CSE 515 Project: Phase 2
Analysis on IMDB+MovieLens Dataset
Group 3

Group Members
Arun Srivatsa Ramesh,
Achyutha Sreenivasa Bharadwaj,
Kushal Bhatt,
Ninad Jadhav,
Pranav Deshpande,
Suraj Shah

Abstract

In phase 2 of CSE 515 class project we are experimenting with vector models and graph models. This phase is based upon phase 1 so many of the functionalities are borrowed from earlier phase. This phase is focused on analyzing MovieLens + IMDB dataset by performing dimensionality reduction techniques and latent semantic indexing to extract hidden features and tensor decompositions. For high-dimensional datasets dimensionality reduction helps in applying a K-nearest neighbors and other clustering algorithms by minimizing the effects of the curse of dimensionality. Later in the project we experiment with page-ranking algorithm. In the last task based on a user's records movie recommendation is made. The project enabled us to further understand the topics discussed in class by hands-on-activities.

Keywords: Term frequency, High dimensional data, Term weighting, User relevance feedback, TFIDF, IMDB, Latent Semantics, Information retrieval

1 Introduction

This phase is divided into 4 major tasks which are further divided into subtasks. The concepts covered are: Dimensionality reduction, Latent semantics indexing, tensors, tensors decomposition, personalized page-ranking algorithm. We are applying this techniques to analyze and extract the required output.
Task 1 is mainly focused on extracting latent semantics using PCA, SVD and LDA.
Task 2 first part extends the SVD to cluster objects based on latent semantics. Later we experiment with tensors.
In Task 3 based on given seed of actors we create similarity matrix. Then based on transition matrix personalized page-ranking is performed to find the most relevant co-actors.
Last task is movie recommendation system for a given user.

2 Terminology

2.1 Actor

Actor/Actress who has played a role in any movie.

2.2 Genre

Categories under which various movies are classified.

2.3 User

A person who rates and tags movies after he/she has watched them.

2.4 TF

Term Frequency - The frequency with which terms appear in a given document.

2.5 IDF

Inverse Document Frequency - This factor is to determine how important a term is in a document given the corpus.

2.6 SVD

Single Valued Decomposition

2.7 PCA

Principal Component Analysis

2.8 LDA

Latent Dirichlet Allocation

2.9 CP

CANDECOMP/PARAFAC Tensor Decomposition - Decomposes a given tensor into corresponding factor matrices and a core matrix.

2.10 MSE

Mean Squared Error

2.11 ALS

Alternating Least square

3 Goal description

In information retrieval, tf-idf is useful to find relevant documents associated with keywords in a query. Similarly, for our imdb dataset, the problem specification is to construct a graph that relates the keywords (that we call tags) to different components pertaining to the movie world like genres, actors, users, etc. The purpose of the phase1 experiment is to finally build a model that can be used to query relevant information about any of the above components just using the tags. For example, if a user searches for keyword "Drama", the movie section will show drama movies, the actor section will show actors who have acted in drama movies and the users section will show all users who've watched drama movies. Our task here is building the model using some known techniques and compare them to analyze the results. In the second phase of the project, we use the same dataset to perform another series of experiments and compare the outcomes using different techniques we learnt in the class, which include SVD, PCA and LDA decomposition on matrices and CP and Tucker for tensor decomposition to finally achieve some results. Our experiments include finding top 10 related actors given a movie or an actor. We also implement the pagerank algorithm and a generic recommendation system using different approaches and compare/contrast the results of each. This phase also helps us understand how to practically work with sparce data and when to stop lossy decompositions as well as what to make out of the given results and choose the best or a combination of models suited for a particular goal.

4 Assumptions

4.1 General Assumptions

1. The data for user rating/tagging is accurate.
2. There exists no duplicate data in the given dataset.
3. Inputs will be given as expected in the specified format.
4. Data will be given as expected in the specified format(Schema).
5. The TF-IDF computations are used from phase 1.

4.2 Task 1

1. Objects are genres and features are tags.
2. If a tag t does not occur, the $tf_{genre,t}$ is assumed to be 0.

4.3 Task 2

1. Tensor which is constructed does not result in singular decompositions(Determinant is 0.)
2. Non overlapping bins of latent semantics built based on the which feature is more discriminating and describes that latent semantic better.
3. Tensor is built after filtering any conditions given.
4. Sufficient data is provided. Sparse data might produce empty latent semantics.

4.4 Task 3

1. Set of Seed Actors are given as terminal input using iPython.
2. Value of α set to 0.85

4.5 Task 4

1. 3 implementations are explored to compare the recommendation resutls
2. Depending on the input data (number of users and movies), we need high computation power to perform the latent factorization(MODEL 3) implementation
3. For MODEL 3, if a user has tagged a movie, we assume that the movie has been rated. Rating for a such movie (in mltags, but not in mlratings table) is generated by assigning the average of genre rating averages for that specific user.If a genre is not seen for a user, then for that genre for the user we give a low rating if 1.
4.The movie recommendations for model 3 will vary based on the dataset size and the value of latent semantic (k) selected.

5. For Model 1, we assign weights to movies based on the assumption that if user watches a new movie then he would prefer movies from similar genre.

5 Description of the solution/implementation

5.1 Preprocessing

Preprocessing of the dataset involves model creation for each of the tables with proper foreign key relations to clean and store any data inconsistencies. While putting all the data into the MySQL database, we convert the date from "YYYY-MM-DD HH-MM-SS" format into seconds from the UNIX Epoch time. After converting it to milliseconds, we make another meta-data column for both mltags and mlratings and using the standard normalization on this data to convert it into values between range of 0 and 1. Same thing is done for movie_actor_rank column in the movieactor table. The only difference in storing the final actor rank weight is that the weight is stored as 1 – norm_weight_value. The reason for doing this is that numerical actor rank value in the given database is inversely proportional to its importance – lower rank needs higher weights. Also, to handle the edge cases, instead of taking the database max and min values for normalization we take max = max+1 and min=min-1 to avoid having weight = 1 for the latest tag and min = 0 for the lowest tag, which would mean it will be ignored from the computations.

We also have 7 more computations tables specific to each task that help speed up execution of the search queries. Each of these tables can be populated by running the tf() method in every task, in the implementation scripts. These methods need to run only once.

Please note that for idf in each task the idf value is normalized between 0 to 1 by finding the range of idf which will be min=0 and max=log(N). It was noted that normalizing the idf does not change the results in any way since the ratio between each normalized idf tag value is still constant. All these preprocessing sourcecode is under the utils folder and in scripts_p2 folder called populate_db, and the tf methods in each subtask.

There are two stages of preprocessing for task 4. During the first stage, we generate a user-movie_rating data matrix. During the second stage that matrix is decomposed using SVD and then optimized using ALS.All intermediate data are stored in csv files for quick retrieval when actually giving the recommendations

5.2 Task 1

5.2.1 1a :

This task given a genre, identifies and gives top 4 latent semantics/topics using various techniques as PCA, SVD and LDA in the tag-space. For all three implementations sklearn and numpy python libraries are used. First object-features input data matrix is created as (genres x tags) matrix. All the genres and tags in the dataset are being considered. Let's call this matrix M. Each cell value is in the matrix is filled with TF-IDF value of that tag and genre pair. If a tag doesn't appear for that genre then value is 0.

i) SVD : For SVD computation the data matrix M is supplied to the SVD calculator package with parameter 4 as we are interested in top 4 latent semantics. It returns the decomposed U, Sigma and Vt matrices. U is (genre x latent semantics) , Vt is (latent semantics x tags) matrix. Sigma is the core matrix. U represents objects in the reduced 4 dimensions.

ii) PCA : First we standardize the input matrix M. Because, PCA gives a feature subspace that maximizes variance along axis so it is better to standardize input to unit scale (mean=0 , variance=1). Let's call standardized matrix M_std. M_std is supplied to PCA calculator and it returns the decomposed U , Vt matrices. The python sklearn library takes care of all the computations. It uses SVD eigen decomposition on M_std to find principal components. Alternatively it can be done manually as well by first finding co-variance matrix and then find eigenvalues, eigenvectors and then extract principal components.

iii) LDA : Since the LDA computation is asked in normal tag space the input data matrix M is filled with count values of tag-genre pairs. To give importance to latest tags instead of normal occurrence counts, sum of tag weights (computed from phase 1 for TF calculation) is used to fill

cell values. Then LDA package is used to decompose the input matrix and extract top 5 latent semantics. The terminating condition for LDA computation is set to 10000 maximum iterations or until perplexity tolerance is below 1e-12. The output is (object x latent semantics) and (latent semantics x features) matrices.

The output is row describing the given genre in the decomposed (genre x latent semantics) matrix. The numeric values for each latent semantic describes the contribution of that semantic for this genre.

To try and understand what this latent semantics represent one additional processing is done on the returned (latent semantics x features) matrix. As we know that the numeric values of this matrix describes how much a particular feature (tag in our case) contributes to that latent semantic. So we are printing top 10 discriminating tags for each latent semantic to get somewhat idea about extracted latent semantics. In order to do that first the (latent semantics x features) matrix returned by PCA or SVD or LDA is normalized column wise. The contribution values of every tag to all latent semantics are normalized so that the value will closer to 1 where that tag contributes the most and discriminates well and 0 where least contribution. Each cell us normalized as described below.

$$normalized\_value = \frac{(value - min\_column\_value)}{(max\_column\_value - min\_column\_value)} \tag{1}$$

Then we pick top 10 biggest values from each row. The corresponding tags roughly describes that latent semantic. Note that this not entirely correct but it helps in understanding the decomposition.

5.2.2 1b :

This task given a genre, identifies and gives top 4 latent semantics/topics using various techniques as PCA, SVD and LDA in the actor space. The overall procedure and output is similar to task 1a. The only difference here is the input matrix M. M is (genre x actor) matrix. For SVD and PCA each cell value is TF-IDF values for that actor-genre pair. For LDA cell value is sum of actor-rank weights (computed in phase1) instead of tag-weights in task1.

5.2.3 1c :

This task given an actor finds and ranks the top 10 similar actors. There are two ways of finding the answer. First option is based on TF-IDF tag vectors and the second option is making use of top 5 latent semantics. Then this vectors are compared by using either cosine distance or euclidean distance to find nearest neighbors.
The input data is (actor x tags) matrix with each cell having TF-IDF value for that tag-actor pair. If tag doesn't appear for an actor then it's weight is 0.0.

i) Using TF-IDF tag vectors : First pairwise cosine or (inverse)euclidean similarity is computed on data matrix by using sklearn library's utility functions. User has option to select the distance measure. The 10 nearest actors to the given actor are extracted based on this by default and this parameter can be altered. It is important to notice that since the input (actor x tags) matrix is very sparse, TF-IDF gives bad results, with most actors as most of the similarity values for top 10 actors are 0.0 with both cosine and inverse_euclidean.

ii) Using Latent Semantics: First we decompose the data matrix using SVD to represent actors in terms of latent semantics. This reduced dimensions also help in comparing actor vectors. Now similar to above we compute pairwise cosine / euclidean similarity between actors in this decomposed U matrix returned by SVD. The 10 nearest actors to the given actor are extracted based on this and that's the output. This method returns much better results than the TF-IDF method due to decomposition and some value assignment which helps to get a similarity matrix between actors with lesser 0.0 values.

5.2.4 1d :

This task given a movie finds 10 most related actors who have not acted in the movie. Similar to 1c here also we can find the answer by using TF-IDF tag vectors or using latent semantics. First

we create two data matrices let's call them M_1 and M_2. M1 is (movie x tag) and M2 is (actor x tag). Each cell of M_1 is TF-IDF value for that movie-tag pair and similarly for M_2 TF-IDF value for that actor-tag pair. 0 if tag doesn't appear.

i) Using TF-IDF tag vectors : Compute matrix M by performing a dot product as given below.

$$M = M_1 * M_2^T$$

M gives movie - actor similarity and we will extract most similar actors from this by sorting. Now from this extracted actors remove actors who already appear for the given movie and keep top 10 of remaining actors. These are the actors who are most related to this movie but did not act in it.

ii) Using Latent Semantics: First we decompose M_1 and M_2 using SVD to extract 5 latent semantics for both (moviesXtags) and (actorsXtags) matrices. The reduced (object x latent semantic) matrices U_1 and U_2 are used for further analysis.
Rest of the computation is similar to above only that now we operate on U_1 and U_2 instead of M_1 and M_2.

5.3 Task 2

5.3.1 2a :

Similarity Matrix Construction: The TF-IDF vectors calculated for each actor during Phase 1 is used to build a actor-tag matrix. Using cosine similarity as the distance measure the similarity between any two given actors is computed as follows,

$$S(actor_i, actor_j) = \frac{< actor_i, actor_j >}{||actor_i|| * ||actor_j||} \tag{2}$$

where,
$actor_i$ = TFIDF tag vector of actor i.
$actor_j$ = TFIDF tag vector of actor j.
Using the formula above the actor-actor similarity square matrix is constructed.

SVD on the Matrix: We use the python package *scipy.sparse.linalg* to import *svds*. SVD is computed as follows,

$$u, sigma, Vt = svds(matrix, k) \tag{3}$$

where,
u = Left Singular Vectors of the Decomposed Matrix
$sigma$ = Core Values of SVD Decomposition of Matrix
Vt = Right Singular Vectors of the Decomposed Matrix
$matrix$ = Matrix to be decomposed
k = Number of latent features under consideration (3 for the particular task)

Reporting top 3 Latent Semantics: The left singular Vector u is a $N \times 3$ matrix for this task (N = Number of actors). Thus each row represents the particular actor's involvement in the given latent semantics. For each actor we normalize the values in each column in the range of 0 to 1. This is done using the below equation,

$$nvalue = \frac{value - min}{max - min} \tag{4}$$

where,
$nvalue$ = normalized value
$value$ = Value to be normalized
max = Maximum value of the row (Row denotes the actor)
min = Minimum value of the row (Row denotes the actor)
Each Latent Semantic thus formed is a combination of Actors. For each latent semantic, we can find a list of common tags across the actors present in that particular latent semantic. Thus the latent semantic can be represented in the form of tags

Grouping: We use K means clustering to group the list of actors into 3 groups. Each actor can be represented by the given latent semantics, we apply knn on u matrix. Python library *sklearn.cluster* is used for clustering. Each cluster represents the actors with similar TF-IDF values.

5.3.2 2b :

Co-actor Matrix Construction: The Co-actor matrix is a $actor \times actor$ square matrix with each cell i,j containing the number of movies $actor_i$ has acted with $actor_j$.

$CoactorMatrix(i, j)$ = Number of movies $actor_i$ acted with $actor_j$

SVD on the Matrix: We use the python package *scipy.sparse.linalg* to import *svds*. SVD is computed as follows,

$$u, sigma, Vt = svds(matrix, k) \tag{5}$$

where,
u = Left Singular Vectors of the Decomposed Matrix
$sigma$ = Core Values of SVD Decomposition of Matrix
Vt = Right Singular Vectors of the Decomposed Matrix
$matrix$ = Matrix to be decomposed
k = Number of latent features under consideration (3 for the particular task)

Reporting top 3 Latent Semantics: The left singular Vector u is a $Nx3$ matrix for this task (N = Number of actors). Thus each row represents the particular coactor's involvement in the given latent semantics. For each coactor we normalize the values in each column in the range of 0 to 1. This is done using the below equation,

$$nvalue = \frac{value - min}{max - min} \tag{6}$$

where,
$nvalue$ = normalized value
$value$ = Value to be normalized
max = Maximum value of the row (Row denotes the coactor)
min = Minimum value of the row (Row denotes the coactor)
Each latent semantic thus obtained can be shown to consist of a list of Coactors.

Grouping: We use K means clustering to group the list of actors into 3 groups. Each coactor can be represented by the given latent semantics, we apply knn on u matrix. Python library *sklearn.cluster* is used for clustering. Each cluster represents similar coactors.

5.3.3 2c :

To obtain all $(actor, movie, year)$ triples we have to perform a join on MlTags table and the MovieActor table.

The number of distinct movies, distinct actors and distinct years make up the dimensions of the tensor. Three python dictionaries are created to maintain the mapping from tensor index to the corresponding item of the triple. For all the $(actor, movie, year)$ triples, the corresponding index in the tensor is set to 1.

Tensorly [6], a python package is used to perform CP decomposition on this tensor with rank set to 5.

The factor matrices obtained from the decomposition will be the $actor \times latentsemantics$, $movie \times latentsemantics$ and $year \times latentsemantics$.

Now, for each actor/movie/year, a latent semantic is chosen in such a way that the chosen latent semantic discriminates the actor/movie/year better than the rest of the semantics.

For each actor/movie/year we normalize the values in each column(factor matrix) in the range of 0 to 1. This is done using the below equation,

$$nvalue = \frac{value - min}{max - min} \tag{7}$$

where,
$nvalue$ = normalized value
$value$ = Value to be normalized
max = Maximum value of the column (Column denotes the latent semantic)
min = Minimum value of the column (Column denotes the latent semantic)

Degree of membership of each actor/movie/year to the latent semantics is identified based on how discriminating that actor/movie/year is to the given latent semantic.

After normalizing, the most discriminating actors for each latent semantic is identified and put into non overlapping bins.

5.3.4 2d

The average rating for each movie is calculated and stored as a separate table. To obtain all $(tag, movie, rating)$ triples we have to perform a join on MlTags table and the MlRatings table. The condition for join is that the rating for a given movie should be greater than the average rating, and that movie must have been tagged at least once by any user.

The number of distinct tags, distinct movies and distinct ratings make up the dimensions of the tensor. Three python dictionaries are created to maintain the mapping from tensor index to the corresponding item of the triple. For all the $(tag, movie, rating)$ triples, the corresponding index in the tensor is set to 1.

Tensorly [6], a python package is used to perform CP decomposition on this tensor with rank set to 5.

The factor matrices obtained from the decomposition will be the $tag \times latentsemantics$, $movie \times latentsemantics$ and $rating \times latentsemantics$.

Now, for each tag/movie/rating, a latent semantic is chosen in such a way that the chosen latent semantic discriminates the tag/movie/rating better than the rest of the semantics.

For each tag/movie/rating we normalize the values in each column(factor matrix) in the range of 0 to 1. This is done using the below equation,

$$nvalue = \frac{value - min}{max - min} \tag{8}$$

where,
$nvalue$ = normalized value
$value$ = Value to be normalized
max = Maximum value of the column (Column denotes the latent semantic)
min = Minimum value of the column (Column denotes the latent semantic)

Degree of membership of each tag/movie/rating to the latent semantics is identified based on how discriminating that tag/movie/rating is to the given latent semantic.

After normalizing, the most discriminating actors for each latent semantic is identified and put into non overlapping bins.

5.4 Task 3a

The input set of Seed actors is read from file.

5.4.1 Similarity Matrix Construction:

The TF-IDF vectors calculated for each actor during Phase 1 is used to build a actor-tag matrix. Using cosine similarity as the distance measure the similarity between any two given actors is computed as follows,

$$S(actor_i, actor_j) = \frac{< actor_i, actor_j >}{||actor_i|| * ||actor_j||} \tag{9}$$

where,
$actor_i$ = TFIDF tag vector of actor i.
$actor_j$ = TFIDF tag vector of actor j.

Using the formula above the actor-actor similarity square matrix is constructed.

5.4.2 Transition Matrix Construction:

The actor-actor similarity matrix does not represent the transition probabilities. The columns of the similarity matrix are normalized to denote transitions. Also, the transition matrix must be a square matrix and all values of the transition matrix should be non negative. The columns of the transition matrix will sum to 1.

$$T(i,j) = \frac{M(i,j)}{\sum_{j=0}^{n} M(i,j)} \tag{10}$$

where,
$M(i,j)$ = Matrix element at row i and column j.

5.4.3 Personalized Page Rank Algorithm

Personalized Page Rank Algorithm [2] is used to find out the 10 most relevant co-actors to the actors in the given seed set S.

$$RelecantCoActors(S) = \alpha * (I - (1-\alpha)T)^{-1} \cdot \vec{t} \tag{11}$$

where,
S = Set of seed actors.
α = Probability of transition or reset.
I = Identity matrix
T = Transition matrix
$\vec{t}$ = Teleportation vector with $1/size(S)$ probabilities for all nodes in S.

The idea of using this algorithm is to find the most relevant co-actors by initializing a random walk with restarts from any of the nodes and make transitions based on the transition matrix edge weights with probability α. In addition to this, there is a chance that random walker will jump to anyone of the given seed nodes with probability $(1-\alpha)$.

In general, inverse computation is expensive. So, inverse is computed once and stored here which can be reused for different query sets of seed actors.

5.5 Task 3b

The input set of Seed actors is read from file.

5.5.1 Co-actor Matrix Construction:

The Co-actor matrix is a $actor \times actor$ square matrix with each cell i,j containing the number of movies $actor_i$ has acted with $actor_j$.

$CoactorMatrix(i,j)$ = Number of movies $actor_i$ acted with $actor_j$

5.5.2 Transition Matrix Construction:

The $actor \times actor$ co-actor matrix does not represent the transition probabilities. The columns of the similarity matrix are normalized to denote transitions. Also, the transition matrix must be a square matrix and all values of the transition matrix should be non negative. The columns of the transition matrix will sum to 1.

$$T(i,j) = \frac{M(i,j)}{\sum_{j=0}^{n} M(i,j)} \tag{12}$$

where,
$M(i,j)$ = Matrix element at row i and column j.

5.5.3 Personalized Page Rank Algorithm

Personalized Page Rank Algorithm [2] is used to find out the 10 most related actors to the actors in the given seed set S.

$$RelatedActors(S) = \alpha * (I - (1-\alpha)T)^{-1} \cdot \vec{t} \tag{13}$$

where,
S = Set of seed actors.
α = Probability of transition or reset.
I = Identity matrix
T = Transition matrix
$\vec{t}$ = Teleportation vector with $1/size(S)$ probabilities for all nodes in S.

The idea of using this algorithm is to find the related actors by initializing a random walk with restarts from any of the nodes and make transitions based on the transition matrix edge weights with probability α. In addition to this, there is a chance that random walker will jump to anyone of the given seed nodes with probability $(1 - \alpha)$.

In general, inverse computation is expensive. So, inverse is computed once and stored here which can be reused for different query sets of seed actors.

5.6 Task 4

Task 4 is about developing a movie recommender system, based on the historical data available about a user and the movies that he has rated or tagged.For this task 3 MODELS were implemented to explore the type output obtained with each implementation

5.6.1 MODEL 1:

This is a simple model which recommends movies based on the genre ratings for specific user. First we assign weights to movies based on the year, using exponential decay.Latest movies get higher rating and older movies get lower rating. We use a decay constant 'k' to get uniform decay values

$$movie_{wt} = e^{(-k*(max\_year - current\_year))} \tag{14}$$

The next step is to generate a user-genre matrix using the movie ratings and the movie weights. For each user, the genre weight is calculated as

$$genre_{wt} = Sum((movie_{wt} * movie\_rating)) \tag{15}$$

where,
$movie_{wt}$ = weight of movie watched by a user
movie_rating = rating given by a user to a movie

Based on the ranking of genres,recommend 5 unwatched movies starting from the genres that are higher ranked. The one possible drawback of this approach is that it considers only 'genre' of a movie as a major deciding factor and hence tend to recommend movies predominantly from a specific genre only thus discarding recommendations from any other genre of which the user might have seen the movie.

5.6.2 MODEL 2:

To handle the drawback of MODEL 1 in a better way, MODEL 2 generates a user-movie matrix by using the genre. Here we assume that the genres are somewhat of 'psuedo latent semantic' as we already have access to it.

First we generate a genre,tag vectors for all genres as done in Phase 1 of the project, as follows.

$$TF_{genre} = Twt/Sum(1 - n)Twt \tag{16}$$

where Twt = sum of (tag_wt) of a tag for a genre
Sum (1-n) Twt = sum of (tag_wt) for all tags of a genre

The IDF value determines the relevance of a tag for all the genre and it is calculated as : IDFgenre = log(Count<genre,movie> / Sum Tag<genre,movie>)

where Count<genre,movie> = All <genre,movie> entries from 'mlmovies_clean' table

$\sum Tag < genre, movie >$ = Sum of (tag_wt) for all <genre,movie> having a given tag

The TF-IDFgenre value is calculated as : Tfgenre * IDFgenre

Now we generate movie-genre matrix from the genre_tag matrix. For a genre assigned to movie, the movie-genre value is sum of tags that belong to a movie and seen for that genre.A user-genre matrix is generated as done for MODEL 1. Now we can rank the the movies for a specific user using the dot product of the transpose of movie_genre_vector and user_genre_vector
We return the top 5 movies(watched movies are filtered out.)

5.6.3 MODEL 3:

MODEL 3 uses matrix factorization [10] to generate the recommendations. The MODEL works in 3 stages.

In stage 1, we generate the user,movie_rating matrix from the mlratings and mltags data tables. The reason for including the mltags tables has been stated in assumptions and the method for calculating the ratings is mentioned.Taking into account the high volume of data, we store the matrix is csv file so that the next stage can retrieve it. We also store the user_ids and the movie_ids so that they can be retrieved to calculate the predicted ratings as reconstruction of matrix after decomposition will lead to loss of this data.

In stage 2, we carry out the matrix decomposition using SVD[10]. The goal is to predict the values of all the matrices using the decomposed matrix as accurately as possible[11], but as our input data matrix is very sparse only using SVD may not suffice. So we minimize the error using only the observed movie recommendations and use regularization to avoid data over fitting [10][11]. For error minimizing, we use the alternating least square approach as it give advantages over using gradient descent[10].

$$\min_{q^*,p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\| q_i \|^2 + \| p_u \|^2) \quad [10]$$

where,
u and p are the decomposed matrices
r = known movie ratings for a user
λ = regularization constant

While decomposing the input matrix latent semantics k=10, 20 and 86(maximum number of movies) was used. It was observed that the errors start converging after 50 iterations for the k=20 and 86, but take about 130 iterations for k=10.

Once the iterations are done we reconstruct the matrix by taking the dot product of optimized decomposed matrices storing the output to csv file.

In stage 3, for a given input of the user ,we map the user_id and movie_id by using the data generated in stage 1, generate the ratings for all the movies, filter out watched movies and return the top 5 movies as recommendation.

The model can be improved by introducing other techinques as discussed in [10] as well as using an optimum 'regularization constant'[10]

6 User Interface Specifications

6.1 Instructions to execute:

6.1.1 Preprocessing/Setting up

Please refer to the readme file provided for in-detailed setting-up instructions.
Preprocessing:
–> Open terminal export django settings and log into ipython
> $exportDJANGO\_SETTINGS\_MODULE = mwd_proj.settings$
> ipython

» import django
» django.setup()
» from mwd_proj.scripts_p2 import (print_genreactor_vector, print_genre_vector, print_user_vector, print_actor_vector, print_movie_vector)
» from mwd_proj.scripts_p2 import (part1)
» from mwd_proj.scripts_p2.Arun import (part2, part3)
» from mwd_proj.scripts_p2.Ninad import (part4)
» from mwd_proj.phase2.models import *
Note - print the output variables to get decomposed matrices

6.2 Task1

1a) a = part1.compute_Semantics_1a('<method>','<genre>',<k>)
where: <method> = SVD, PCA or LDA
<genre> = Action, Horror, etc
<k> = No of latent_semantics to be considered
eg: a=part1.compute_Semantics_1a('SVD','Action',4)

1b) b = part1.compute_Semantics_1b('<method>','<genre>',<k>)
where: <method> = SVD, PCA or LDA
<genre> = Action, Horror, etc
<k> = No of latent_semantics to be considered
eg: b=part1.compute_Semantics_1b('SVD','Action',4)

1c) c, z = part1.compute_Semantics_1c('<method>','<actor-name>','<similarity-measure>',<actors-count>,<k>,<flag>)
where: <method> = TF-IDF, SVD
<actor-name> = Lillard, Matthew; Affleck, Ben; etc
<similarity-measure> = cosine, euclidean
<actors-count> = No of similar actors to be returned
<k> = No of latent semantics to be considered for computations
<flag> = True, False - displays output on stdout if true
eg: c, z = part1.compute_Semantics_1c('TF-IDF','Lillard, Matthew','cosine',10,5,True)
eg: c, z = part1.compute_Semantics_1c('SVD','Lillard, Matthew','cosine',10,5,True)

1d) d = part1.compute_Semantics_1d('<method>','<movie-name>',<actors-count>,<k>,<flag>)
where: <method> = TF-IDF, SVD
<movie-name> = Swordfish, Harry Potter and the Prisoner of Azkaban, Pitch Black, etc
<actors-count> = No of similar actors to be returned
<k> = No of latent semantics to be considered for computations
<flag> = True, False - displays output on stdout if true
eg: d = part1.compute_Semantics_1d('SVD','Swordfish',10,5,True)

6.3 Task2

2a) a = part2.compute_Semantics_2a(<k>,<max-actors>) where: <k> = No of latent semantics
<max-actors> = maximum actors to output in each group
eg: a = part2.compute_Semantics_2a(3,5)

2b) b = part2.compute_Semantics_2b(<k>,<max-actors>)
where: <k> = No of latent semantics
<max-actors> = maximum actors to output in each group
eg: b = part2.compute_Semantics_2b(3,5)

2c) eg: c = part2.compute_Semantics_2c()

2d) eg: d = part2.compute_Semantics_2d()

6.4 Task3

3a) part3.compute_Semantics_3a(<actor-ids-list>)
where: <actor-ids-list> = set([1860883,486691,1335137,901175]), etc
eg: part3.compute_Semantics_3a(set([1860883,486691,1335137,901175]))

3b) part3.compute_Semantics_3b(<actor-ids-list>)
where: <actor-ids-list> = set([1860883,486691,1335137,901175]), etc
eg: part3.compute_Semantics_3b(set([1860883,486691,1335137,901175]))

6.5 Task4

Please note this task requires following preprocessing to be run only once:
> cd MWDBProject/mwd_proj/mwd_proj/scripts_p2/Ninad
> python preprocessor_model3.py
> python preprocessor_model3_comp.py
4a) part4.compute_Semantics_4(<user-id>)
where: <user-id> = 1027, etc
eg: part4.compute_Semantics_4(1027)

6.6 System Requirements

There are hardware prerequisites a system or PC needs to have, these are;
• Hard-disk space: 1 GB
• Minimum RAM: 4 GB
For software, any of the following can be used:
• Ubuntu OS
• Command prompt/Terminal
• Python 2.7
• MySQL

7 Related Work

Dimensionality reduction for efficient retrieval and reducing computation times for high dimensional data is addressed in this phase by experimenting with various dimensionality reduction techniques like PCA, SVD, LDA [1][3]. Using object-object similarity information to obtain relevant objects to a given set of objects [2][8] is used in this phase by doing a Random Walk with Restart (Personalized Page Rank). PCA,SVD,LDA are well studied and widely used techniques in high dimensional datasets. It is used in many fields from bio-informatics, NLP to recommender systems and data mining. [7] shows KNN to be the best approach for clustering. The main advantage lies in its linear time complexity. The only possible drawback of KNN would be not achieving near optimal solution. [9] discusses on techniques to obtain a good solution for KNN.

8 References

[1] Dr. K Selcuk Candan, Dr. Maria Luisa Sapino. “Data Management for Multimedia Retrieval. Cambridge Publication” (Section 12.4).
[2]J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In KDD, pages 653658, 2004.
[3] `https://plot.ly/ipython-notebooks/principal-component-analysis`
[4] `https://github.com/maheshakya/my_first_project/wiki/Creating-a-data-set-with-svd`
[5] `http://blog.districtdatalabs.com/principal-component-analysis-with-python`
[6]`https://github.com/tensorly/tensorly/tree/master/tensorly/decomposition`
[7] Steinbach, Michael, George Karypis, and Vipin Kumar. "A comparison of document clustering techniques." KDD workshop on text mining. Vol. 400. No. 1. 2000.
[8] `https://github.com/ashkonf/PageRank`
[9] https://dl.acm.org/citation.cfm?id=1015408, "K-means clustering via principal component analysis"

[10] MATRIX FACTORIZATION TECHNIQUES FOR RECOMMENDER SYSTEMS, Yehuda Koren, Yahoo Research Robert Bell and Chris Volinsky, AT&T Labs—Research
[11]https://stanford.edu/ rezab/classes/cme323/S15/notes/lec14.pdf

9 Appendix

9.1 Contribution of team members for tasks

TASK 1a - Kushal,Suraj
TASK 1b - Kushal,Suraj
TASK 1c - Suraj,Kushal
TASK 1d - Suraj,Kushal
TASK 2a - Pranav,Achyutha
TASK 2b - Pranav,Achyutha
TASK 2c - Arun,Achyutha
TASK 2d - Arun,Achyutha
TASK 3a - Arun,Ninad
TASK 3b - Arun,Ninad
TASK 4 - Ninad

9.2 Sample Outputs

1a

U: 20 4 Sigma: (4, 4) V: (4, 79)

For genre Action Latent semantics are: [-0.11170345 0.82682627 0.50857812 -0.0906995]

Latent Semantics: 1 = [u'crime', u'marvel', u'murder', u'motorcycle', u'court', u'comedy', u'dark', u'gore', u'hannibal lecter', u'nudity (topless)']

Latent Semantics: 2 = [u'franchise', u'fish', u'harry potter', u'time travel', u'video game adaptation', u'history', u'england', u'espionage', u'magic', u'zombies']

Latent Semantics: 3 = [u'over the top', u'nudity (topless - notable)', u'hackers', u'hacking', u'interesting', u'cool', u'musicians', u'not funny', u'nudity (rear)', u'action']

Latent Semantics: 4 = [u'nudity (rear)', u'war', u'wwii', u'dumb but funny', u'musicians', u'best war films', u'san francisco', u'world war ii', u'interesting', u'boxing']

[[-1.56619917e-01 2.66614581e-02 -8.71711934e-02 -1.57132252e-01]
[1.08420217e-19 8.13151629e-20 1.35525272e-19 -9.48676901e-20]
[0.00000000e+00 -2.77555756e-17 -8.32667268e-17 8.32667268e-17]
[2.01106156e-01 -2.31263778e-01 2.87442367e-01 -2.60429009e-01]
[0.00000000e+00 3.46944695e-18 2.77555756e-17 -1.73472348e-18]
[-1.68231898e-02 6.14848185e-02 -1.38285081e-01 -2.28397983e-01]
[0.00000000e+00 5.55111512e-17 1.11022302e-16 -8.32667268e-17]
[2.15002760e-01 -2.38317310e-01 2.94888347e-01 -2.63184464e-01]
[-7.56849251e-01 -1.24574153e-01 -1.59889559e-02 -4.10103897e-01]
[0.00000000e+00 0.00000000e+00 -2.77555756e-17 3.46944695e-17]
[1.06723269e-01 1.12974858e-01 1.24660808e-01 -5.77835668e-02]
[1.11022302e-16 5.55111512e-17 2.22044605e-16 -6.93889390e-17]
[-1.11703454e-01 8.26826270e-01 5.08578120e-01 -9.06995033e-02]
[-5.55111512e-17 -5.55111512e-17 -1.94289029e-16 0.00000000e+00]
[-5.26503503e-02 -1.56932126e-01 1.53290507e-01 -1.67003044e-01]
[3.01807184e-01 -7.85633670e-02 2.41496281e-01 -5.22124386e-01]
[-5.62854244e-02 -1.66555762e-01 1.62269372e-01 -1.78090057e-01]
[6.93889390e-18 -6.93889390e-18 -1.38777878e-17 8.67361738e-18]
[-2.93837830e-01 5.17605105e-02 -1.56377119e-01 -2.12621303e-01]
[3.29333642e-01 3.34794149e-01 -6.29797518e-01 -4.78410034e-01]]

The above matrix is decomposed U matrix. Each row represent a genre and column latent semantic. Cell values describes how much that latent semantic describes a genre. As explained earlier the mapping shows top 10 tags that are describing the latent semantic.
The output follows the same pattern for PCA and LDA as well.

1b

Latent Semantic: 1 = [u'Pitt, Brad', u'Radcliffe, Daniel', u'Papenbrook, Bob', u'Payne, Bruce', u'Penn, Kal', u'McDermott, Dylan', u'Allen, Tim', u'Shandling, Garry', u'Seth, Joshua', u'Macy, William H.']

Latent Semantic: 2 = [u'Ice Cube', u'Stiller, Ben', u'Hawke, Ethan', u'Neri, Francesca', u'Mussett, Tory', u'Thieriot, Max', u'Hounsou, Djimon', u'Nielsen, Connie', u'Jackman, Hugh', u'Keitel, Harvey']

Latent Semantic: 3 = [u'McDormand, Frances', u'Bilson, Rachel', u'Dutcher, Richard', u'Manterola, Patricia', u'Katt, Nicky', u'Duchovny, David', u'Kodjoe, Boris', u'Douglas, Michael', u'McCormack, Mary', u'Bynes, Amanda']

Latent Semantic: 4 = [u'Toussaint, Beth', u'Moore, Kenya', u'Hauser, Cole', u'Landes, Michael', u'Belle, Camilla', u'Robbins, Tim', u'Palmer, Gretchen', u'Liotta, Ray', u'Hopkins, Anthony', u'Cassidy, Katie']

[[0.05013316 0.84903136 0.05078257 0.05005292]
[0.08346014 0.08462502 0.74841359 0.08350125]
[0.74792151 0.08398431 0.08467438 0.0834198]
[0.00947261 0.97166859 0.0095144 0.0093444]
[0.08674478 0.08641537 0.7392874 0.08755245]
[0.01010197 0.96965411 0.01014326 0.01010065]
[0.08341944 0.74770369 0.08541843 0.08345845]
[0.00619938 0.98132172 0.00633249 0.0061464]
[0.00397156 0.00415876 0.98793138 0.00393831]
[0.08337843 0.74893842 0.08428731 0.08339585]
[0.91586569 0.0284296 0.02789219 0.02781252]
[0.03686969 0.88942881 0.03690516 0.03679634]
[0.98142555 0.00621881 0.0062016 0.00615404]
[0.74680321 0.08638685 0.08338819 0.08342174]
[0.03587339 0.0368286 0.89153509 0.03576292]
[0.0043446 0.98728186 0.00423463 0.00413891]
[0.05030709 0.05215091 0.84748505 0.05005695]
[0.05011966 0.84905485 0.05073088 0.05009462]
[0.0140831 0.95714375 0.01468697 0.01408618]
[0.00398077 0.66199789 0.00405327 0.32996807]]

The above matrix is decomposed (genre x latent semantic) matrix. Each row represent a genre and column latent semantic. Cell values describes how much that latent semantic describes a genre. As explained earlier the mapping shows top 10 actors that are describing the latent semantic.

9.2.1 Task 2

2a

Sample output for reporting 3 Latent Semantics and partitioning into 3 Groups.
————————————-SEMANTICS————————————-
LATENT SEMANTIC 1:
[(u'dark', 0.6415388955117399), (u'heroine in tight suit', 0.6104438382820595), (u'marvel', 0.6064242364429479), (u'bad acting', 0.6047843078124928), (u'comic book', 0.590314631165654)]

LATENT SEMANTIC 2:
[(u'interesting', 24.625262857657166), (u'pointless', 4.735677295960617), (u'motorcycle', 2.239891033093137), (u'based on a tv show', 1.9127814900139841), (u'predictable', 1.1604393986320254), (u'drugs', 1.114736864721087), (u'family', 1.0662289330513015), (u'based on book', 0.9315019286015793)]

LATENT SEMANTIC 3:
[(u'dreamworks', 1.2599864850729672), (u'murder', 1.2432882991134857), (u'pirates', 1.1297310075860727), (u'not funny', 1.111978265631911), (u'swashbuckler', 0.8928703266122562)]

————————————-GROUPING————————————-
GROUP 1: Allen, Tim Astin, Sean Banderas, Antonio Bell, Jamie Bleek, Memphis Bleu, Corbin

Bradshaw, Terry Bright, Cameron Brimley, Wilford Brooks, Avery Brown, Matthew A. Burke, Billy Burns, Edward Butler, Gerard Campbell, Adam Carell, Steve Carman Carman, Michael Chinlund, Nick Cho, John Christensen, Hayden Church, Thomas Haden Cooper, Chris Corddry, Rob Daniels, Jeff Dash, Damon De Niro, Robert Diesel, ...

GROUP 2: Affleck, Ben Andrieu, Sebastien Ashker, John Beesley, Max Bon Jovi, Jon Brisbin, David Burn, Scott Byrne, Gabriel Campbell, Jeremie Cedric the Entertainer Chan, Jackie Cheadle, Don Conley, Jack Cook, Dane Corbett, John Costner, Kevin Cronin, Jonathan Cross, Joseph Curtis, Cliff Cusick, Henry Ian Daly, Tim David, Keith Dempsey, Patrick Depp, Johnny DMX Douglas, Michael Duchovny, David Duncan, Michael Clarke Dutton, Charles S. Epps, Omar Farrell, Colin Fillion, Nathan Finney, ...

GROUP 3: Anderson, Anthony Baldwin, Alec Buscemi, Steve Duke, Bill Feore, Colm Fiennes, Joseph Germann, Greg Harris, Ed Hopkins, Anthony Isaacs, Jason Landes, Michael Law, Jude Lawrence, Martin Leary, Denis Leguizamo, John Liotta, Ray Muniz, Frankie Oldman, Gary Paetkau, David Pitt, Brad Reeves, Keanu Romano, Ray Snoop Dogg Stiller, Ben Visnjic, Goran Whitaker, Forest Williamson, Fred Wilson, Owen Yoakam, ...

2b

Sample output for reporting 3 Latent Semantics and partitioning into 3 Groups.
————————————-SEMANTICS————————————- LATENT SEMANTIC 1:
Ulliel, Gaspard Neri, Francesca Fishburne, Laurence Flanagan, Tommy Moore, Julianne
LATENT SEMANTIC 2:
Williamson, Fred Law, Jude Wen, Ming-Na Walken, Christopher Dempsey, Patrick
LATENT SEMANTIC 3:
Willis, Bruce Whitfield, Lynn Perry, Tyler Luke, Derek McCole Bartusiak, Skye ————————————————-GROUPING————————————-
GROUP 1:
Pollak, Kevin Bliss, Lucille Long, Nia McDormand, Frances Farrell, Colin Warren, Estella Berry, Halle Affleck, Ben Jackson, Samuel L. Cruz, PenÃ©lope Ryan, Meg Schreiber, Liev Cook, Dane Staunton, Imelda Klein, Chris Gyllenhaal, Jake Pitt, Brad Blair, Selma Campbell, Adam Cox, Courteney Vaughn, Vince Hawke, Ethan Highmore, Freddie Hu, Kelly LaBeouf, Shia Weisz, Rachel Ne-Yo Williamson, Fred Dempsey, Patrick Williams, Katt Walken, Christopher Goldblum, Jeff Whalin, Justin Pegg, Simon Kelly, David Reid, Tara Morgan, Tracy Astin, Sean Wong, ...

GROUP 2: Flanagan, Tommy Papenbrook, Bob Smart, Amy Koteas, Elias Hunter, Holly Depp, Johnny Mitchell, Radha Isaacs, Jason Parrack, Jim Levi, Zachary Nouri, Michael Campbell, Jeremie Kodjoe, Boris Lillard, Matthew Anderson, Anthony Dourdan, Gary Larter, Ali Hounsou, Djimon Crewson, Wendy Visnjic, Goran Lane, Diane Bleek, Memphis Paxton, Bill TachovskÃ¡, Helena-Lia Edwards, Anthony Thomas, Eddie Kaye Baldwin, Alec Finney, Albert Bonham Carter, Helena Christensen, Hayden Ulliel, Gaspard Fishburne, Laurence Bright, Cameron Grammer, Kelsey Carell, ...

GROUP 3: Franco, James Armstrong, Brittany Mainwaring, Cameron Shaw, Fiona Sisto, Jeremy De Niro, Robert Burke, Billy Turner, Frank C. Kilmer, Val McDermott, Dylan Cho, John Andrieu, Sebastien Chaplin, Carmen Bell, Jamie DMX McCormack, Mary Cheadle, Don McCole Bartusiak, Skye Church, Thomas Haden Dhavernas, Caroline Wayans, Marlon Swank, Hilary Conley, Jack Robb, AnnaSophia Spearritt, Hannah Cook, ...

2c

LATENT SEMANTIC 1 [u'Lane, Diane', 1.0] [u'Hanks, Colin', 1.0] LATENT SEMANTIC 2 [u'Anderson, Anthony', 1.0] LATENT SEMANTIC 3 [u'Hunter, Holly', 1.0] [u'Dunst, Kirsten', 1.0] [u'Thornton, Billy Bob', 1.0] [u'Freeman, Morgan', 1.0] [u'Wilson, Owen', 0.9128714156393383 8] [u'Taylor-Johnson, Aaron', 0.9128714127207046] [u'Fisher, Tom', 0.9128714127207046] [u'Chan, Jackie', 0.9128714127207046] [u'Zeta-Jones, Catherine', 0.89012000127814872] [u'Pfeiffer, Michelle', 0.89012000127814872] [u'Pitt, Brad', 0.89012000127814872] [u'Fiennes, Joseph', 0.89012000127814872] [u'Larter, Ali', 0.79621828143137863] [u'Cook, A.J.', 0.79621828143137863] [u'Paetkau, David', 0.79621828143137863] [u'Landes, Michael', 0.79621828143137863] [u'Williamson, Fred', 0.79357359363562596] [u'Stiller, Ben', 0.79357359363562596] [u'Snoop Dogg', 0.79357359363562596] [u'McDormand, Frances', 0.79357359084895429] [u'Maguire, Tobey', 0.79357359084895429] [u'Douglas, Michael', 0.79357359084895429] [u'Downey Jr., Robert', 0.79357359084714707] [u'Toussaint, Beth', 0.79357359073166012] [u'Rutherford,

Kelly', 0.79357359073166012] [u'Schreiber, Liev', 0.79357359073166012] [u'Jackson, Roger', 0.79357359073166012] [u'Palmer, Gretchen', 0.79357359072880684] [u'Moore, Kenya', 0.79357359072880684] [u'Smith, Soloman K.', 0.79357359072880684] [u'Dourdan, Gary', 0.79357359072880684] [u'Mays, Jayma', 0.79357359072128331] [u'Coolidge, Jennifer', 0.79357359072128331] [u'Penn, Kal', 0.79357359072128331] [u'Campbell, Adam', 0.79357359072128331] [u'Good, Meagan', 0.79357359072080758] [u'Short, Columbus', 0.79357359072080758] [u'Ne-Yo', 0.79357359072080758] [u'Henson, Darrin Dewitt', 0.79357359072080758] [u'Popper, Robert', 0.79357359071820877] [u'Pegg, Simon', 0.79357359071820877] [u'Nighy, Bill', 0.79357359071820877] [u'Freeman, Martin', 0.79357359071820877] [u'Romijn, Rebecca', 0.7935735907169944] [u'Kinnear, Greg', 0.7935735907169944] [u'De Niro, Robert', 0.7935735907169944] [u'Bright, Cameron', 0.7935735907169944] [u'Whitfield, Lynn', 0.79357359071699218] [u'Wen, Ming-Na', 0.79357359071699218] [u'Weisz, Rachel', 0.79357359071699218] [u'Tyson, Cicely', 0.79357359071699218] [u'Thurman, Uma', 0.79357359071699218] [u'Thompson, Tessa', 0.79357359071699218] [u'Theron, Charlize', 0.79357359071699218] [u'Texada, Tia', 0.79357359071699218] [u'Sykes, Wanda', 0.79357359071699218] [u'Smart, Amy', 0.79357359071699218] [u'Shaw, Fiona', 0.79357359071699218] [u'Ryan, Meg', 0.79357359071699218] [u'Robb, AnnaSophia', 0.79357359071699218] [u'Reid, Tara', 0.79357359071699218] [u'Polley, Sarah', 0.79357359071699218] [u'Parker, Sarah Jessica', 0.79357359071699218] [u'Page, Ellen', 0.79357359071699218] [u'Neri, Francesca', 0.79357359071699218] [u'Mussett, Tory', 0.79357359071699218] [u'Morgan, Carrie', 0.79357359071699218] [u'Moore, Julianne', 0.79357359071699218] [u'Moore, Demi', 0.79357359071699218] [u'McCormack, Mary', 0.79357359071699218] [u'McCole Bartusiak, Skye', 0.79357359071699218] [u'Manterola, Patricia', 0.79357359071699218] [u'Lucas, Jessica', 0.79357359071699218] [u'Lamont, Ceilidh', 0.79357359071699218] [u'Keener, Catherine', 0.79357359071699218] [u'Jovovich, Milla', 0.79357359071699218] [u'Isitt, Kate', 0.79357359071699218] [u'Hunt, Bonnie', 0.79357359071699218] [u'Gellar, Sarah Michelle', 0.79357359071699218] [u'Foster, Jodie', 0.79357359071699218] [u'Forlani, Claire', 0.79357359071699218] [u'Ferris, Pam', 0.79357359071699218] [u'Devine, Loretta', 0.79357359071699218] [u'Deschanel, Emily', 0.79357359071699218] [u'Crewson, Wendy', 0.79357359071699218] [u'Cox, Courteney', 0.79357359071699218] [u'Cook, Rachael Leigh', 0.79357359071699218] [u'Chaplin, Carmen', 0.79357359071699218] [u'Cassidy, Katie', 0.79357359071699218] [u'Cardellini, Linda', 0.79357359071699218] [u'Caplan, Lizzy', 0.79357359071699218] [u'Breslin, Abigail', 0.79357359071699218] [u'Bonham Carter, Helena', 0.79357359071699218] [u'Bloodgood, Moon', 0.79357359071699218] [u'Bliss, Lucille', 0.79357359071699218] [u'Blair, Selma', 0.79357359071699218] [u'Bilson, Rachel', 0.79357359071699218] [u'Berry, Halle', 0.79357359071699218] [u'Bello, Maria', 0.79357359071699218] [u'Belle, Camilla', 0.79357359071699218] [u'Armstrong, Brittany', 0.79357359071699218] [u'Abdul, Paula', 0.79357359071699218] [u'Willis, Bruce', 0.79357359071699218] [u'Williams, Katt', 0.79357359071699218] [u'Williams, Jeremy', 0.79357359071699218] [u'Whitaker, Forest', 0.79357359071699218] [u'Weber, Jake', 0.79357359071699218] [u'Watson, Barry', 0.79357359071699218] [u'Walker, Paul', 0.79357359071699218] [u'Wahlberg, Donnie', 0.79357359071699218] [u'Visnjic, Goran', 0.79357359071699218] [u'Vaughn, Vince', 0.79357359071699218] [u'Van Der Beek, James', 0.79357359071699218] [u'Underwood, Blair', 0.79357359071699218] [u'Turner, Frank C.', 0.79357359071699218] [u'Thomas, Marcus', 0.79357359071699218] [u'Thomas, Eddie Kaye', 0.79357359071699218] [u'Stahl-David, Michael', 0.79357359071699218] [u'Sparks, Omillio', 0.79357359071699218] [u'Slater, Christian', 0.79357359071699218] [u'Sigel, Beanie', 0.79357359071699218] [u'Shandling, Garry', 0.79357359071699218] [u'Shalhoub, Tony', 0.79357359071699218] [u'Schwarzenegger, Arnold', 0.79357359071699218] [u'Russell, Kurt', 0.79357359071699218] [u'Romano, Ray', 0.79357359071699218] [u'Robbins, Tim', 0.79357359071699218] [u'Rhames, Ving', 0.79357359071699218] [u'Reno, Jean', 0.79357359071699218] [u'Reeves, Keanu', 0.79357359071699218] [u'Radcliffe, Daniel', 0.79357359071699218] [u'Quinn, Aidan', 0.79357359071699218] [u'Quaid, Dennis', 0.79357359071699218] [u'Prinze Jr., Freddie', 0.79357359071699218] [u'Pollak, Kevin', 0.79357359071699218] [u'Phifer, Mekhi', 0.79357359071699218] [u'Perry, Tyler', 0.79357359071699218] [u'Parrack, Jim', 0.79357359071699218] [u'Oldman, Gary', 0.79357359071699218] [u'Nouri, Michael', 0.79357359071699218] [u'Morgan, Tracy', 0.79357359071699218] [u'Miller, T.J.', 0.79357359071699218] [u'McDermott, Dylan', 0.79357359071699218] [u'Matthews, Dave', 0.79357359071699218] [u'Mainwaring, Cameron', 0.79357359071699218] [u'Luke, Derek', 0.79357359071699218] [u'LL Cool J', 0.79357359071699218] [u'Liotta, Ray', 0.79357359071699218] [u'Lillard, Matthew', 0.79357359071699218] [u'Leguizamo, John', 0.79357359071699218] [u'Leary, Denis', 0.79357359071699218] [u'Law, Jude', 0.79357359071699218] [u'LaBeouf, Shia', 0.79357359071699218] [u'Kutcher, Ashton', 0.79357359071699218] [u'Koteas, Elias', 0.79357359071699218] [u'Kodjoe, Boris', 0.79357359071699218] [u'Klein, Chris', 0.79357359071699218] [u'Kilmer, Val', 0.79357359071699218] [u'Kelly, David', 0.79357359071699218] [u'Katt, Nicky', 0.79357359071699218] [u'Jackman, Hugh', 0.79357359071699218] [u'Isaacs, Jason', 0.79357359071699218] [u'Hounsou, Djimon', 0.79357359071699218] [u'Hopkins, Anthony', 0.79357359071699218] [u'Highmore, Freddie', 0.79357359071699218] [u'Hawke,

Ethan', 0.79357359071699218] [u'Harris, Ed', 0.79357359071699218] [u'Griffiths, Richard', 0.79357359071699218] [u'Greenwood, Bruce', 0.79357359071699218] [u'Grammer, Kelsey', 0.79357359071699218] [u'Glauser, Michael', 0.79357359071699218] [u'Germann, Greg', 0.79357359071699218] [u'Gamble, Ken', 0.79357359071699218] [u'Franco, James', 0.79357359071699218] [u'Foley, Macka', 0.79357359071699218] [u'Flanagan, Tommy', 0.79357359071699218] [u'Fishburne, Laurence', 0.79357359071699218] [u'Epps, Omar', 0.79357359071699218] [u'Epps, Mike', 0.79357359071699218] [u'El-Balawi, Beans', 0.79357359071699218] [u'Dutton, Charles S.', 0.79357359071699218] [u'Dutcher, Richard', 0.79357359071699218] [u'Duchovny, David', 0.79357359071699218] [u'Dorff, Stephen', 0.79357359071699218] [u'Depp, Johnny', 0.79357359071699218] [u'Dash, Damon', 0.79357359071699218] [u'Daniels, Jeff', 0.79357359071699218] [u'Cusick, Henry Ian', 0.79357359071699218] [u'Curtis, Cliff', 0.79357359071699218] [u'Cross, Joseph', 0.79357359071699218] [u'Cronin, Jonathan', 0.79357359071699218] [u'Costner, Kevin', 0.79357359071699218] [u'Corddry, Rob', 0.79357359071699218] [u'Corbett, John', 0.79357359071699218] [u'Conley, Jack', 0.79357359071699218] [u'Church, Thomas Haden', 0.79357359071699218] [u'Christensen, Hayden', 0.79357359071699218] [u'Cho, John', 0.79357359071699218] [u'Chinlund, Nick', 0.79357359071699218] [u'Cheadle, Don', 0.79357359071699218] [u'Cedric the Entertainer', 0.79357359071699218] [u'Carman, Michael', 0.79357359071699218] [u'Carman', 0.79357359071699218] [u'Carell, Steve', 0.79357359071699218] [u'Campbell, Jeremie', 0.79357359071699218] [u'Byrne, Gabriel', 0.79357359071699218] [u'Butler, Gerard', 0.79357359071699218] [u'Buscemi, Steve', 0.79357359071699218] [u'Burns, Edward', 0.79357359071699218] [u'Burn, Scott', 0.79357359071699218] [u'Burke, Billy', 0.79357359071699218] [u'Brown, Matthew A.', 0.79357359071699218] [u'Brooks, Avery', 0.79357359071699218] [u'Brimley, Wilford', 0.79357359071699218] [u'Bradshaw, Terry', 0.79357359071699218] [u'Bleek, Memphis', 0.79357359071699218] [u'Bell, Jamie', 0.79357359071699218] [u'Banderas, Antonio', 0.79357359071699218] [u'Baldwin, Alec', 0.79357359071699218] [u'Astin, Sean', 0.79357359071699218] [u'Andrieu, Sebastien', 0.79357359071699218] [u'Ice Cube', 0.79357359071699196] [u'Cook, Dane', 0.79357359071699196] [u'Beesley, Max', 0.79357359071699196] [u'Ashker, John', 0.79357359071699196] [u'Procter, Emily', 0.79357359071699107] [u'Long, Nia', 0.79357359071699107] [u'Cruz, Penope', 0.79357359071699107] [u'Yoakam, Dwight', 0.79357359071699107] [u'Levi, Zachary', 0.79357359071699107] [u'Stewart, Kristen', 0.79357359071699007] [u'Beals, Jennifer', 0.79357359071699007] [u'Thieriot, Max', 0.79357359071699007] [u'Bleu, Corbin', 0.79357359071699007] [u'Swank, Hilary', 0.79357359071690448] [u'Staunton, Imelda', 0.79357359071690448] [u'Glenn, Scott', 0.79357359071690448] [u'Dempsey, Patrick', 0.79357359071690448] [u'Paxton, Bill', 0.79357359071637079] [u'McConaughey, Matthew', 0.79357359071637079] [u'Keitel, Harvey', 0.79357359071637079] [u'Bon Jovi, Jon', 0.79357359071637079] [u'Ruffalo, Mark', 0.79357359071518496] [u'Gyllenhaal, Jake', 0.79357359071518496] [u'Edwards, Anthony', 0.79357359071518496] [u'Tachovsk, Helena-Lia', 0.79357359071424682] [u'Gong, Li', 0.79357359071424682] [u'Ulliel, Gaspard', 0.79357359071424682] [u'Thomas, Aaran', 0.79357359071424682] [u'Macy, William H.', 0.79357359071373379] [u'Allen, Tim', 0.79357359071373379] [u'Linney, Laura', 0.79357359071373379] [u'Dhavernas, Caroline', 0.79357359071373379] [u'Phillippe, Ryan', 0.79357359071373379] [u'Cooper, Chris', 0.79357359071373379] [u'Whalin, Justin', 0.79357359070382139] [u'Wayans, Marlon', 0.79357359070382139] [u'Payne, Bruce', 0.79357359070382139] [u'Irons, Jeremy', 0.79357359070382139] [u'Mitchell, Radha', 0.79357359070232247] [u'Hauser, Cole', 0.79357359070232247] [u'Diesel, Vin', 0.79357359070232247] [u'David, Keith', 0.79357359070232247] [u'Miller, Lara Jill', 0.79357359069213751] [u'Seth, Joshua', 0.79357359069213751] [u'Papenbrook, Bob', 0.79357359069213751] [u'Lodge, David', 0.79357359069213751] [u'Wood, Elijah', 0.79357357459871924] [u'Goldblum, Jeff', 0.79357357459871924] [u'Hayek, Salma', 0.79357357459871813] [u'Stevenson, Cynthia', 0.79357357043642685] [u'Spearritt, Hannah', 0.79357357043642685] [u'Muniz, Frankie', 0.79357357043642685] [u'Aaliyah', 0.7935735329657323] [u'Wong, Russell', 0.7935735329657323] [u'Washington, Isaiah', 0.7935735329657323] [u'Nielsen, Connie', 0.78955323414183165] [u'Jackson, Samuel L.', 0.78955323414183165] [u'Daly, Tim', 0.78955323414183165] [u'Travolta, John', 0.78955323413857326] [u'Garner, Jennifer', 0.7865369272305095] [u'Farrell, Colin', 0.7865369272305095] [u'Duncan, Michael Clarke', 0.7865369272305095] [u'Affleck, Ben', 0.7865369272305095] [u'Feore, Colm', 0.67646588182542955] [u'Duke, Bill', 0.67646588182542955] [u'Lawrence, Martin', 0.67646588182216993] [u'Zahn, Steve', 0.67646586570715539] [u'Hu, Kelly', 0.60812554187225809] [u'DMX', 0.60812554187225809] [u'Li, Jet', 0.60812548412099787] [u'Warren, Estella', 0.60812283464878614] [u'Walken, Christopher', 0.60812283464878614] [u"O'Connell, Jerry", 0.60812283464878614] LATENT SEMANTIC 4 [u'Roberts, Julia', 1.0] [u'Didawick, Dawn', 1.0] [u'Finney, Albert', 1.0] [u'Brisbin, David', 1.0] LATENT SEMANTIC 5 [u'Russell, Keri', 1.0] [u'Hines, Cheryl', 1.0] [u'Sisto, Jeremy', 1.0] [u'Fillion, Nathan', 1.0] [u'Preston, Kelly', 0.24373034121993198] [u'Bynes, Amanda', 0.24373034121993198] [u'Atkins, Eileen', 0.24373034121993198] [u'Firth, Colin', 0.24373034121993198] LATENT SEMANTIC 1 [u'Untraceable', 1.0] LATENT SEMANTIC 2 [u'Cradle 2 the Grave', 1.0] [u'Kangaroo Jack', 0.99999919679803662] LATENT SEMANTIC 3 [u'Levity', 1.0] [u'Shanghai

Knights', 0.9128714485185622 8] [u'Sinbad: Legend of the Seven Seas', 0.8901203569907005 8] [u'Final Destination 2', 0.7962184326930790 1] [u'Agent Cody Banks 2: Destination London', 0.7935759191403853 9] [u'Romeo Must Die', 0.7935743288613765 8] [u'Chain of Fools', 0.7935736878815148 3] [u'Wonder Boys', 0.7935736637679613 5] [u'Scream 3', 0.7935736281801397 7] [u'Trois', 0.7935736273155258 3] [u'Breach', 0.7935736241356649] [u'Wild Hogs', 0.7935736240851378 7] [u'Hannibal Rising', 0.7935736240291454 4] [u'Zodiac', 0.7935736239287482] [u'Freedom Writers', 0.7935736237447532 6] [u'Swordfish', 0.7935736237353771] [u'Antitrust', 0.7935736237353618 9] [u'Harold & Kumar Escape from Guantanamo Bay', 0.7935736237353617 8] [u"Nim's Island", 0.7935736237353617 8] [u'Smart People', 0.7935736237353617 8] [u'Jumper', 0.7935736237353617 8] [u'First Sunday', 0.7935736237353617 8] [u'Cloverfield', 0.7935736237353617 8] [u"My Mom's New Boyfriend", 0.7935736237353617 8] [u'United 300', 0.7935736237353617 8] [u'Over the Hedge', 0.7935736237353617 8] [u'Half Light', 0.7935736237353617 8] [u'Ultraviolet', 0.7935736237353617 8] [u"Madea's Family Reunion", 0.7935736237353617 8] [u'Eight Below', 0.7935736237353617 8] [u'When a Stranger Calls', 0.7935736237353617 8] [u'Annapolis', 0.7935736237353617 8] [u'Robots', 0.7935736237353617 8] [u'Be Cool', 0.7935736237353617 8] [u'Bigger Than the Sky', 0.7935736237353617 8] [u'Because of Winn-Dixie', 0.7935736237353617 8] [u'Constantine', 0.7935736237353617 8] [u'Boogeyman', 0.7935736237353617 8] [u'Alone in the Dark', 0.7935736237353617 8] [u'Assault on Precinct 13', 0.7935736237353617 8] [u'Charlie and the Chocolate Factory', 0.7935736237353617 8] [u'Harry Potter and the Prisoner of Azkaban', 0.7935736237353617 8] [u'Godsend', 0.7935736237353617 8] [u'Dawn of the Dead', 0.7935736237353617 8] [u'Spartan', 0.7935736237353617 8] [u'Against the Ropes', 0.7935736237353617 8] [u'Catch That Kid', 0.7935736237353617 8] [u'Torque', 0.7935736237353617 8] [u'Scooby-Doo', 0.7935736237353617 8] [u'Panic Room', 0.7935736237353617 8] [u'Stolen Summer', 0.7935736237353617 8] [u'Ice Age', 0.7935736237353617 8] [u'Full Frontal', 0.7935736237353617 8] [u'All About the Benjamins', 0.7935736237353617 8] [u'State Property', 0.7935736237353617 8] [u'Rollerball', 0.7935736237353617 8] [u'Collateral Damage', 0.7935736237353617 8] [u'Texas Rangers', 0.7935736237353617 8] [u'Clubhouse Detectives in Big Trouble', 0.7935736237353617 8] [u'Final Fantasy: The Spirits Within', 0.7935736237353617 8] [u'Brigham City', 0.7935736237353617 8] [u'15 Minutes', 0.7935736237353617 8] [u'Carman: The Champion', 0.7935736237353617 8] [u'3000 Miles to Graceland', 0.7935736237353617 8] [u'Sweet November', 0.7935736237353617 8] [u'Hannibal', 0.7935736237353617 8] [u'Enemy at the Gates', 0.7935736237349946 3] [u"Big Momma's House 2", 0.7935736237234114 5] [u'Bandidas', 0.7935736237234095 6] [u'Hot Fuzz', 0.7935736236051833 5] [u'U-571', 0.7935736235470707 3] [u'Stomp the Yard', 0.7935736233270768 2] [u'Epic Movie', 0.7935736232761817 5] [u'Dungeons & Dragons', 0.7935736197442561 1] [u'Pitch Black', 0.7935736192900567 6] [u'Digimon: The Movie', 0.7935736162037219 1] [u'Starsky & Hutch', 0.7935735146980050 8] [u'Basic', 0.7895532668238803 8] [u'Daredevil', 0.7865369540230297] [u'National Security', 0.6764660532108662 8] LATENT SEMANTIC 4 [u'Erin Brockovich', 1.0] LATENT SEMANTIC 5 [u'Waitress', 1.0] [u'What a Girl Wants', 0.2437303394053368 9] LATENT SEMANTIC 1 [2008L, 1.0] LATENT SEMANTIC 2 [2003L, 1.0] [2004L, 2.109871887598424e-05] LATENT SEMANTIC 3 [2006L, 3.6563129064882167e-06] [2005L, 3.6559047260802356e-06] [2002L, 3.6559047260694639e-06] [2001L, 3.6558895159654496e-06] LATENT SEMANTIC 4 [2000L, 1.0] LATENT SEMANTIC 5 [2007L, 1.0]

2d

d = part2.compute_Semantics_2d()
Tag Bins LATENT SEMANTIC 1: [u'action', 0.3333333333333337] LATENT SEMANTIC 2: [u'over the top', 0.4249616980227276] [u'hacking', 0.4249616980227276] [u'cool', 0.4249616980227276] [u'nudity (topless - notable)', 0.4249616980227276] [u'hackers', 0.4249616980227276] [u'remake', 0.4249616980186614] [u'nudity (topless - brief)', 0.4249616980186614] [u'zombie', 0.4249616980186614] [u'directorial debut', 0.4249616980186614] [u'zombies', 0.4249616980186614] [u'drugs', 0.3698037736294186 3] [u'espionage', 0.3698037736294186 3] [u'geeks', 0.3698037736294186 3] [u'comedy', 0.3698037736294186 3] [u'history', 0.3698037736294186 3] [u'not funny', 0.3698037736294186 3] [u'court', 0.3698037736294186 3] [u'san francisco', 0.3698037736294186 3] [u'family', 0.3698037736294186 3] [u'predictable', 0.3698037736294186 3] [u'violent', 0.3698037736294186 3] [u'dreamworks', 0.3698037736294186 3] [u'buddy movie', 0.3698037736294186 3] [u'based on book', 0.3698037736294186 3] [u'hilarious', 0.3698037736294186 3] [u'murder', 0.3698037736294186 3] [u'pirates', 0.3698037736294186 3] [u'based on a tv show', 0.3698037736294186 3] [u'nudity (rear)', 0.3698037736294186 3] [u'nudity (topless)', 0.3698037736294186 3] [u'slapstick', 0.3698037736294186 3] [u'best war films', 0.3698037736294186 3] [u'musicians', 0.3698037736294186 3] [u'interesting', 0.3698037736294186 3] [u'wwii', 0.3698037736294186 3] [u'terrorism', 0.3698037736294186 3] [u'dumb but funny', 0.3698037736294186 3] [u'war', 0.3698037736294186 3] [u'swashbuckler', 0.3698037736294186 3] [u'england', 0.3698037736294186 3] [u'crime', 0.3698037736294186 3] [u'world war ii', 0.3698037736294186 3] [u'pointless', 0.3698037736294186 3] [u'motorcycle', 0.3698037736294186 3] [u'gore', 0.3698037736294186 3] [u'boxing', 0.3698037736294186 3]

[u'cgi', 0.32019271186488329] [u'sci-fi', 0.32019271186488329] [u'fish', 0.32019271186488329] [u'anime', 0.32019271186488329] [u'computer animation', 0.32019271186488329] [u'video game adaptation', 0.32019271186488329] [u'hannibal lecter', 0.30798122919424159] [u'cannibalism', 0.30798122919424159] [u'atmospheric', 0.30798122919424159] [u'boring', 0.30798122919424159] [u'psychology', 0.30798122919424159] [u'serial killer', 0.30798122919424159] [u'bad acting', 0.27012665477368791] [u'marvel', 0.27012665477368791] [u'dark', 0.27012665477368791] [u'superhero', 0.27012665477368791] [u'heroine in tight suit', 0.27012665477368791] [u'comic book', 0.27012665477368791] LATENT SEMANTIC 3: [u'sequel', 0.57587390199510324] [u'witch', 0.57587390199510324] [u'fantasy', 0.57587390199510324] [u'time travel', 0.57587390199510324] [u'wizards', 0.57587390199510324] [u'harry potter', 0.57587390199510324] [u'magic', 0.57587390199510324] [u'franchise', 0.57587390199510324] [u'cartoon', 0.34784566280483703] [u'animated', 0.34784566280483703] [u'redemption', 0.34784566280483703] [u'animation', 0.34784566280483703] [u'disney', 0.34784566280483703] [u'talking animals', 0.34784566280483703] LATENT SEMANTIC 4: LATENT SEMANTIC 5:
Movie Bins LATENT SEMANTIC 1 [u'Ice Age', 0.55955214348723326] [u'Agent Cody Banks 2: Destination London', 0.5354208458503823] [u'Antitrust', 0.5354208458503823] [u'National Security', 0.5354208458503823] [u'Texas Rangers', 0.5354208458503823] [u'Sweet November', 0.5354208458503823] [u'Clubhouse Detectives in Big Trouble', 0.5354208458503823] [u'Starsky & Hutch', 0.5354208458503823] [u'Scooby-Doo', 0.5354208458503823] [u'15 Minutes', 0.5354208458503823] [u'What a Girl Wants', 0.5354208458503823] [u'Full Frontal', 0.5354208458503823] [u'Collateral Damage', 0.5354208458503823] [u'Sinbad: Legend of the Seven Seas', 0.5354208458503823] [u'Shanghai Knights', 0.5354208458503823] [u'Panic Room', 0.5354208458503823] [u'Enemy at the Gates', 0.5354208458503823] [u'Godsend', 0.5354208458503823] [u'Torque', 0.5354208458503823] [u'Final Destination 2', 0.5354208458503823] [u'Against the Ropes', 0.5354208458503823] [u'Final Fantasy: The Spirits Within', 0.5] [u'Swordfish', 0.4659089771337343] [u'Dawn of the Dead', 0.46590897713125023] LATENT SEMANTIC 2 [u'Harry Potter and the Prisoner of Azkaban', 0.34930724022791021] LATENT SEMANTIC 3 [u'Daredevil', 0.6800626618573371] [u'Hannibal', 0.28651009200903405] LATENT SEMANTIC 4 LATENT SEMANTIC 5
Rating Bins LATENT SEMANTIC 1 [3L, 0.33333333333333331] LATENT SEMANTIC 2 LATENT SEMANTIC 3 LATENT SEMANTIC 4 [4L, 0.5] [5L, 0.5] LATENT SEMANTIC 5

9.2.2 Task 3

Given a set of seed actors, the related actors are printed with descending order of weights(pagerank)
part3.compute_Semantics_3a(set([1860883,486691,1335137,901175])) ==========
[90, 44, 122, 157, 75, 20, 79, 56, 53, 34, 30, 65, 13, 7] Seed Actors: Cross, Joseph Radcliffe, Daniel Gyllenhaal, Jake Lillard, Matthew (901175L, u'Gyllenhaal, Jake', 0.18123894070990795) (486691L, u'Cross, Joseph', 0.18123894070990793) (1335137L, u'Lillard, Matthew', 0.18075440318719913) (1860883L, u'Radcliffe, Daniel', 0.18075439537711688) (720541L, u'Fishburne, Laurence', 0.00061394070990796845) (312142L, u'Burn, Scott', 0.00061394070990796834) (749056L, u'Franco, James', 0.00061394070990796823) (602906L, u'Dorff, Stephen', 0.00061394070990796823) (565786L, u'Depp, Johnny', 0.00061394070990796823) (408123L, u'Cho, John', 0.00061394070990796823) (377600L, u'Cedric the Entertainer', 0.00061394070990796823) (642688L, u'Edwards, Anthony', 0.00061394070990796812) (263611L, u'Bradshaw, Terry', 0.00061394070990796812) (133985L, u'Banderas, Antonio', 0.00061394070990796812)
part3.compute_Semantics_3b(set([1860883,486691,1335137,901175])) Seed Actors: Cross, Joseph Radcliffe, Daniel Gyllenhaal, Jake Lillard, Matthew (1860883L, u'Radcliffe, Daniel', 0.18214285714285713) (1335137L, u'Lillard, Matthew', 0.18214285714285713) (901175L, u'Gyllenhaal, Jake', 0.18186177248677252) (486691L, u'Cross, Joseph', 0.18159567611605865) (608635L, u'Downey Jr., Robert', 0.01011904761904762) (3704908L, u'Shaw, Fiona', 0.010119047619047618) (2946264L, u'Ferris, Pam', 0.010119047619047618) (2746767L, u'Cardellini, Linda', 0.010119047619047618) (1839848L, u'Prinze Jr., Freddie', 0.010119047619047618) (878356L, u'Griffiths, Richard', 0.010119047619047618) (2997171L, u'Gellar, Sarah Michelle', 0.010119047619047616) (3251912L, u'Lane, Diane', 0.009845348494884819 2) (925935L, u'Hanks, Colin', 0.0098379629629629633) (642688L, u'Edwards, Anthony', 0.0098379629629629633)

9.2.3 Task 4

Model 1 Execution command : python user_movie_matrix.py 1027 Sample Output for MODEL 1: ——Watched movies——- ('Pitch Black', 'Thriller') ('Romeo Must Die', 'Crime|Action|Thriller') ('U-571', 'Action|Thriller') ('Dawn of the Dead', 'Action|Thriller|Drama') ('Constantine', 'Action|Thriller') ('Cloverfield', 'Action|Thriller') ——————————————————————————————— ——- ———-Genre weights based in user rating of movie and movie year—— [('Thriller', 9.435909),

('Action', 9.329804), ('Crime', 1.3479869999999998), ('Drama', 0.898658), ('Adventure', 1e-06)] ——————————————- ——-Recommended movies—— ('Scream 3', 'Thriller') ('Trois', 'Thriller') ('Antitrust', 'Thriller') ('Hannibal', 'Thriller') ('3000 Miles to Graceland', 'Action|Thriller')

Model 2 Execution command : python user_movie_matrix2.py 1027 Sample Output for MODEL 2: General preprocessing done. Starting preprocessing for genre tag vectors... done ——Watched movies———- ('Pitch Black', 'Thriller') ('Romeo Must Die', 'Crime|Action|Thriller') ('U-571', 'Action|Thriller') ('Dawn of the Dead', 'Action|Thriller|Drama') ('Constantine', 'Action|Thriller') ('Cloverfield', 'Action|Thriller') ——————————————- ——-Top 5 Recommended movies—— ('Dungeons & Dragons', 'Adventure') ('Bigger Than the Sky', 'Drama') ('Robots', 'Adventure') ('Because of Winn-Dixie', 'Drama') ('Final Destination 2', 'Thriller')

Model 2 Execution command : python user_movie_matrix2.py 1027 Sample Output for MODEL 2 (k = 20): ——Watched movies———- Movie details: Name: Constantine; Genre Action,Thriller Movie details: Name: Romeo Must Die; Genre Crime,Action,Thriller Movie details: Name: U-571; Genre Action,Thriller Movie details: Name: Dawn of the Dead; Genre Action,Thriller,Drama Movie details: Name: Pitch Black; Genre Thriller Movie details: Name: Cloverfield; Genre Action,Thriller

——-Top 5 Recommended movies—— Movie details: Name: Final Fantasy: The Spirits Within; Genre Adventure Movie details: Name: Jumper; Genre Thriller,Adventure,Drama Movie details: Name: Final Destination 2; Genre Thriller Movie details: Name: Basic; Genre Thriller,Drama Movie details: Name: Over the Hedge; Genre Adventure