CS 5683: Big Data Analytics

Project-4: Recommender Systems

Total Points: 100 (15% toward final)

Due date: Nov 1, 2020 at 11:59pm

In this project, we will experiment with two algorithms for movie recommendation. In particular, we will explore (a) the simple item-item collaborative filtering, and (b) a latent factor model with stochastic gradient descent, discussed in our class. 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 x 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.

Item-item Collaborative Filtering:

Unlike latent factor models, we do not require to optimize any error functions in item-item collaborative filtering. Instead, we will read the test data (of course, without '*rating*') and give predictions directly. Our task includes (i) to set the similarity metric, (ii) to calculate similarities of multiple movies, and (iii) to predict the rating of the given movie ***i*** for given user ***x***. Predictions using the item-item collaborative filtering can be calculated using **Eq. 1**:

$$R_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} * R_{xj}}{\sum_{j \in N(i;x)} s_{ij}} \qquad \textit{Eq. 1}$$

where N is a set of movies that are similar to the movie ***i*** and also rated by the user ***x***, s_{ij} is the similarity score of items ***i*** and ***j***. We will experiment with the following two types of similarity measures to calculate similarity score s_{ij} of two movies ***i*** and ***j***.

1. *Cosine similarity*:

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

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

2. *Adjusted cosine similarity*: As discussed in class, cosine similarity works, but it does not consider different rating schemes of different users. To avoid this scenario, we modify the cosine similarity score with **Eq. 3**:

$$sim(i,j) = \frac{\sum_y^U \left(R_{yi} - R_y^*\right)\left(R_{yj} - R_y^*\right)}{\sqrt{\sum_y^U \left(R_{yi} - R_y^*\right)^2}\sqrt{\sum_y^U \left(r_{yj} - R_y^*\right)^2}} \quad \textit{Eq. 3}$$

   where R_y^* is the average rating of user ***y***.

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*** for both cosine similarity and adjusted cosine similarity to improve the model performance

Latent Factor Model[1]:

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).

[1] Idea motivated from Stanford Mining Massive Datasets course assignment

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, λ=0.1, μ=0.1, and number of algorithm iterations=40. Optimize the number of iterations as much as possible until you reach the steady state for ***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 *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 evaluation for all 3 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 models 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:** RMSE for the test data using both Cosine similarity and Adjusted cosine similarity. Compare and contrast results based on 2 similarity metrics
    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:

1. Collaborative Filtering – Total: 35 points
    a. Cosine similarity: 15 points
    b. Adjusted cosine similarity: 10 points
    c. Results: 10 points
2. Latent Factor Model: 40 points
    a. SGD: 10 points
    b. Compute E: 10 points
    c. Plot: 10 points
    d. Results: 10 points
3. Compare and contrast models: (5 points)
4. Submission requirements: (20 points)
    a. Documentation and program file organization: 10 points
    b. Team work: 10 points