PHASE 2 GROUP 10 FALL 2017

GROUP MEMBERS:

- Abhishek Thorat
- Krishna Chaitanya Bejjipuram
- Raviteja Upmaka
- Sachin Sundar
- Shiva Tejaswi Rampally
- Shubam Gondane

ABSTRACT:

This project is about developing a recommender engine on the Movie+Lens database. We will be performing dimensionality reduction i.e., data from high dimension is reduced to low dimension, finding similarity matrices, finding the top latent features, finding the most similar data for an object for specific user input, performing similarity functions, performing decompositions like Singular Value Decomposition, Principal Component Analysis, Latent Dirichlet Allocation on data matrices and, CP Decomposition on tensors. We implemented Personalized Page Ranking/Random Walk with Restarts, where we give object-object similarity matrix as the seeds and find the 10 most related objects for the given seed.

KEYWORDS:

- SVD – Singular Value Decomposition
- PCA – Principal Component Analysis
- LDA – Latent Dirichlet Allocation
- CP Decomposition
- Personalized Page Ranking/ Random Walk with Restarts
- TF – IDF
- Similarity Matrix
- Tensor

INTRODUCTION:

TERMINOLOGY:

- Singular Value Decomposition: The singular value decomposition of a matrix A is the factorization of A into the product of three matrices A = UDV T where the columns of U and V are orthonormal, and the matrix D is diagonal with positive real entries.
- Principal Component Analysis: Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components (or sometimes, principal modes of variation).
- Latent Dirichlet Allocation: latent Dirichlet allocation (LDA), a generative probabilistic model for collections of discrete data such as text corpora. LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of topics.
- CP Decomposition: CP decomposition is generalization of SVD for tensors.
- Page Rank: Page Rank (PR) measures stationary distribution of one specific kind of random walk that starts from a random vertex and in each iteration, with a predefined probability p, jumps to a random vertex, and with probability1-p follows a random outgoing edge of the current vertex
- Personalized Page Ranking: Personalized Page Rank (PPR) is the same as PR other than the fact that jumps are back to one of a given set of starting vertices. In a way, the walk in PPR is biased towards (or personalized for) this set of starting vertices and is more localized compared to the random walk performed in PR.
- Tensor: Tensors are multidimensional arrays (generalization of one dimensional array)

GOAL DESCRIPTION:

We need to do the following tasks for MOVIE LENS+IMDB data set.

TASK 1:

- **1a:** Implement a program which, given a genre, identifies and reports the top-4 latent semantics/topics using
  ∗ PCA in TF-IDF space of tags,
  ∗ SVD in TF-IDF space of tags, and
  ∗ LDA in the space of tags.
- **1b:** Implement a program which, given a genre, identifies and reports the top-4 latent semantics/topics using
  ∗ PCA in TF-IDF space of actors,
  ∗ SVD in TF-IDF space of actors, and
  ∗ LDA in the space of actors.
- **1c:** Implement a program which, given an actor, finds and ranks the 10 most similar actors by comparing actors'
  ∗ TF-IDF tag vectors,
  ∗ top-5 latent semantics (PCA, SVD, or LDA) in the space of tags.
- **1d:** Implement a program which, given a movie, finds and ranks the 10 most related actors who have not acted in the movie, leveraging the given movie's
  ∗ TF-IDF tag vectors,
  ∗ top-5 latent semantics (PCA, SVD, or LDA) in the space of tags.

TASK 2:

- **2a:** Implement a program which
  1. creates an actor-actor similarity matrix (using tag vectors),
  2. performs SVD on this actor-actor similarity matrix,
  3. reports the top-3 latent semantics, in the actor space, underlying this actor-actor similarity matrix, and
  4. partitions the actors into 3 non-overlapping groups based on their degrees of memberships to these 3 semantics.
- **2b:** Implement a program which
  1. creates a coactor-coactor matrix based on co-acting relationships (recording the number of times two actors played acted in the same movie),
  2. performs SVD on this coactor-coactor matrix,
  3. reports the top-3 latent semantics, in the actor space, underlying this coactor-coactor matrix, and
  4. partitions the actors into 3 non-overlapping groups based on their degrees of memberships to these 3 semantics.

- **2c:** Implement a program which
  1. creates an actor-movie-year tensor, where the tensor contains 1 for any actor-movie-year triple if the given actor played in the stated movie and the movie was released in the stated year (the tensor contains 0 for all other triples)
  2. performs CP on this actor-movie-year tensor with target rank set to 5,
  3. reports the top-5 latent
     - actor
     - movie
     - year

  semantics underlying this tensor, and
  4. partitions
     - actors
     - movies
     - years

  into 5 non-overlapping groups based on their degree of memberships to these 5 semantics.
- **2d:** Implement a program which
  1. creates tag-movie-rating tensor, where the tensor contains 1 for any tag-movie-rating triple if the given tag was assigned to a movie by at least one user and the movie has received an average rating lower than or equal to the given rating value (the tensor contains 0 for all other triples).
  2. performs CP on this actor-movie-rating tensor with target rank set to 5,
  3. reports the top-5 latent semantics in terms of
     - tag
     - movie
     - rating

  memberships underlying this tensor, and
  4. partitions
     - tag
     - movies
     - ratings

  into 5 non-overlapping groups based on their degree of memberships to these 5 semantics.

TASK 3:

- **3a:** Implement a program which
  1. creates an actor-actor similarity matrix (using tag vectors),
  2. given a set, S, of "seed" actors (indicating the user's interest), identifies the 10 most related actors to the actors in the given seed set using Random Walk with ReStarts (RWR, or Personalized PageRank, PPR) score. See J.-Y. Pan, H.-J. Yang, C. Faloutsos,

and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In KDD, pages 653658, 2004.

- **3b:** Implement a program which
  1. creates a coactor- coactor matrix based on the number of movies two actors acted in together,
  2. given a set, S, of "seed" actors (indicating the user's interest), identifies the 10 most related actors to the actors in the given seed set using RWR.

TASK 4:

- Implement a program which
  1. given all the information available about the set of movies a given user has watched, recommends the user 5 more movies to watch.

ASSUMPTIONS:

In task 2d we have tag-movie-rating triple, so here we are only considering the movies for which a particular user has given a rating as well as a tag.

IMPLEMENTATION/DESCRIPTION OF PROPOSED SOLUTION:

Task 1

Task 1a – In this task, we have taken genre as input and computed the top 4 latent semantics using various dimensionality reduction algorithms like PCA, SVD and LDA. The feature vectors represented are tags. TF-IDF was computed for each tag which is used as a input matrix to SVD and covariance matrix for PCA. The input matrix for LDA will be the count of each tag in each movie. This is represented as a matrix and given as input to the LDA algorithm. The output of the algorithms consists of matrices U,S and Vt. We take the top four rows from the matrix Vt representing the top 4 latent semantics.

Mapping genre -> Movies ->Tags

PCA

We have calculated the covariance matrix initially and passed the covariance matrix to the SVD function.

SVD

We have calculated the object feature similarity matrix, here representing the movie- tag similarity matrix to the SVD function which gives 3 matrices namely U, S and Vt. We then computed the top 4 latent semantics by taking the top 4 rows of the Vt matrix.

Output of SVD function – U, S , VT

The V matrix represents the latent semantics in terms of latent features.

The S matrix represents the ranking of the latent features.

The U matrix represents the objects in terms of latent semantics.

LDA

The input matrix for LDA will be the count of each tag in each movie. This is represented as a matrix and given as input to the LDA algorithm

Task 1b – In this task, we have taken genre as input and computed the top 4 latent semantics using various dimensionality reduction algorithms like PCA, SVD and LDA. The feature vectors represented are actors. TF-IDF was computed for each tag which is used as a input matrix to SVD and covariance matrix for PCA. The input matrix for LDA will be the count of each tag in each movie. This is represented as a matrix and given as input to the LDA algorithm. The output of the algorithms consists of matrices U,S and Vt. We take the top four rows from the matrix Vt representing the top 4 latent semantics.

Mapping genre -> Movies ->Actors

PCA

We have calculated the covariance matrix initially and passed the covariance matrix to the SVD function.

SVD

We have calculated the object feature similarity matrix, here representing the movie-actor similarity matrix to the SVD function which gives 3 matrices namely U, S and Vt. We then computed the top 4 latent semantics by taking the top 4 rows of the Vt matrix.

Output of SVD function – U, S , VT

The V matrix represents the latent semantics in terms of latent features.

The S matrix represents the ranking of the latent features.

The U matrix represents the objects in terms of latent semantics.

LDA-

The input matrix for LDA will be the count of each tag in each movie. This is represented as a matrix and given as input to the LDA algorithm.

Task 1c – In this task, we take a particular actor as input and get the top 10 similar actors to the given input actor. The matrix M will be an actor tag matrix with the tags representing feature vectors. The matrix will consist of TF-IDF values of each tag corresponding to that particular

actor. We then take the input actor and compare it with all the actor vectors in the matrix M using Cosine similarity to find the top 10 similar actors.

Mapping Actor->movies->tags

We then take the Matrix M as an input to the dimensionality reduction algorithm SVD.

SVD

Input – Matrix M representing actor – tag relationship.

Output – U, S , Vt

We then take the top 5 columns from the U matrix representing the top 5 latent semantics. In order to find the top 10 similar actors, we now compare the input actor vector with the matrix obtained after getting the top five latent semantics. The algorithm used to find the top 10 similar actors is Cosine Similarity.

Cosine Similarity -

$$\cos(x, y) = \frac{x \cdot y}{\|x\| \|y\|}$$

PCA - Input – Covariance matrix representing actor – tag relationship.

Output – U, S , Vt

We pass the input covariance matrix to the SVD function. We then take the top 5 columns from the U matrix representing the top 5 latent semantics. In order to find the top 10 similar actors, we now compare the input actor vector with the matrix obtained after getting the top five latent semantics. The algorithm used to find the top 10 similar actors is Cosine Similarity.

LDA

We took the actor-tag matrix as input and passed the actor tag matrix as an input to the LDA algorithm. We then take the top 5 latent semantics from the decomposition matrix and then compare the object – latent semantic matrix with the given input actor vector to find the 10 most similar actors.

Task 1d – In this task, we take a particular movie as input and get the top 10 similar actors which have not acted in the movie. The matrix M will be an actor tag matrix with the tags

representing feature vectors. The matrix will consist of TF-IDF values of each tag of each actor. We then take the input actor and compare it with all the actor vectors in the matrix M using Cosine similarity to find the top 10 similar actors who have not acted in the movie.

Mapping movie->actor->tags

We the take the Matrix M as an input to the dimensionality reduction algorithm SVD.

SVD

Input – Matrix M representing actor – tag relationship.

Output – U, S , Vt

We then remove the actors corresponding to the inputted movie from the U matrix obtained to obtain a matrix R. We then take the top 5 columns from the U matrix representing the top 5 latent semantics. In order to find the top 10 similar actors, we now compare the input actor vector with the matrix obtained after getting the top five latent semantics. The algorithm used to find the top 10 similar actors is Cosine Similarity.

PCA

Input – Covariance matrix representing actor – tag relationship. We pass the covariance matrix into the SVD function.

Output – U, S , Vt

We then remove the actors corresponding to the inputted movie from the U matrix obtained to obtain a matrix R . We then take the top 5 columns from the U matrix representing the top 5 latent semantics. In order to find the top 10 similar actors, we now compare the input actor vector with the matrix obtained after getting the top five latent semantics. The algorithm used to find the top 10 similar actors is Cosine Similarity.

LDA

We took the actor-tag matrix as input and passed the actor tag matrix as an input to the LDA algorithm. We then remove the actors corresponding to the inputted movie from the U matrix obtained to obtain a matrix R .We then take the top 5 latent semantics from the decomposition matrix and then compare the object – latent semantic matrix with the given input actor vector to find the 10 most similar actors.

TASK 2:

Task 2a

To create an actor-actor similarity matrix we need to map the actors to tags first. So, for each actor we are finding out the relevant tags by mapping the actors to movies first and then finding tags associated with the movies.

Mapping: Actors → Movies → Tags

So, we get the actor-tag matrix let's denote this matrix by D, where the values in the matrix are essentially the TF-IDF weights of the tags for each actor. In order to find the actor-actor similarity matrix we take the transpose of actor-tag matrix D^T and take a dot product of D and D^T.

actor-actor similarity matrix = D. D^T

To find the Singular Value Decomposition (SVD), we are using the linear algebra module from the scipy package of python. The linear algebra module has a SVD decomposition function which given an input matrix would return us the left factor matrix denoted by U, right factor matrix denoted by Vh and the core matrix denoted by s.

U, s, Vh = svd (actor-actor similarity matrix)

We need to report the top-3 latent semantics in the actor space. Since our input matrix is actor-actor similarity matrix we can use either U or Vh to report the latent semantics. The core matrix s is already sorted in decreasing order of Eigen values, so we can use the first 3 columns of U matrix to report the latent semantics.

So, we are mapping the actorids to the U matrix and each latent semantic we are printing the <actorid, weight> pairs sorted in decreasing order of weights. To partition the actors into 3 non-overlapping groups we are using the KMeans clustering method. The U matrix essentially represents the objects (in our case the actors) in terms of k latent features. Since we only use the top-3 latent semantics the actors are represented using the 3 latent features.

actor → (latent_feature1, latent_feature2, latent_feature3)

KMeans helps to cluster the similar objects into three groups in the 3-dimensional vector space represented by these latent features. We then report the actors clustered in each of these three groups.

Task 2b:

To create the coactor-coactor matrix we need to map the actors to the movies and then use that information to find out the number of times two actors played in the same movie.

Mapping: Actors → Movies

We start by creating a set of movies each actor has played. Next, we take intersection of the sets of movies for pair of actors and essentially the number of movies in this set gives us the number of movies in which given pair of actors have acted together. We then create a coactor-coactor matrix by filling the elements of the matrix by the count of the movies in the intersection set. The rows and the columns of this matrix correspond to the actors.

coactor-coactor[actor1][actor2] = count (actor1 {movies} ∩ actor2 {movies})

To find the Singular Value Decomposition (SVD), we are using the linear algebra module from the scipy package of python. The linear algebra module has a SVD decomposition function which given an input matrix would return us the left factor matrix denoted by **U**, right factor matrix denoted by **Vh** and the core matrix denoted by **s.**

U, s, Vh = svd (coactor-coactor matrix)

We need to report the top-3 latent semantics in the actor space. Since our input matrix is coactor-coactor similarity matrix we can use either **U** or **Vh** to report the latent semantics. The core matrix **s** is already sorted in decreasing order of Eigen values, so we can use the first 3 columns of **U** matrix to report the latent semantics.

So, we are mapping the actorids to the **U** matrix and each latent semantic we are printing the **<actorid, weight>** pairs sorted in decreasing order of weights. To partition the actors into 3 non-overlapping groups we are using the KMeans clustering method. The **U** matrix essentially represents the objects (in our case the actors) in terms of **k** latent features. Since we only use the top-3 latent semantics the actors are represented using the 3 latent features.

actor → (latent_feature1, latent_feature2, latent_feature3)

KMeans helps to cluster the similar objects into three groups in the 3-dimensional vector space represented by these latent features. We then report the actors clustered in each of these three groups.

Task 2c

Here we have to create a 3-mode tensor where the modes are actor, movie and year. The tensor can be represented using the n-dimensional array of numpy module in python. The value for the triple – actor-movie-year triple will be 1 if an actor acted in a movie in a specific year. We denote the 3-D array by ‘T1’.

T1 → (Actor, Movie, Year)

We are using a 3-dimensional array to represent the tensor T1. Before applying the decomposition, we are converting the 3-D array to a tensor representation denoted by T, necessary for the decomposition function. We are using the scikit-tensor library for tensor decomposition.

T = dtensor(T1)

The above function converts the 3-D array to appropriate representation. Next, we have to do the tensor decomposition of the 3-mode tensor using the Canonical/ Parafac Decomposition [1,2]. The rank of the resultant decomposed tensor is set to 5. The factor matrices are initialized randomly.

P = cp.als (T, 5, init = 'random')

This gives us the decomposed tensor in the Kruskal [3] format which can be represented as follows.

$X \in IRn1 \times n2 \times n3$

$X = [[\lambda; F, G, H]]$

where $\lambda \in IRr$, $F \in IRn1 \times r$, $G \in IRn2 \times r$, and $H \in IRn3 \times r$

F, G and R are the factor matrices for each of the modes which in our case would be for Actor, Movie and Year. The rank is represented by r. The lambda λ corresponds to the strength of groupings represented by the super-diagonal of the core tensor.

We are mapping the actorids to the factor matrix for actors, similarly for other factor matrices.

For each factor matrix represents the object mode in terms of latent semantic. So, for each latent semantic in a given factor matrix we are reporting the <object, weight> pairs sorted in decreasing order of weights.

Next task for the factor matrix for actors, is to partition the actors according to the degree of memberships to these 5 semantics. We are using the KMeans clustering method to partition the actors. KMeans helps to cluster the similar objects into five non-overlapping groups in the 5-dimensional vector space represented by these latent semantics. We then report the actors clustered in each of these five groups.

Similarly, the movies and years can be partitioned into 5 non-overlapping groups respectively

Task 2d:

Here we have to create a 3-mode tensor where the modes are tag, movie and rating. The tensor can be represented using the n-dimensional array of numpy module in python. The value for the triple–tag-movie-rating triple will be 1 if the given tag was assigned to a movie by at least one user and the movie has received an average rating lower than or equal to the given rating value. So, for each movie we calculate the average rating using the given ratings by the users.

Average_rating (movie) = Σ ratings by users / total number of users

So, we only consider the rating values which are higher than or equal to the calculated average rating for a given user, and then use these users to find out the tags related to the movies.

U1 → users who gave rating >= average_rating

Mapping: U1(ratings) → movies → tags

We denote the final 3-D array by '**T1**'.

T1 → (Tag, Movie, Rating)

We are using a 3-dimensional array to represent the tensor **T1**. Before applying the decomposition, we are converting the 3-D array to a tensor representation denoted by **T**, necessary for the decomposition function. We are using the scikit-tensor library for tensor decomposition.

T = dtensor(T1)

The above function converts the 3-D array to appropriate representation. Next, we have to do the tensor decomposition of the 3-mode tensor using the Canonical/ Parafac Decomposition [1,2]. The rank of the resultant decomposed tensor is set to 5. The factor matrices are initialized randomly.

P = cp.als (T, 5, init = 'random')

This gives us the decomposed tensor in the Kruskal [3] format which can be represented as follows.

$X \in IRn1 \times n2 \times n3$

$X = [[\lambda; F, G, H]]$

where $\lambda \in IRr$, $F \in IRn1 \times r$, $G \in IRn2 \times r$, and $H \in IRn3 \times r$

F, G and R are the factor matrices for each of the modes which in our case would be for Tag, Movie and Rating. The rank is represented by **r**. The lambda λ corresponds to the strength of groupings represented by the super-diagonal of the core tensor.

We are mapping the actorids to the factor matrix for Tags, similarly for other factor matrices.

For each factor matrix represents the object mode in terms of latent semantic. So, for each latent semantic in a given factor matrix we are reporting the **<object, weight>** pairs sorted in decreasing order of weights.

Next task for the factor matrix for actors, is to partition the actors according to the degree of memberships to these 5 semantics. We are using the KMeans clustering method to partition the actors. KMeans helps to cluster the similar objects into five non-overlapping groups in the 5-dimensional vector space represented by these latent semantics. We then report the actors clustered in each of these five groups.

Similarly, the movies and years can be partitioned into 5 non-overlapping groups respectively.

TASK 3:

For task 3, we will use the actor-actor matrix or cofactor-coactor matrix as the transition matrix for random walk with restarts algorithm. The given actors are considered as the seeds and a teleportation matrix for actors is create using the seeds. Equal weights are given to the seeds in the teleportation matrix.

Task 3a:

For each actor, find the set of movies acted and using the set of movies, find the set of tags for the movies. So, for an actor, there are a set of tags. Calculate TF-IDF for the tags of each actor. To calculate the TF,

TF = no. of times a tag has appeared for an actor / total no. of tags associated with actor

Since, some tags might appear many times in a movie because different users can tag the same movie, use timestamp value to find the recent tag in those many appearances. Timestamp weight is calculate using,

time-weight = timestamp of that tag / sum of all the timestamps of the tags in the movie

Add the time-weight to the TF to get the time-weighted TF. Time weight is always added to the TF because the TF is between the tags and same tags needs some differentiation in weightage. It

is not added to IDF because IDF is finds in how many movies it has appeared and time relevance is not needed there to differentiate the tags. To compute the IDF of tag,

IDF = logarithm (total no. of actors / no. of actors that are associated with the tag)

In this task, actor and movies have a rank relation. The lesser the rank of an actor in a movie, the higher the weightage for that actor. To find the weightage for each rank, use reciprocal rank method. For a given actor, the rank weight is,

rank weight = 1 / rank of the actor in the movie

The reciprocal rank method makes sure that actor in a movie with lower rank gets higher weight than in a movie with higher rank.

So, the final formula is,

TF-IDF weight = (TF + time-weight) * IDF + rank weight if given model is TF-IDF

Time weight and rank weight are added because we are joining different kind of ranks and any of the rank join method is should be used. This method because it does not change the order of weights after adding the weights.

Using the values of TF-IDF values of tags for all the actors, create an actor-tag matrix where the rows are actors and columns are tags. The values are the TF-IDF value of the actor-tag. To calculate the actor-actor similarity matrix, multiply the actor-tag matrix with transpose of the actor-tag matrix.

actor-actor = (actor-tag) * $(\text{actor-tag})^{T}$

In Random walk with restart, start with the transition matrix(T) and teleportation matrix (seed matrix, s) and find the pagerank matrix(p). The logic of random walk is, jump from a node to another randomly with some proprability. Considering that the walker always jumps, with the remaining probability, jump to one of the teleportation node(here it is the seed nodes). So based on the amount of time the walker spends on each node, the nodes are ranked using pagerank matrix. The pagerank formula is,

p = ((1 – alpha) * T *p) + (alpha * s)

Here, the nodes are the actors. The teleportation nodes are the seed actors. The actor-actor matrix is the transition matrix. Do random walk until the pagerank matrix converges, ie, sum of previous pagerank matrix and the sum of current pagerank matrix difference value is less than 0.001.

Sort the pagerank matrix and display the top 10 actors and their pagerank value. These 10 actors are related to the seed actors.

Task 3b:

To create the coactor-coactor matrix, map the actors to the movies and then use that information to find out the number of times two actors played in the same movie. Find the set of movies the actor has acted in. Next, find the intersection of the sets of movies for pair of actors and essentially the number of movies in this set gives the number of movies in which given pair of actors have acted together. Then create a coactor-coactor matrix by filling the elements of the matrix by the count of the movies in the intersection set. The rows and the columns of this matrix correspond to the actors.

coactor-coactor[actor1][actor2] = count (actor1 {movies} actor2 {movies})

In Random walk with restart, start with the transition matrix(T) and teleportation matrix (seed matrix, s) and find the pagerank matrix(p). The logic of random walk is, jump from a node to another randomly with some proprability. Considering that the walker always jumps, with the remaining probability, jump to one of the teleportation node(here it is the seed nodes). So based on the amount of time the walker spends on each node, the nodes are ranked using pagerank matrix. The pagerank formula is,

p = ((1 – alpha) * T *p) + (alpha * s)

Here, the nodes are the actors. The teleportation nodes are the seed actors. The coactor-coactor matrix is the transition matrix. Do random walk until the pagerank matrix converges, ie, sum of previous pagerank matrix and the sum of current pagerank matrix difference value is less than 0.001.

Sort the pagerank matrix and display the top 10 actors and their pagerank value. These 10 actors are related to the seed actors.

TASK 4:

Given a user, find the set of movies he has watched. To find the movies related to the movies the user has watched, use random walk with restart algorithm. The movie-movie matrix is the transition matrix and the set of movies the user has watched are the seeds. Calculate the movie-movie matrix. For each movie, find the set of tags associated with it. Calculate the TF-IDF for the tags with respect to movies. From this, create the movie-tag matrix with TF-IDF values. The TF-IDF calculation is,

TF = no. of times a tag has appeared in movie / total no. of tags associated with movie

time-weight = timestamp of that tag / sum of all the timestamps of the tags in the movie

IDF = logarithm (total no. of movies / no. of movies that are associated with the tag)

TF-IDF weight = (TF + time-weight) * IDF + rank weight

In Random walk with restart, start with the transition matrix(T) and teleportation matrix (seed matrix, s) and find the pagerank matrix(p). The logic of random walk is, jump from a node to another randomly with some proprability. Considering that the walker always jumps, with the remaining probability, jump to one of the teleportation node(here it is the seed nodes). So, based on the amount of time the walker spends on each node, the nodes are ranked using pagerank matrix. The pagerank formula is,

p = ((1 – alpha) * T *p) + (alpha * s)

Here, the nodes are the movies. The teleportation nodes are the seed movies. The movie-movie matrix is the transition matrix. Do random walk until the pagerank matrix converges, ie, sum of previous pagerank matrix and the sum of current pagerank matrix difference value is less than 0.001.

Sort the pagerank matrix and display the top 5 movies and their pagerank value. These 5 movies are related to the movies the user has watched.

SYSTEM REQUIREMENTS/INSTALLATION AND EXECUTION INSTRUCTIONS:

SYSTEM REQUIREMENTS:

- Operating System: Linux x86 architecture
- Database – KDB/Q in memory database
- Python version – 2.7 4.
- Packages – Pandas, Numpy, Logger, Scikit-tensor

INSTALLATION STEPS:

- I have created single bash executable that will automatically install all the required packages and database. Please run requirements.sh executable in sudo mode.
- Command – sudo bash requirements.sh
- This will download KDB/Q community version and install it. Also, it will install all packages of python with pip.

This package uses distutils, which is the default way of installing python modules. The use of virtual environments is recommended.

- pip install scikit-tensor

To install in development mode

- git clone git@github.com:mnick/scikit-tensor.git
- pip install -e scikit-tensor/

EXECUTION STEPS:

- TASK 1:
  - 1a:
    - Command - ./print_actor_vector actorid model
    - Example - ./print_actor_vector 1234 TF
  - 1b:
    - Command - ./print_actor_vector actorid model
    - Example - ./print_actor_vector 1234 TF
  - 1c:
    - Command - ./print_actor_vector actorid model
    - Example - ./print_actor_vector 1234 TF
  - 1d:
    - Command - ./print_actor_vector actorid model
    - Example - ./print_actor_vector 1234 TF

- TASK 2:
  - 2a:
    - Command - ./print_actor_vector actorid model
    - Example - ./print_actor_vector 1234 TF
  - 2b:
    - Command - ./print_actor_vector actorid model
    - Example - ./print_actor_vector 1234 TF
  - 2c:
    - Command - ./print_actor_vector actorid model
    - Example - ./print_actor_vector 1234 TF
  - 2d:
    - Command - ./print_actor_vector actorid model
    - Example - ./print_actor_vector 1234 TF

- TASK 3:
  - 3a:
    - Command - ./print_actor_vector actorid model
    - Example - ./print_actor_vector 1234 TF
  - 3b:
    - Command - ./print_actor_vector actorid model

- Example - ./print_actor_vector 1234 TF

- TASK 4:
  - Command - ./print_actor_vector actorid model
  - Example - ./print_actor_vector 1234 TF

RELATED WORK:

- J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery: We used the algorithm given in the paper for calculating the page rank of all the actors based on the user's interest which is given in the input. The seed matrix is constructed based on the input. The paper uses Random Walks With Restarts algorithm for calculating the page rank. For calculating the top 10 similar actors for Task 3a and 3b we have used the same algorithm and have taken the top 10 page rank values of the actors.
- R. A. Harshman, Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis, UCLA Working Papers in Phonetics, 16 (1970): We referred this paper to know about the CP decomposition.
- J. D. Carroll and J. J. Chang, Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition, Psychometrika: This paper describes how to implement CP decomposition.
- [3]J.B. Kruskal, *Rank, decomposition, and uniqueness for 3-way and N-way arrays*, in *Multiway Data Analysis*, R. Coppi and S. Bolasco (eds.), Elsevier, Amsterdam, 1989, 7–18. : This paper gives an idea about how to represent tensors.

CONCLUSION:

Based on different decomposition methods like Single Value Decomposition, Principal Component Analysis, Latent Dirichlet Allocation we find top latent semantics, find and rank the similar actors by comparing actors and most related actors who have not acted in a movie. Implemented Singular Value Decomposition on the object – object similarity matrix to find the top latent semantics and partition the objects into non-overlapping groups based on their degree of membership. Implemented CP Decomposition on the tensor to find the top latent semantics underlying this tensor and partition these into non-overlapping groups based on their degree of membership. For an object-object similarity matrix with a set of seeds we implemented Personalized Page Ranking/ Random Walk with Restarts and found the 10 most related objects.

LIST OF PUBLICATIONS RELAVENT TO WORK:

- [1]R. A. Harshman, Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis, UCLA Working Papers in Phonetics, 16 (1970), pp. 1–84. Available at http://publish.uwo.ca/˜harshman/wpppfac0.pdf.
- [2]J. D. Carroll and J. J. Chang, Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition, Psychometrika, 35 (1970), pp. 283–319.
- [3]J.B. Kruskal, *Rank, decomposition, and uniqueness for 3-way and N-way arrays*, in *Multiway Data Analysis*, R. Coppi and S. Bolasco (eds.), Elsevier, Amsterdam, 1989, 7–18.
- [4]B.W. Bader and T.G. Kolda, A Preliminary Report on the Development of MATLAB Tensor Classes for Fast Algorithm Prototyping, Technical Report SAND2004–3487, Sandia National Laboratories, Livermore, CA, July 2004.

- [5]J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In KDD, pages 653658, 2004.

APPENDIX:

- Abhishek Thorat and Shiva Tejaswi Rampally implemented the Task 1.
- Shubam Gondane and Krishna Chaitanya Bejjipuram implemented Task 2.
- Sachin Sundar and Raviteja Upmaka implemented Task 3.
- Task 4 is implemented by combined ideas of all the team members.