*Implementing and comparing various techniques for building a Recommender System*

**Design Document**

***29-October-2018***

Index

**1.Overview**

**2**.**Dataset**

**3.Programming language**

**4.Libraries used**

**5.Input Format**

**6.Data Structures**

**7.Collaborative Filtering**

**8.Singular Value Decomposition**

**9.CUR Decompostion**

**10.Programming Details**

**11.Execution Steps**

**12.Results**

1. **Overview:**

The purpose of this project is to implement and compare different implementations of recommender systems. The considered options are –

Collaborative filtering, singular value decomposition, cur decomposition and all three with 90% energy savings

**2. Dataset**

Movie lens data set for movie reviews was used. The data set is pre-processed to extract relevant information

Dataset URL :- https://grouplens.org/datasets/movielens/

1. **Programming language**: Python3.6.5
2. **Libraries used**:

* numpy – for managing the arrays
* numpy.linalg – for eigen value decomposition
* Pickle – Serialization and deserialization of python objects
* heapq– For sorting

1. **Input Format:**

The input to the program is a “**ratings.dat**” file containing the following details users,movie and their ratings

Further, the file is preprocessed to divide in to test and train set and dumped into pickle file movieDataset

1. **Data structures used :**

Python list and dictionaries -list are they are dynamic arrays with exponential over-allocation and python dictionaries are implemented as hash tables internally.In addition numpy arrays are used and priority queues are used .

1. **Collaborative Filtering:**

Collaborative-Filtering systems focus on the relationship between users

and items. Similarity of items is determined by the similarity of the

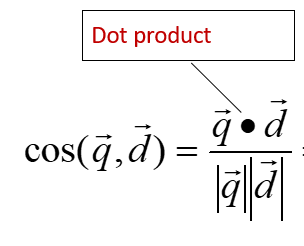
ratings of those items by the users who have rated both items. In the newer, narrower sense, collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The data set is centered around 0 (pearson correlation/center cosine).

Here we are implementing user-user collaborative filtering and user-user collaborative filtering with baseline estimates. In this method the predicted rating for the movie is the rating of the movie by similar set of users. (k nearest neightbours)

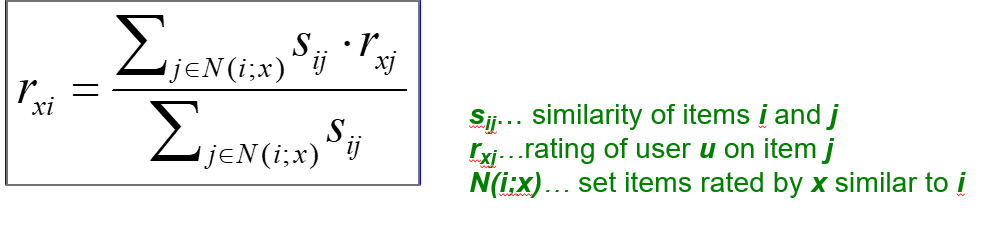
Formulas:

***K value*** : 10

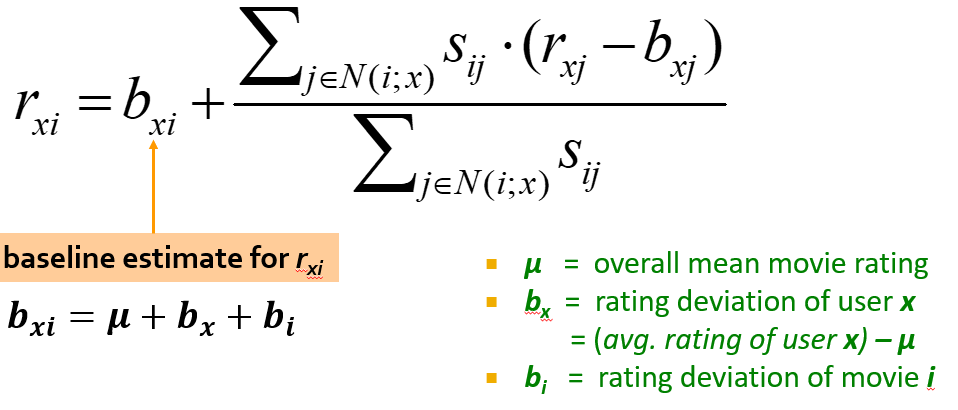
**Cosine Similarity(q,d)**



**Prediction without baselines**



**prediction with baselines :**

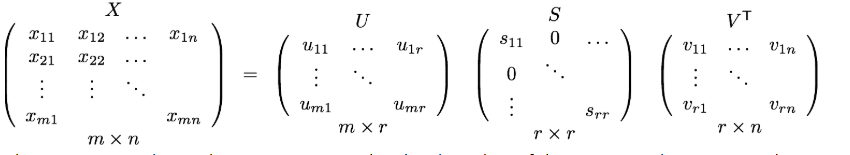


**Pros&Cons:**

Works for any kind of item. Can have popularity biases.For most collaborative filtering systems, having to deal with too few ratings is a far more serious problem.This can cause sparsity problem This problem occurs when the amount of items become very large reducing the number of items users have rated to a tiny percentage. In such a situation it is likely that two people have few rated items in common making the correlation coefficient less reliable

1. **Singular Value Decomposition**

The Singular Value Decomposition (SVD) is a well known matrix factorization technique that factors an **m** by **n** matrix **X** into three matrices as follows:



The matrix S is a diagonal matrix containing the singular values of the matrix X. There are exactly r singular values, where r is the rank of X.

t of users. (k nearest neightbours)

SVD helps in low dimensional representation of a high dimensional matrix

a) U is the matrix symbolising relationship between Users and the movies. that is which concept is most liked by an user.

b)Sigma gives the strength of each of concepts rather relative strength of concepts for given data set and users

c)V gives the relation of movies to concepts that is which movie is best related to which concept

**Steps**:

To calculate SVD , that is U,V,Sigma , we first calculate V vector

V vector is the eigen vector of dot product of transpose of given rating matrix say M\_T and given rating matrix say M. that is

**V = eigen\_vector(M\_T.M)**

Similary U is obtained.now for sigma is diagonal matrix with elements as square root of eigen values in descending order that is

**Sigma[i][i]=sqrt(eigen\_values(M\_T.M))**

**Pros&Cons:**

1)The matrix U and V for large sized M ggives dense matrix and that is computationally very heavy and that can lead to high usage of memory and shortage of it.

SOLUTION :: Use CUR , which gives more error than SVD but gives 2 of its 3 matrix as sparse matrix.

2) The eigen values are high in number and not always the most useful while calculating the original matrix.

SOLUTION :: Retaining only say those starting eigen values whose sum of sqaures is greater than .9 of total sum of squares of the eigen value matrix.This is called retaining 90% energy during SVD

Works for any kind of item.

1. **CUR Decomposition:**

CUR is decomposition method similar to SVD with a few differences which include

it also gives 3 matrices after decomposition but C (analogous to U in SVD ) is column vector and R (analogous to V matrix in SVD) is row vector which are sparse if M is sparse. And U vector (analogous to Sigma in SVD) is dense but that wouldnt matter too much .

How is CUR calculated ?

For C and R first we calculate probabilitymatrix for all indexes in the given rating matrix

temp\_prob(row or col) = (sum of square of all values in the row/column)/p

p=sum of squares of all values in the given matrix

C=Given\_Matrix{ : , for i in temp\_prob ] with values scaled down

R==Given\_Matrix{ for i in temp\_prob , : ] with values scaled down

W = intersection of C and R

We then carry out SVD decomposition of W following all the rules specified above which gives us following



Then we calculate the U matrix as follows

Σ+= psuedo-inverse of Σ

Thus we now have all three of our matrices. So now reconstructing original matrix as

M = C.U.R

Pros&Cons:

Takes less computation time compared to SVD and outer matrices are sparse. Only one small dense matrix

The main shortcoming of CUR is that because it reduces the dimension of the original matrix and is always a approximation for any value of r that is number of concepts taken .

solution::No solution but it is always a good approximation of the original matrix and thus pretty much converges each time .

­­­­

1. **Programming Details:**

Program names:

* + 1. **rs.py** -3 classes implementing all the 3 models

The detailed documentation of the classes and methods can be seen in the program comments.

Files:

* + 1. ratings.dat – raw dataset
    2. movieDataset -preprocessed file

1. **Execution steps**: The Detailed execution steps are outlined in the README file
2. **Results**:

**Collaborative** : Number of neighbours = 10

**Top k precision** : k = 300

**CUR = k -> # of row or columns = rank of matrix = 1412**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Recommender System Technique** | **RMSE** | **Precision on Top K** | **Spearman Rank Correlation** | **Time taken for Prediction** |
| Collaborative | 0.96182 | 0.497059 | 0.999999 | 526.732 |
| Collaborative with Baseline Approach | 1.17673 | 0.337143 | 0.999998 | 427.477 |
| SVD | 1.26628 | 0.413223 | 0.999997 | 100.153 |
| SVD with 90% energy retained | 1.25175 | 0.381356 | 0.999997 | 144.544 |
| CUR | 1.55723 | 0.42 | 0.999996 | 20.0339 |
| CUR with 90% energy retained | 1.18268 | 0.46 | 0.999998 | 54.9173 |

It can be observed that cur method takes least time among the 3 to compute the ratings. But also has more error values