
    predict_neighbor = neighbor_predictions(i, blanked_idx);
    nonzero_idx = predict_neighbor > 0;
    % only count the ratings that NH method could predict
    % ie. don't count ratings that are zero
    err_neighbor = nansum((predict_neighbor(nonzero_idx) - actual(nonzero_idx)).^2);
    total_err_neighbor = total_err_neighbor + err_neighbor;
    % number of predictions for neighborhood must be kept separately
    sum_neighbor = sum_neighbor + length(nonzero_idx);

    % evaluate sgd bias
    bias_sgd = repmat(mu, 1, blanked_len) + repmat(bu(i), 1, blanked_len) + bi(blanked_idx)';
    predict_bias = bias_sgd + (q(:, blanked_idx)'*p(:, i))';
    err_bias = sum((predict_bias - actual).^2);
    total_err_bias = total_err_bias + err_bias;

    % evaluate SVD++
    num_filled_itm = lookup_tbl(i,1);
    filled_itm_idx = lookup_tbl(i,2:num_filled_itm+1);

    Nu = num_filled_itm^-0.5;
    normalized_yjsum = Nu * sum(y(:, filled_itm_idx), 2);
    bias pp = repmat(mu pp, 1, blanked len) + ...
        repmat(bu pp(i), 1, blanked len) + bi pp(blanked idx)';
    predict_svdpp = bias_pp + (q_pp(:, blanked_idx)'*(p_pp(:, i) + normalized_yjsum))';
    err_svdpp = sum((predict_svdpp - actual).^2);
    total_err_svdpp = total_err_svdpp + err_svdpp;
end
% Evaluation based on root mean square
sqrt(total_err_neighbor/sum_neighbor);
sqrt(total_err_bias/sum_blanked);
sqrt(total_err_svdpp/sum_blanked);
```

neighborhood.m

```
function [ id_most_sim_usr, coeff_most_sim_usr ] = neighborhood( train_data, test_data, K, L )
    % neighborhood
    %
    % Description:
    %   Neighborhood method for memory based collaborative filtering. Finds
    %   the L most similar users from the training data for each user in
    %   the test data. Users from the training data are only considered if
    %   they have K item ratings in common with a target user from the test
    %   data. Uses Pearson Correlation measure to compute similarity.
    %
    % Parameters:
    %   train_data: number of training users x m matrix of ratings
    %   test_data:  number of testing users x m matrix of ratings
    %   K:          minimum common item ratings between a training user
    %               and target user
    %   L:          minimum number of users needed for the weighted average
    %
    % Output:
    %   id_most_sim_usr:    user ids for the L most similar users from the
    %                       training set to the target users from testing.
    %                       number of test users x L matrix
    %   coeff_most_sim_usr: similarity measures for each of the L most
    %                       similar users. number of test users x L matrix

    [n_train, ~] = size(train_data);
    [n_test, ~] = size(test_data);

    id_most_sim_usr = zeros(n_test,L); % ids for most L similar users
    coeff_most_sim_usr = zeros(n_test,L);% Pearson correlation for each L similar users
    for i  = 1:n_test  % i is the target user id from testing
        usr_test = test_data(i, :);
        % item ids for the items that the test user has rated
        [~, test_obj_id] = find(usr_test);

        tic % time each iteration
        similarity  = zeros(1,n_train);
        for j = 1:n_train % j is the user id from training
            usr_train = train_data(j, :);
            % item ids for the items that the training user has rated
            [~, train_obj_id] = find(usr_train);

            % -------------------------------------------------------------
            % TODO:
            % 1. Use intersect() function to find the items that both the
            % test and training user have rated. You can find the item ids
            % that the test user has rated in the test_obj_id vector. You
            % can find the item ids that the training user has rated in the
            % train_obj_id vector.
            %
            % 2. If the two users have less than k ratings in common,
            % then do not consider the training user in the similarity
            % calculation.
            %
            % 3. Otherwise, compute the Pearson correlation coefficient of
            % both user's common ratings using the function corr()
            %
            % 4. Place the results into similarity(j)
            % -------------------------------------------------------------
            % PLACE YOUR CODE HERE
        end

        similarity(isnan(similarity)) = 0; % corr can return NaN sometimes
        % Find the top L similar users and their similarity coefficients
        [sim_sorted, sim_usr_ids] = sort(similarity,'descend');
        id_most_sim_usr(i,:) = sim_usr_ids(1:L); % user index into training data
        coeff_most_sim_usr(i,:) = sim_sorted(1:L); % similarity values for each user
        toc
    end
end
```

sgd_bias.m

```
function [ q, p, bu, bi, mu ] = sgd_bias( ratings, f, lambda, step_size )
    % sgd_bias
    %
    % Description:
```

```
%   Matrix Factorization for model based collaborative filtering.
%   Uses stochastic gradient descent to create a mapping between
%   users/items and a latent feature space of f features. Also
%   computes user and item bias values.
%
% Parameters:
%   ratings:    a user (n) x item (m) matrix of ratings
%   f:          number of latent factors
%   lambda:     regularization weight
%   step_size:  step size for gradient descent
%
% Output:
%   q:          a f x m item matrix mapped to the latent feature space
%   p:          a f x n user matrix mapped to the latent feature space
%   bu:         a 1 x n user bias matrix
%   bi:         a 1 x m item bias matrix
%   mu:         mean rating in the ratings matrix

[num_usr, num_obj] = size(ratings);
num_ratings = nnz(ratings);

[usr_idx, obj_idx, vals] = find(ratings);

threshold = 1.0e-3;
% 30 SGD passes over the data set
maxiter = 30*num_ratings;

% Initialize with random values from Gaussian distribution
q = randn(f,num_obj);
p = randn(f,num_usr);
bu = randn(num_usr, 1);
bi = randn(num_obj, 1);
mu=mean(vals);

total_err = inf;
for iter = 0:maxiter
    % t is the index into pick_idx
    t = mod(iter,num_ratings)+1;
    if t == 1
        % Random permutation of ratings
        pick_idx = randperm(num_ratings);

        avg_err = total_err/num_ratings;
        if avg_err < threshold
            break;
        end

        disp(['pass is ' num2str(floor(iter/num_ratings))]);
        disp(['average err is ' num2str(avg_err)]);
        total_err = 0;
    end
    % pick a random rating for descent
    idx = pick_idx(t);

    u = usr_idx(idx); % u is for user
    i = obj_idx(idx); % i is for item

    r_true = vals(idx); % true rating for user u and item i

    % ----------------------------------------------------------------
    % TODO:
    % Compute the prediction rhat for a particular user u and item i.
```

```
            % Refer to the lab overview for the exact formula.
            % -----------------------------------------------------------------
            rhat = ;

            err = r_true - rhat;
            total_err = total_err+(err^2);

            % -----------------------------------------------------------------
            % TODO:
            % Update q_i, p_u, b_u, and b_i using the rules you derived in the
            % lab warm-up
            % -----------------------------------------------------------------
            q(:,i) = ;
            p(:,u) = ;
            bu(u) = ;
            bi(i) = ;
        end
    end
```

svdpp.m

```
function [ q, p, bu, bi, mu, y ] = svdpp( ratings, itm_rated_4_user, f, lambda, step_size )
    % svdpp
    %
    % Description:
    %   SVD++ algorithm for model based collaborative filtering.
    %   Uses stochastic gradient descent to create a mapping between
    %   users/items and a latent feature space of f features. Also
    %   computes user and item bias values.
    %
    % Parameters:
    %   ratings:            a user (n) x item (m) matrix of ratings
    %   itm_rated_4_user:   a n x m lookup table to get the ids of items that
    %                       a user has provided ratings for. The first cell
    %                       in each row is the number of non-zero entries
    %                       in the row. To get the list of item ids, use
    %                       the code snippet below where u is the user
    %                       id.
    %
    %                       num_itm_rated = itm_rated_4_user(u, 1);
    %                       itm_rated = itm_rated_4_user(u, 2:num_itm_rated+1);
    %
    %   f:                  number of latent factors
    %   lambda:             regularization weight
    %   step_size:          step size for gradient descent
    %
    % Output:
    %   q:          a f x m item matrix mapped to the latent feature space
    %   p:          a f x n user matrix mapped to the latent feature space
    %   bu:         a 1 x n user bias matrix
    %   bi:         a 1 x m item bias matrix
    %   mu:         mean rating in the ratings matrix
    %   y:          a f x m matrix of implicit data values for each latent
    %               feature f of an item
    [num_usr, num_itm] = size(ratings);
    num_ratings = nnz(ratings);

    [usr_idx, itm_idx, vals] = find(ratings);

    threshold = 1.0e-3;
    maxiter = 30*num_ratings;
```

```
% Initialize with random values from Gaussian distribution
q = randn(f,num_itm);
p = randn(f,num_usr);
bu = randn(num_usr, 1);
bi = randn(num_itm, 1);
y = randn(f,num_itm);
mu=mean(vals);

total_err = inf;

tic
for iter = 0:maxiter
    % t is the index into pick_idx
    t = mod(iter,num_ratings)+1;
    if t == 1
        toc
        % Random permutation of ratings
        pick_idx = randperm(num_ratings);

        avg_err = total_err/num_ratings;
        if avg_err < threshold
            break;
        end

        disp(['pass is ' num2str(floor(iter/num_ratings))]);
        disp(['average err is ' num2str(avg_err)]);
        total_err = 0;
        tic
    end
    % pick a random rating for descent
    idx = pick_idx(t);
    u = usr_idx(idx); % u is for usr
    i = itm_idx(idx); % i is for item

    r_true = vals(idx); % true rating for usr u and item i

    % For speed, get items that a user has rated from lookup table
    num_itm_rated = itm_rated_4_user(u, 1);
    itm_rated = itm_rated_4_user(u, 2:num_itm_rated+1);

    % -----------------------------------------------------------------
    % TODO:
    % Compute the prediction rhat for a particular user u and item i.
    % Refer to the lab overview for the exact formula.
    % For this you will need:
    % 1. |N(u)| where N(u) is the set of items that user u has rated
    % 2. The sum of the y_j's where j is an id of an item in N(u).
    %
    % The vector itm_rated contains the ids of items that a user has
    % rated (ie. the j's in N(u)) and the variable num_itm_rated
    % contains |N(u)|.
    % -----------------------------------------------------------------
    rhat = ;

    err = r_true - rhat;
    total_err = total_err+(err^2);

    % -----------------------------------------------------------------
    % TODO:
    % Update q_i, p_u, b_u, and b_i using the rules you derived in the
    % lab warm-up.
    % -----------------------------------------------------------------
```

```
        q(:,i) = ;
        p(:,u) = ;
        bu(u) = ;
        bi(i) = ;

        % -----------------------------------------------------------------
        % TODO:
        % For all items that a user has rated, update the corresponding
        % y_j vectors.
        % -----------------------------------------------------------------
        y_itm_rated = y(:, itm_rated); %cache for speed
        y(:,itm_rated) = ;
    end

end
```

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 International 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.