CS 5683: Big Data Analytics

Project-4: Recommender Systems

Total Points: 100 (15% toward final)

Due date: Nov 12, 2020 at 11:59pm

In this project, we will experiment with two modes of factorization models for movie recommendations. In particular, we will implement factorization models that optimizes (a) interpolation weight matrix $\boldsymbol{w}$ in the item-item collaborative filtering, and (b) singular matrices $\boldsymbol{P}$ *and* $\boldsymbol{Q}$ in the latent factor model with stochastic gradient descent. We will evaluate the performance of models with Root Mean Squared Error (RMSE) and report them to complete this project. *Although implementation of these algorithms is little easier compared to other project, execution may take significant time.* This is a group project. Groups can be of maximum size 2.

Dataset: We will use the openly available movie ratings data in this project (Source: https://grouplens.org/datasets/movielens/100k/). We have processed the data and made training and test data samples available. Both training and test data have columns: '*user_id*', '*item_id*', '*rating*', and '*movie_name*'. *The 'rating' feature in the test_dataset should be used only for model evaluation*. In other words, you should use 'rating' feature neither for similarity measure in Item-Item Collaborative Filtering nor for training phase in Latent Factor Recommender system.

Consider the training dataset as a matrix R of ratings. The element R_{xi} of this matrix corresponds to the rating given by user x to movie i. The size of R is $m \times n$, where m is the number of users, and n is the number of movies. Most of the elements of the matrix R are unknown because each user can only rate a few movies.

(1) Item-item Collaborative Filtering with Interpolation Weight:

As discussed in the class a simple item-item collaborative filtering with weighted average has multiple pitfalls. We will replace the weighted average of the item-item collaborative filtering with the interpolation weight. The task of this section the project is to find optimal values of interpolation weight with *Stochastic Gradient Descent*. We define a simple error function given in **Eq. 1** to optimize the values of interpolation weight **w**:

$$J(w) = \sum_{x,i} \left(\left[b_{xi} + \sum_{j \in N(i;x)} w_{ij} \left(r_{xj} - b_{xj} \right) \right] - r_{xi} \right)^2 \qquad \textit{Eq. 1}$$

where N is a set of movies that are "*similar*" to the movie $\boldsymbol{i}$ and rated by the user $\boldsymbol{x}$, r_{xi} is the rating of movie 'i' given by user 'x', and $b_{xi} = \mu + R_x^* + R_i^*$ given that μ = Overall mean movie rating, R_x^* = Average rating of user **x** - μ, and R_i^* = Average rating of item **i** - μ.

We measure similarity of items ***i*** and ***j*** with cosine similarity equation given in **Eq. 2**.

$$sim(i,j) = \frac{\sum_{y}^{U} r_{yi} \cdot r_{yj}}{\sqrt{\sum_{y}^{U} r_{yi}^2}\sqrt{\sum_{y}^{U} r_{yj}^2}} \quad \textit{Eq. 2}$$

where *U* is the set of all users who have rated movies ***i*** and ***j***

We optimize the ***w*** using Stochastic Gradient Descent with $w \leftarrow w - \eta \nabla_w J$, where η is the learning rate and $\nabla_w J$ is the partial derivative of *J(w)* w.r.t. ***w*** as given in **Eq. 3**:

$$\nabla_w J = \sum_{x,i}\left(\left[b_{xi} + \sum_{j \in N(i;x)} w_{ij}\left(r_{xj} - b_{xj}\right)\right] - r_{xi}\right)\left(r_{xj} - b_{xj}\right) \quad \textit{Eq. 3}$$

NOTE-1: You do not need to calculate $sim(i,j)$ for all movie pairs in this project. You only need to find similarity of movies that were rated by user ***x***. You can pre-compute this similarity by consolidating a list of movies from the test data

NOTE-2: Consider movies that are at least 50% similar for the set ***N*** in **Eq. 1**. You can fine tune the parameter ***N*** to improve the model performance

Initialize ***w*** with random values [0,1], number of iterations to 40, and experiment the values for η. You can optimize all parameters as much as possible until you reach the steady state for J(w). **Plot the value of the objective function *J* (given in Eq. 1) on the training set as a function of the number of iterations.**

(2) Latent Factor Model:

Methods like *Collaborative Filtering* would require naïve assumption like *Similarity Measure* to predict recommendations. In this section of the project, we will utilize *Stochastic Gradient Descent* algorithm to build a Latent Factor Recommendation system.

Our goal with Latent Factor model is to find two matrices *P* and *Q*, such that $R \approx PQ^T$. The dimensions of *P* are $m \times k$, and the dimensions of *Q* are $n \times k$. k is a parameter of the algorithm.

We define the error function ***E*** as

$$E = \left(\sum_{(x,i) \in ratings} (r_{xi} - p_x . q_i^T)^2\right) + \lambda\left[\sum_x ||p_x||_2^2 + \sum_i ||q_i||_2^2\right] \quad \textit{Eq. 4}$$

The $\sum_{(x,i) \in ratings}$ means that we sum only on the pairs (user, movie) for which the user has rated the item, *i.e.* the *(x,i)* entry of the matrix *R* is known. p_x denotes the x^{th} row of the matrix *P* (corresponding to a user), and q_i denotes the i^{th} row of the matrix *Q* (corresponding to a movie). p_x and q_i are both row vectors of size *k*. λ is the regularization parameter. $||\,.\,||_2$ is the L_2 norm and $||\,.\,||_2^2$ is the square of the L_2 norn, *i.e.*, it is the sum of square of elements of the given vector (*For example,* p_x and q_i).

We optimize the matrices P and Q using Stochastic Gradient Descent with $P_x \leftarrow P_x - \eta \nabla P_x(R_{xi})$ and $Q_i \leftarrow Q_i - \eta \nabla Q_i(R_{xi})$ respectively, where η is the learning rate and ∇P_x and ∇Q_i are partial derivatives of E w.r.t. P_x and Q_i respectively as given below:

$$\nabla P_x(R_{xi}) = \frac{\partial E}{\partial P_x} = -(R_{xi} - p_x.q_i^T)q_i + \lambda p_x$$

$$\nabla Q_i(R_{xi}) = \frac{\partial E}{\partial Q_i} = -(R_{xi} - p_x.q_i^T)p_x + \lambda q_i$$

Implement Stochastic Gradient Descent and optimize matrices P and Q. To emphasize, you are not allowed to store the matrix R in memory (as we did for Collaborative Filtering). You have to read each element R_{xi} one at a time from disk and apply your update equations (to each element) each iteration. Each iteration of the algorithm will read the whole file.

Choose $k=25$, $\lambda=0.1$, $\mu=0.1$, and number of algorithm iterations=*40*. You can optimize all parameters as much as possible until you reach the steady state for $\boldsymbol{E}$. You do not need to change other values unless you want to have bonus points for the project. **Plot the value of the objective function $\boldsymbol{E}$ (given in Eq. 4) on the training set as a function of the number of iterations.**

Implementation Tips:

1. *Initialization of P and Q:* Initialize P and Q matrices in such a way that $p_x.q_i^T \in [0,5]$. To achieve this, initialize all elements of P and Q to random values in $[0,\sqrt{5/k}]$
2. *Update equations*: In each iteration, we update p_x with q_i and q_i with p_x. Compute the new values of p_x and q_i using old values and then update vectors p_x and q_i
3. *Compute E* at the end of a full iteration of training. Computing E in pieces during the iteration is incorrect since P and Q are still being updated

Model Evaluation:

Implement the following steps to do evaluate both models:

1. Read the test dataset and hide the '*rating*' column
2. Predict the '*rating*' value using models discussed above
3. Compare '*actual_rating*' and '*predicted_rating*' with Root Mean Squared Error (RMSE). Code snippet to calculate RMSE is given below:

```
from sklearn.metrics import mean_squared_error

from math import sqrt

def RMSE(y_actual, y_predicted):
rms = sqrt(mean_squared_error(y_actual, y_predicted))
return round(rms,4)
```

Bonus task: Refine your best model either by hyper-parameter tuning or by extending the model to one of the advance models discussed in the lecture. Report the improved RMSE score in the leaderboard. Top 5 groups will receive some bonus points towards the final. Give a very good documentation for bonus tasks. **NOTE:** *If you are using a different error function (E), report its derivation also.* Check *Project-4_Guidelines* slides on how to use the shared leaderboard.

Submission requirements:

1. Students can utilize Python programming. PySpark implementation will be considered for bonus points tie breaker
2. Programs should be well documented – the grader should understand program modules clearly with your documentation
3. Submit only the following two files: **CF.py** (collaborative filtering) and **LF.py** (latent factor model) [YES, you can submit notebook (.ipynb) files also]. Include team member names in both files
4. What to output in each task?
    a. **CF.py:** Plot of J(w) as a function of iterations (plot should have clear naming conventions for x-axis, y-axis, and title) and RMSE for the test set.
    b. **LF.py:** Plot of E as a function of iterations (plot should have clear naming conventions for x-axis, y-axis, and title) and RMSE for the test set. Finally, compare and contrast results from **CF.py** and **LF.py**
    c. If you are trying for bonus points, you can create another .py file and give results. Do not overwrite **CF.py** or **LF.py**
5. Include the plots, result comparisons, and their reasoning in appropriate programs
6. [IMPORTANT] Since this is a group project, give the participation report (which team member contributed to which module in both programs) at the end of **LF.py**

Grading Rubric:

****Students who choose to work independently can ignore the* ***Collaborative Filtering***

1. Collaborative Filtering – Total: 40 points
    a. SGD: 10 points
    b. Compute J(w): 10 points
    c. Plot: 10 points
    d. Results: 10 points
2. Latent Factor Model: 50 points
    a. SGD: 20 points
    b. Compute E: 10 points
    c. Plot: 10 points
    d. Results: 10 points
3. Submission requirements: (10 points)
    a. Documentation and program file organization: 5 points
    b. Team work: 5 points