INFORMATION RETRIEVAL

CS F469

Recommender Systems

Abhishek Bapna 2018A7PS0184H

Ashna Swaika 2018A7PS0027H

Hardik Parnami 2018A7PS0062H

**Recommender Systems**

This application of recommender systems predicts ratings a user might give to a particular movie based on the previous history of ratings given by various users for various movies, and thus it can recommend new movies to the user. Several techniques were used to achieve this:

1. Collaborative filtering
2. SVD
3. CUR

along with respective enhancements. For each method used, RMSE ,Spearman Rank Correlation and Top K Precision were calculated using a validation/test set of ratings.

**90% energy rule:** Retain enough singular values to make up 90% of the energy. That is, the sum of squares of the retained singular values should be at least 90% of the sum of squares of all the singular values.

**Dataset :-** [MovieLens dataset](https://grouplens.org/datasets/movielens/)

In order to construct our Utility Matrix along with a test set of ratings,a dataset of about 1,00,000 ratings from roughly 950 users on 1682 movies was taken from MovieLens. Large percentage of the ratings were used to construct our Utility Matrix which then was used to train our models and help in predictions of future ratings. The remaining ratings were used as a test set to evaluate the performance of our models.

All ratings are in the following format: UserID::MovieID::Rating::Timestamp

User information is in the following format: UserID::Gender::Age::Occupation::Zip-code

Language Used : python3

Libraries used

* + Numpy
  + Pandas
  + Scipy

| **Recommender System** | **Root Mean Square Error (RMSE)** | **Precision on top K** | **Spearman Rank Correlation** | **Time taken for prediction** |
| --- | --- | --- | --- | --- |
| **Collaborative** | 1.3148 | 0.18 | 1.0 | 33.0174 |
| **Collaborative along with Baseline approach** | 1.1871 | 0.1998 | 1.0 | 24.8590 |
| **SVD** | 1.4466 | 0.0 | 0.9998 | 42.3082 |
| **SVD (with 90% retained energy)** | 1.3137 | 0.3160 | 0.9998 | 42.6213 |
| **CUR** | 140.355 | 0.3160 | 0.9998 | 3.0807 |
| **CUR (with 90% retained energy)** | 0.9313 | 0.3160 | 0.9998 | 8.62909 |

**RESULTS**

1. In terms of computational time SVD takes significantly higher time in comparison to CUR and collaborative filtering. Although we also see that it's at par with the others in terms of accuracy, it does not change too much when the amount of energy preserved is 90%.
2. Collaborative filtering is the best for recommending top K movies to the users based on precision on top K.
3. Spearman correlation does not vary very much throughout all three algorithms.
4. Runtime order shows CUR < Collaborative Filtering < SVD

**Collaborative Filtering & Collaborative filtering with baseline approach**

The process of identifying similar users and recommending what similar users like

is called collaborative filtering.Item-Item collaborative filtering was used as similarity between items is much more useful than similarity between users.

The model does not need data apart from other users since recommendations are user specific but poses a challenge for new users or new types of items

By using Baseline Approach we manage to correct the negative predictions by averaging the distance from the baseline predictor. It also helps extend collaborative filtering to large user bases that is in this case handles strict,lenient raters efficiently

IMPLEMENTATION

We used collaborate.py with the estimate(self,user,item,k=2,baseLine=False) fn estimates the rating for a given user(int) and item(int) and k nearest neighbours(int) and baseLine is by default set to false.

Fill function, fills the gaps in the matrix based on the estimate function created. Pearson\_similarity is calculated in similarities.py to calculate the similarity in collaborative filtering.

The collaborative filtering with baseline approach just makes the baseline boolean true.

**Singular Value Decomposition & SVD with 90% retained energy**

SVD is calculated for any given **M**x**N** matrix by decomposing that matrix into three component matrices defined as follows:

* **Matrix U:** An **M**x**r** column orthonormal matrix where r is the rank of the original matrix.
* **Matrix ∑:** An **r**x**r** diagonal matrix containing the singular values of the original matrix.
* **Matrix V:** An **r**x**N** column orthonormal matrix where r is the rank of the original matrix.

The original matrix can then be approximated by calculating the product of these matrices as follows:

Approximation of original matrix = **U\*∑\*VT**

SVD simplifies data,removes noise to improve algorithm performance but the transformed data may be difficult to understand

**Dimensionality Reduction:** If the rank **r** of the matrix is substantially smaller than **N**, we can intuitively see that the combined sizes of the component matrices will be smaller than that of the original matrix, thereby providing a valuable reduction in dimensionality for computational purposes.

IMPLEMENTATION

The svd function of SVDAlgorithm() is applied the input matrix and decomposed into U\*sigma\*V.

A dimension reduction parameter is given which is normally set to one but is set to 0.9 when the energy to be preserved is said to be 0.9. The dimension reduction of sigma is also performed in this function.

Another function called eigen\_decomposition is called in order to return eigen values of Input matrix M. It returns a sorted list of eigenvalues(list) and eigenvectors(numpy matrix).

**CUR & CUR with 90% retained energy**

Utility matrix is divided into three matrices C, U, R:

* **Matrix C**: An **M**x**r** randomly chosen matrix where r is the no. of columns. Usually a bit more than the number of rows taken for SVD
* **Matrix U**: An **r**x**r** diagonal matrix containing the singular values of the original matrix.
* **Matrix R**: An **r**x**N** randomly chosen matrix where r is the no. of rows Usually a bit more than the number of rows taken for SVD.

The original matrix can then be approximated by calculating the product of these matrices as follows:

Approximation of original matrix = **C\*U\*R**

The run time,complexity is very less compared to other recommender systems but the precision value also decreases drastically

IMPLEMENTATION

cur\_algorithm is a function in the cur folder which implements cur.It returns the final matrix and takes input matrix, number of columns - c (int), number of rows-r (int) and a boolean repeat which is initially set to true.

Selection function which selects the rows and columns takes the Input matrix, sum of squares of elements in the input matrix, number of columns to select and a boolean repeat\_allowed.

We use SVDAlgorithm() to compute the U matrix.