RECOMMENDER SYSTEM (PHASE 2)
BY
GROUP 2

GROUP MEMBERS

Last Name	First Name
Shah	Mihika
Hashmi	Syed Usama
Nolastname	Vaishnavi Raj
Dikshit	Ishan
Kurunthiah	Satheesh
Sadaye	Raj

ABSTRACT

Multimedia objects cannot be stored with all available features due to dimensionality curse and cost factor. Initial step for any multimedia database design is feature selection or dimensionality reduction. Vector space model is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers, such as, for example, index terms. It is used in information filtering, information retrieval, indexing and relevancy rankings. This project aims in obtaining discriminating tag vectors for given inputs in a database so that it can be used for feature selection or dimensionality reduction. Once data has been reduced then other operations such as clustering and classification can be done.

Keywords: term frequency; inverse document frequency; PCA; SVD; LDA; Page Rank; Tensor;

INTRODUCTION

TERMINOLOGY

- Term – A term is any feature of an object in a vector space or dataset **[2]**
- Document – A document is an object that can have many features or terms **[2]**
- Term Frequency (TF) – It is the raw count of a term in a document, i.e. the number of times that term t occurs in document d **[4]**
- Inverse Document Frequency (TF-IDF) – It is a measure of how much information the word provides, that is, whether the term is common or rare across all documents D **[4]**
- Pandas – It is a powerful Python data analytics toolkit. It is used to read input files, process and store results as csv file
- Data Frame – It is a 2-dimensional labeled data structure with columns of potentially different types. It is like a spreadsheet or SQL table
- Tensor - It is an array of arbitrary dimension. **[2]**

GOAL DESCRIPTION

The goal of this project is to learn the steps involved in representing the higher dimension data into lower dimension by various dimensionality reduction techniques. The goal of each task is as follows

- The goal in task 1a is to find top 4 latent semantics using PCA and SVD on TF-IDF space of tags and LDA on TF space of tags
- The goal in task 1b is to find top 4 latent semantics using PCA and SVD on TF-IDF space of actors and LDA on TF space of actors
- The goal of task 1c is to find and rank the 10 most similar actors by comparing the actor's TF-IDF tag vector. The input for this task which is an actor id will be given as input by the user. Also, we are free to use either PCA or SVD or LDA for dimensionality reduction and retrieve top 5 latent semantics to map the actor points in the tag vector space.
- The goal of task 1d is to find and rank the 10 most related actor who have not acted in the given movie by comparing the movie's TF-IDF tag vectors. The input to this task which is a movie id will be given by user. Also, we are free to use either PCA or SVD or LDA for dimensionality reduction and retrieve top 5 latent semantics to map the movie point in the tag vector space.
- The goal of task 2a is to from actor-actor similarity matrix, perform SVD on it and then form non overlapping groups according to memberships to top 3 latent semantics in the actor space.
- The goal of task 2b is to from co-actor-co-actor matrix, perform SVD on it and then form non overlapping groups according to memberships to top 3 latent semantics in the actor space.

- The goal of task 2c is to create an actor-movie-year tensor and perform CP on it, with target rank equal to 5. The top 5 latent semantics and clusters for actor, movie and year need to be reported.
- The goal of task 3a is to from actor-actor similarity matrix, and use it to find 10 most related actors using Random Walk with Restarts.
- The goal of task 3b is to from actor-actor similarity matrix, and use it to find 10 most related actors using Random Walk with Restarts.
- The goal of task 4 is to recommend the user 5 movies to watch based on the information provided. This information could be list of movies which he had previously watched. Based on this information we need to recommend top 5 similar movies that relate to these parameters.

ASSUMPTIONS

- The files in Input directory is hard coded in the program. Any change in file name will not work
- The column names in each input files are also hard coded. Any change in column names will not work
- Task 1c
    - Run the program by typing python task1c.py <actor-id>
    - <actor-id> is any valid actor id in the database
- Task 1d
    - Run the program by typing python task1d.py <movie-id>
    - <movie-id> is any valid movie id in the database
- Task 4
    - Run the program by typing python task4.py <movie-id> <movie-id> .. <movie-id>
    - <movie-id> is any valid movie id in the database

DESCRIPTION OF THE IMPLEMENTATION

The implementation of all tasks is done using Python. As discussed in project phase 1, TF-IDF stands for "Term Frequency, Inverse Document Frequency". It is a way to score the importance of words (or "terms") in a document based on how frequently they appear across multiple documents. A common formula for IDF is as below:

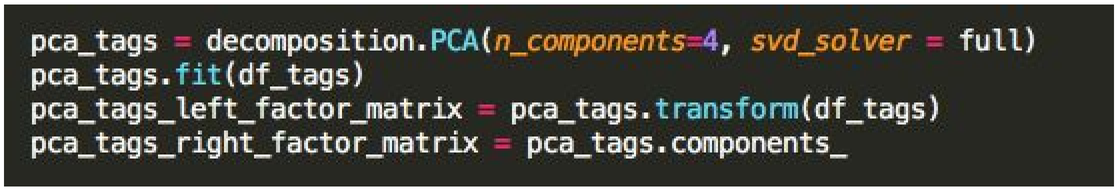
$$idf(\text{term}) = \ln\left(\frac{n_{\text{documents}}}{n_{\text{documents containing term}}}\right)$$

TASK 1A

In this task, we need an input matrix of genre vs TF-IDF of tag vectors on which dimensionality reduction is applied using PCA, SVD and LDA. After applying dimensionality reduction, top four new semantics should be reported.

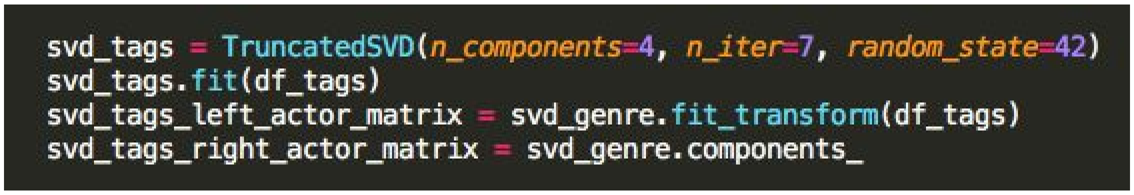
Get Input matrix
In case of PCA and SVD, for the input matrix, TF-IDF for each genre is calculated in tag space. All these vectors are appended together to form a N x M matrix, where N is number of genres and M is number of tags in the dataset. Understandably, a number of cells will be 0 as every genre might not have a relationship with a tag. In such cases we are assigning 0 value to such cells.
For LDA, input matrix is computed by taking into account TF values on the tag space for each genre.

PCA on input matrix
Python's library sklearn is used for decomposition computations on the input matrix. The function 'fit' is used to load the model with number of latent features (n_components) selected as 4 and SVD solver selected to be 'full'. Function transform applies the model to the N x M dataset where N = number of objects and M = number of old features. It returns an N x K matrix where K = number of latent semantics after dimensionality reduction, which is also the left factor matrix or U as taught in the class. 'Components_' returns the right factor matrix (K x M) or the rV matrix.



```
pca_tags = decomposition.PCA(n_components=4, svd_solver = full)
pca_tags.fit(df_tags)
pca_tags_left_factor_matrix = pca_tags.transform(df_tags)
pca_tags_right_factor_matrix = pca_tags.components_
```

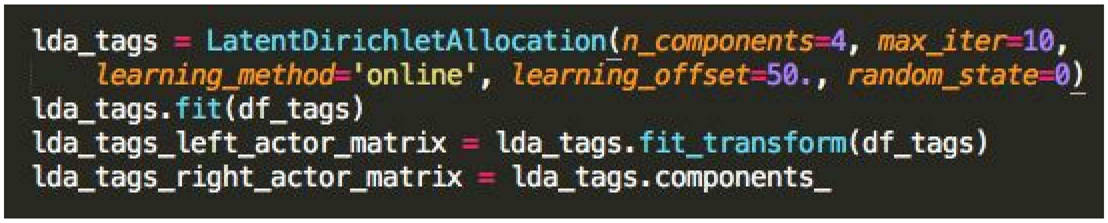
SVD on input matrix
The function 'fit' is used to load the SVD model with number of latent features (n_components) selected as 4 and n_iter, which denotes number of iterations is set to 7. Function transform applies the SVD model to the N x M original dataset where N = number of objects and M = number of old features. It returns an N x K matrix where K = number of latent semantics after dimensionality reduction, which is also the left factor matrix or U as taught in the class. 'Components_' returns the right factor matrix (K x M) or the V matrix.



```
svd_tags = TruncatedSVD(n_components=4, n_iter=7, random_state=42)
svd_tags.fit(df_tags)
svd_tags_left_actor_matrix = svd_genre.fit_transform(df_tags)
svd_tags_right_actor_matrix = svd_genre.components_
```

LDA on input matrix

For LDA, we are computing matrix on the term frequency space of vectors. Again, function 'fit' is used to load the LDA model with number of latent features (n_components) selected as 4 and number of iterations set to 10. Function transform applies the model to the N x M dataset where N = number of objects and M = number of old features. It returns an N x K matrix where K = number of latent semantics after probabilistic dimensionality reduction.

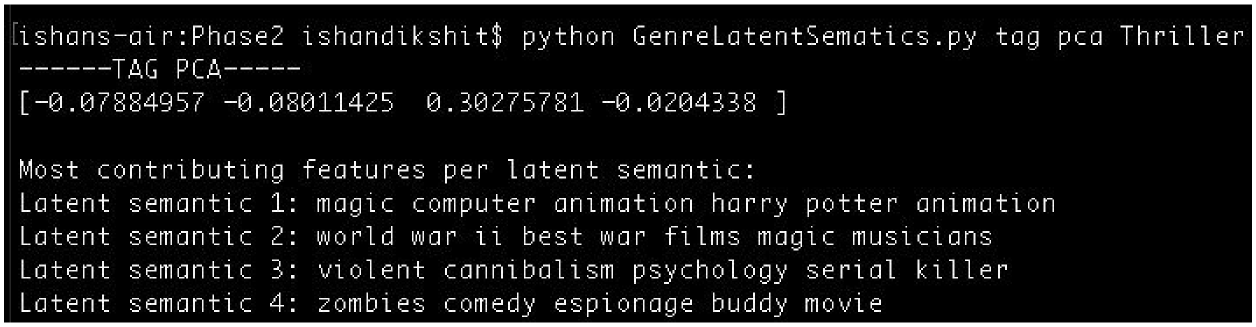


```
lda_tags = LatentDirichletAllocation(n_components=4, max_iter=10,
    learning_method='online', learning_offset=50., random_state=0)
lda_tags.fit(df_tags)
lda_tags_left_actor_matrix = lda_tags.fit_transform(df_tags)
lda_tags_right_actor_matrix = lda_tags.components_
```

Input command format: *"python GenreLatentSematics.py tag <model> <genre>"*

Sample Input command 1: python GenreLatentSematics.py tag pca Thriller

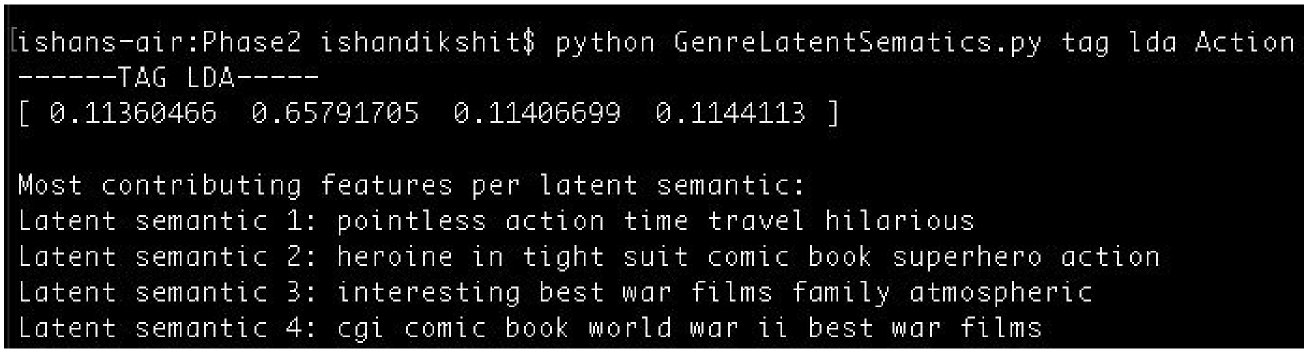
Output:



```
ishans-air:Phase2 ishandikshit$ python GenreLatentSematics.py tag pca Thriller
------TAG PCA-----
[-0.07884957 -0.08011425  0.30275781 -0.0204338 ]

Most contributing features per latent semantic:
Latent semantic 1: magic computer animation harry potter animation
Latent semantic 2: world war ii best war films magic musicians
Latent semantic 3: violent cannibalism psychology serial killer
Latent semantic 4: zombies comedy espionage buddy movie
```

Sample Input command 2: python GenreLatentSematics.py tag lda Action

Output:



```
ishans-air:Phase2 ishandikshit$ python GenreLatentSematics.py tag lda Action
------TAG LDA-----
[ 0.11360466  0.65791705  0.11406699  0.1144113 ]

Most contributing features per latent semantic:
Latent semantic 1: pointless action time travel hilarious
Latent semantic 2: heroine in tight suit comic book superhero action
Latent semantic 3: interesting best war films family atmospheric
Latent semantic 4: cgi comic book world war ii best war films
```

TASK 1B

In this task, we need an input matrix of genre vs TF-IDF of actor vectors on which dimensionality reduction is applied using PCA. After applying PCA, top four new semantics should be reported.

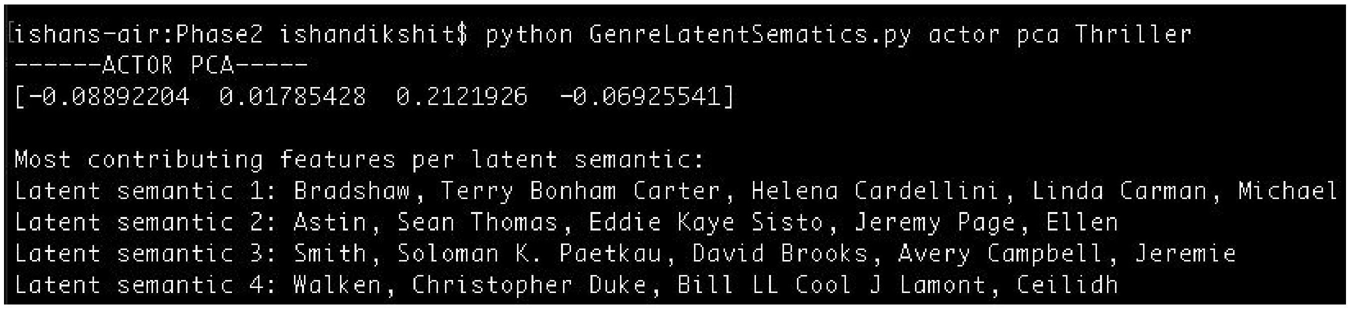
Get Input matrix
In case of PCA and SVD, for the input matrix, TF-IDF for each genre is calculated in actor space. All these vectors are appended together to form a N x M matrix, where N is number of genres and M is number of actors in the dataset. Understandably, a number of cells will be 0 as every genre might not have a relationship with an actor. In such cases we are assigning 0 value to such cells.
For LDA, input matrix is computed by taking into account TF values on the actor space for each genre.

Apply PCA, SVD and LDA on input matrix
Again, similar to task 1a, python's library sklearn is used for decomposition computations on the input matrix. The difference being, in this task the input matrix consists of genre vectors in actor space instead of tag space as in program 1a.
Python's library sklearn is used for decomposition computations on the input matrix.

Input command format: *"python GenreLatentSematics.py actor <model> <genre>"*

Sample Input Command 1: python GenreLatentSematics.py actor pca Thriller

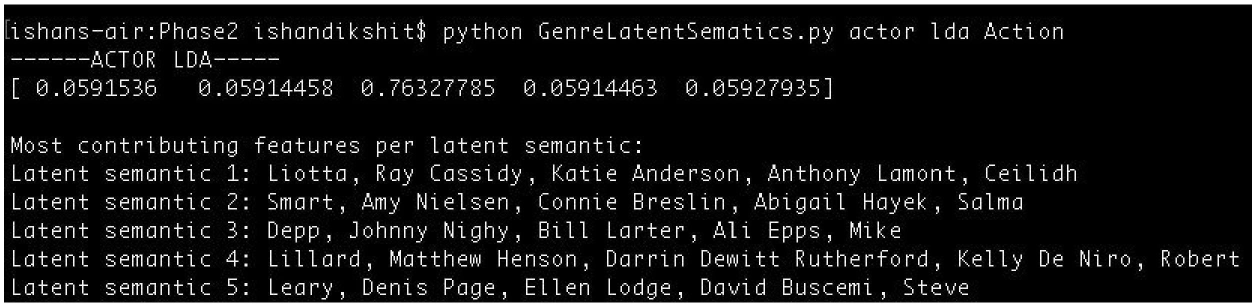
Output:



```
ishans-air:Phase2 ishandikshit$ python GenreLatentSematics.py actor pca Thriller
------ACTOR PCA-----
[-0.08892204  0.01785428  0.2121926  -0.06925541]

Most contributing features per latent semantic:
Latent semantic 1: Bradshaw, Terry Bonham Carter, Helena Cardellini, Linda Carman, Michael
Latent semantic 2: Astin, Sean Thomas, Eddie Kaye Sisto, Jeremy Page, Ellen
Latent semantic 3: Smith, Soloman K. Paetkau, David Brooks, Avery Campbell, Jeremie
Latent semantic 4: Walken, Christopher Duke, Bill LL Cool J Lamont, Ceilidh
```

Sample Input Command 2: python GenreLatentSematics.py actor lda Action

Output:



```
ishans-air:Phase2 ishandikshit$ python GenreLatentSematics.py actor lda Action
------ACTOR LDA-----
[ 0.0591536   0.05914458  0.76327785  0.05914463  0.05927935]

Most contributing features per latent semantic:
Latent semantic 1: Liotta, Ray Cassidy, Katie Anderson, Anthony Lamont, Ceilidh
Latent semantic 2: Smart, Amy Nielsen, Connie Breslin, Abigail Hayek, Salma
Latent semantic 3: Depp, Johnny Nighy, Bill Larter, Ali Epps, Mike
Latent semantic 4: Lillard, Matthew Henson, Darrin Dewitt Rutherford, Kelly De Niro, Robert
Latent semantic 5: Leary, Denis Page, Ellen Lodge, David Buscemi, Steve
```

TASK 1C

Get Input matrix

For any given new data set Actor's TF-IDF tag vectors are calculated first using methods discussed in Phase 1 project. This model was stored in a csv file which has actor id, tags associated with this actor and its corresponding TF-IDF weights. This is used to calculate actor's tag vector. The cell where an actor has no tag will be assigned as zero. Thus, our input matrix will have all actors in rows and all tags in column and its corresponding TF-IDF values. Then we apply PCA method to reduce the dimensionality of the data.

Apply PCA on this input matrix

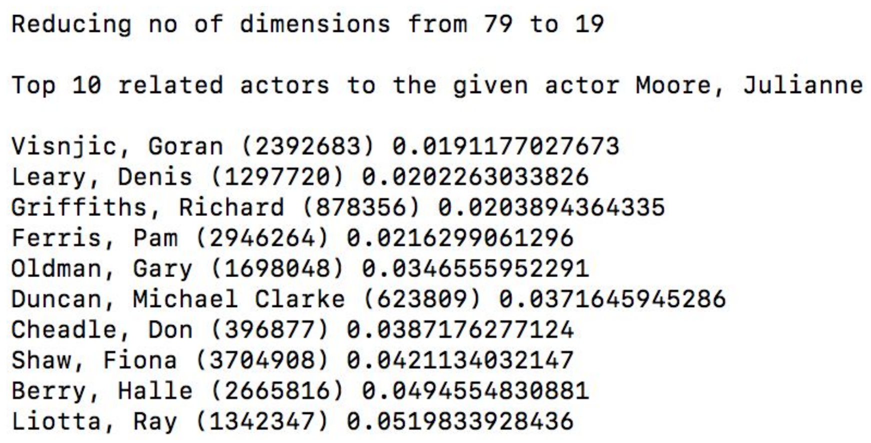
- Find the covariance matrix from the input matrix
- Get Eigen values and Eigen vectors as pairs from this covariance matrix.
- These pairs are sorted based on Eigen values in descending order. We cannot assume always the Eigen decomposition will give sorted Eigen values and Eigen vector pair. If the TF-IDF values are not normalized then we get random or unsorted Eigen value vector pair
- We decide number of dimensions to keep based on the values obtained from a factor called "explained variance". This is calculated as follows
  - Total_sum = sum(eig_vals)
  - Exp_var = [(index / tot) * 100 for index in eig_vals]
  - cum_var_exp = np.cumsum(var_exp)
- The explained variance tells us how much information (variance) can be attributed to each of the principal components. In order to hold 100% of variance we keep the threshold as 100 for this cum_var_exp and keep adding top dimensions until this number is reached
- Here, we are reducing the m-dimensional feature space to a k-dimensional feature subspace (k<<m), by choosing the "top k" eigenvectors with the highest eigenvalues

Find Similar actor

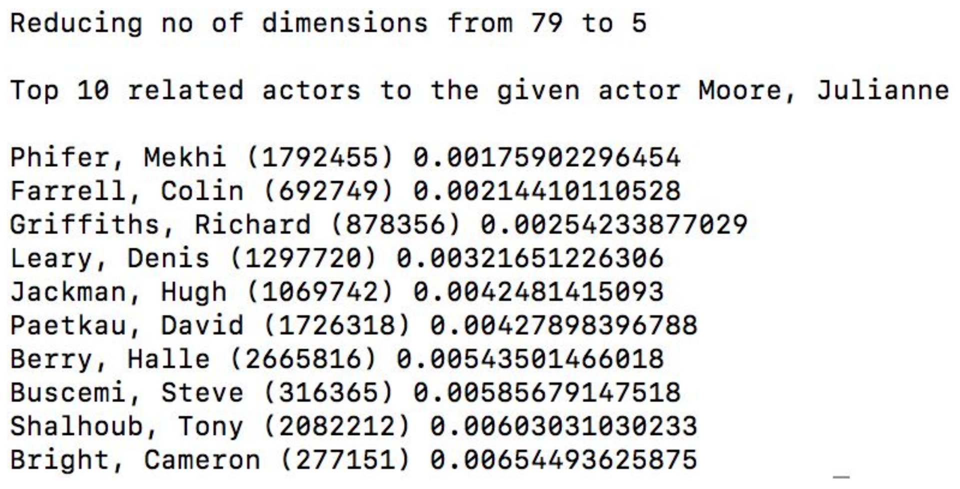
Our points in new vector space represents the actors. If two points are closer we assume they are related in tag space and are considered similar. We use scipy package to compute Euclidean distance between the given actor id and all actor id in the reduced dimension space. We sort the distance array in ascending order and get first 10 values.

Output

- The output represents related actor with their id and the distance between given actor. Here we try to maintain 90% of original variance in the reduced new dimension. We get 90% of variance when we add 19 latent semantics.


```
Reducing no of dimensions from 79 to 19

Top 10 related actors to the given actor Moore, Julianne

Visnjic, Goran (2392683) 0.0191177027673
Leary, Denis (1297720) 0.0202263033826
Griffiths, Richard (878356) 0.0203894364335
Ferris, Pam (2946264) 0.0216299061296
Oldman, Gary (1698048) 0.0346555952291
Duncan, Michael Clarke (623809) 0.0371645945286
Cheadle, Don (396877) 0.0387176277124
Shaw, Fiona (3704908) 0.0421134032147
Berry, Halle (2665816) 0.0494554830881
Liotta, Ray (1342347) 0.0519833928436
```

· Here we have further reduced the dimensions to only 5. This will not capture 90% variance but good enough for comparing similarities between objects.


```
Reducing no of dimensions from 79 to 5

Top 10 related actors to the given actor Moore, Julianne

Phifer, Mekhi (1792455) 0.00175902296454
Farrell, Colin (692749) 0.00214410110528
Griffiths, Richard (878356) 0.00254233877029
Leary, Denis (1297720) 0.00321651226306
Jackman, Hugh (1069742) 0.0042481415093
Paetkau, David (1726318) 0.00427898396788
Berry, Halle (2665816) 0.00543501466018
Buscemi, Steve (316365) 0.00585679147518
Shalhoub, Tony (2082212) 0.00603031030233
Bright, Cameron (277151) 0.00654493625875
```

TASK 1D

Get Input matrix

For any given new data set Movie's TF-IDF tag vectors are calculated first using methods discussed in Phase 1 project. This model was stored in a csv file which has movie id, tags associated with this movie and its corresponding TF-IDF weights. This is used to calculate movie's tag vector. The cell where a movie has no tag will be assigned as zero. Thus, our input matrix will have all movies in rows and all tags in column and its corresponding TF-IDF values. Then we apply PCA method to reduce the dimensionality of the data.

Apply PCA on this input matrix

- Find the covariance matrix from the input matrix
- Get Eigen values and Eigen vectors as pairs from this covariance matrix.
- These pairs are sorted based on Eigen values in descending order. We cannot assume always the Eigen decomposition will give sorted Eigen values and Eigen vector pair. If

the TF-IDF values are not normalized then we get random or unsorted Eigen value vector pair

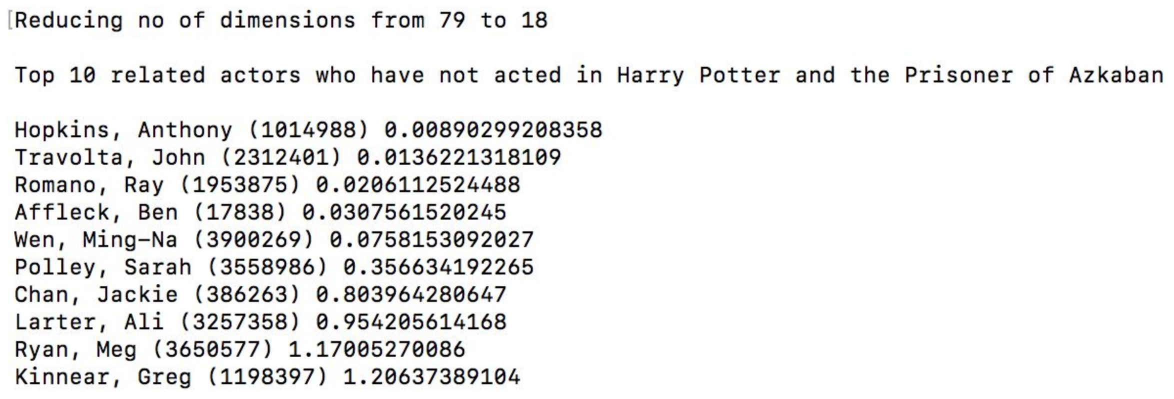
- We decide number of dimensions to keep based on the values obtained from a factor called “explained variance”. This is calculated as follows
  - Total_sum = sum(eig_vals)
  - Exp_var = [(index / tot) * 100 for index in eig_vals]
  - cum_var_exp = np.cumsum(var_exp)
- The explained variance tells us how much information (variance) can be attributed to each of the principal components. In order to hold 100% of variance we keep the threshold as 100 for this cum_var_exp and keep adding top dimensions until this number is reached
- Here, we are reducing the m-dimensional feature space to a k-dimensional feature subspace (k<<m), by choosing the "top k" eigenvectors with the highest eigenvalues

Find Similar actor

- Our points in new vector space represents the movies. If two points are closer we assume they are related in tag space and are considered similar. We use scipy package to compute Euclidean distance between the given movie id and all movie id in the reduced dimension space. We sort the distance array in ascending order and get similar actors who have not acted in the given movie as follows
- From the similar movie get the actors who have acted in this movie
- Select the actor with lower actor rank and check if he has not acted in the given movie and if so select this candidate as related actor.
- Repeat this process until you get 10 related actors

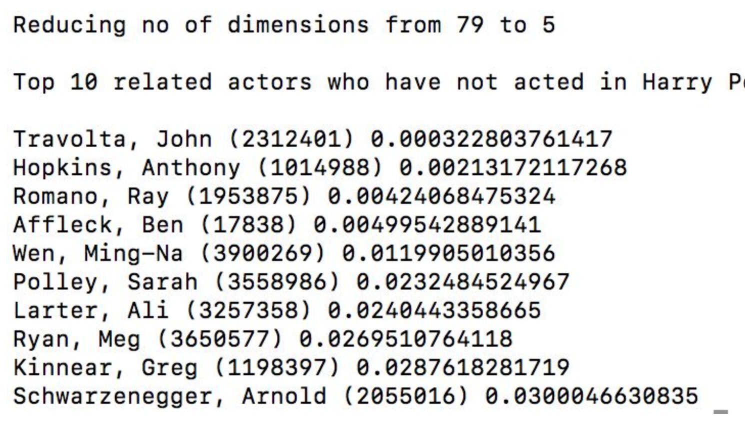
Output

· The output represents related actor with their id and the distance between given movie. Here we try to maintain 90% of original variance in the reduced new dimension. We get 90% of variance when we add 18 latent semantics.



```
Reducing no of dimensions from 79 to 18

Top 10 related actors who have not acted in Harry Potter and the Prisoner of Azkaban

Hopkins, Anthony (1014988) 0.00890299208358
Travolta, John (2312401) 0.0136221318109
Romano, Ray (1953875) 0.0206112524488
Affleck, Ben (17838) 0.0307561520245
Wen, Ming-Na (3900269) 0.0758153092027
Polley, Sarah (3558986) 0.356634192265
Chan, Jackie (386263) 0.803964280647
Larter, Ali (3257358) 0.954205614168
Ryan, Meg (3650577) 1.17005270086
Kinnear, Greg (1198397) 1.20637389104
```

· Here we have further reduced the dimensions to only 5. This will not capture 90% variance but good enough for comparing similarities between objects.



```
Reducing no of dimensions from 79 to 5

Top 10 related actors who have not acted in Harry Potter and the Prisoner of Azkaban

Travolta, John (2312401) 0.000322803761417
Hopkins, Anthony (1014988) 0.00213172117268
Romano, Ray (1953875) 0.00424068475324
Affleck, Ben (17838) 0.00499542889141
Wen, Ming-Na (3900269) 0.0119905010356
Polley, Sarah (3558986) 0.0232484524967
Larter, Ali (3257358) 0.0240443358665
Ryan, Meg (3650577) 0.0269510764118
Kinnear, Greg (1198397) 0.0287618281719
Schwarzenegger, Arnold (2055016) 0.0300046630835
```

TASK 2

TASK 2A

Filename: task2a.py
Goals:
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
4. partitions the actors into 3 non-overlapping groups based on their degrees of memberships to these 3 semantics

Pretask:
Imports:

```
# this function is from phase 1 and is used to get the tfidf weights for tags found for each actor
import print_actor_vector as pav

# sklearn decomposition is used for performing SVD
import sklearn.decomposition as sd

# sklearn clustering is used for kmeans clustering
import sklearn.cluster as sc

# numpy is used for finding covariance for mahalanobis distance
import numpy as np

# dataframes is a pre-task for the first phase that gives access to the datased
import dataframes as df

# these are distance metrics that are being used
# user can choose from each of these metrics to get similarity matrix
```

```
from sklearn.metrics.pairwise import manhattan_distances
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics.pairwise import cosine_distances
from sklearn.metrics.pairwise import distance
```

Generation of the tag vectors for each actor is a prerequisite for the first task. The function used is from phase 1 in the file is given below:

```
# to generate the actor vs tag matrix
# where each actor is one row and is represented in terms of tfidf of tags
# that has appeared for that particular actor
def __get_actor_tag_matrix__(consider_zero_tag_vectors=True):

  # getting tfidf tags for all vectors (this function was written in phase 1)
  actor_data = pav.get_tf_idf_for_all_actors()
  tag_vector = {}
  actor_vector = {}
  actor_list = []
  actor_count = 0
  tag_count = 0

  # generating a list of all actors present in the dataset to associate with the matrix
  # the index of the actorid is the same as the index/row number in the actor_tag_matrix
  for actor_datum in actor_data:
      if not actor_vector.has_key(actor_datum['actorid']):
          actor_vector[actor_datum['actorid']] = actor_count
          actor_list.append(actor_datum['actorid'])
          actor_count += 1
      if not tag_vector.has_key(actor_datum['tag']):
          tag_vector[actor_datum['tag']] = tag_count
          tag_count += 1

  # check if the user wants to ignore zero tag actors
  if consider_zero_tag_vectors:

      # if user wants to keep the actors with no tags
      # adds the user tags to the list
      for actor in df.movie_actors_data['actorid'].unique():
          if not actor_vector.has_key(actor):
              actor_vector[actor] = actor_count
              actor_list.append(actor)
              actor_count += 1

  actor_tag_matrix = [[[] for _ in range(tag_count)] for _ in range(len(actor_list))]

  # generating the actor_tag matrix from tag vectors received above
  for actor_datum in actor_data:
      actor_index = actor_vector.get(int(actor_datum['actorid']))
      tag_index = tag_vector.get(str(actor_datum['tag']))
      actor_tag_matrix[actor_index][tag_index] = float(actor_datum['tfidfweight'])
  # filling 0 for all tag values that were not present in the actor for the given row
  for row_index in range(len(actor_tag_matrix)):
      for column_index in range(len(actor_tag_matrix[row_index])):
          if not actor_tag_matrix[row_index][column_index]:
              actor_tag_matrix[row_index][column_index] = 0
  return actor_list, actor_tag_matrix
```

This part does not contain any mathematical assumptions or formulation. Here, tags with their weights for each actor are converted into an actor vs tag matrix where actors are rows and tags are columns.

Goal 1: creates an actor-actor similarity matrix (using tag vectors)

Code:

```
# generates the actor-actor similarity matrix based on the distances between their tags
# this is capable of generating similarity matrix using a total of
# 4 different distance/similarity measures
# all distance measures consider either inverse or 1/measure to get similarity
# metric instead of distance measure
def get_actor_actor_similarity_matrix(similarity_measure=3, consider_zero_tag_vectors=True):
  actor_list, actor_tag_matrix = __get_actor_tag_matrix__(consider_zero_tag_vectors)
  actor_actor_similarity_matrix = [[[] for _ in range(len(actor_tag_matrix))]
                                  for _ in range(len(actor_tag_matrix))]
  if similarity_measure == 0:

      # inverse of manhattan distance matrix generation
      actor_actor_similarity_matrix = \
          __get_similarity_from_distance_matrix__(manhattan_distances(actor_tag_matrix))

  elif similarity_measure == 1:

      # cosine similarity matrix generation
      actor_actor_similarity_matrix = cosine_similarity(actor_tag_matrix)

  elif similarity_measure == 2:

      # inverse of cosine distance matrix generation
      actor_actor_similarity_matrix = \
          __get_similarity_from_distance_matrix__(cosine_distances(actor_tag_matrix))

  elif similarity_measure == 3:

      # mahalanobis distance calculation but the distance value is inverted i.e 1/value
      # once the distance is calculated and put in the similarity metric
      cov_matrix = np.cov(np.transpose(actor_tag_matrix))
      for actor_index in range(len(actor_tag_matrix)):
          for actor_index_2 in range(len(actor_tag_matrix)):
              dist = distance.mahalanobis(u=actor_tag_matrix[actor_index],
                            v=actor_tag_matrix[actor_index_2],
                            VI=cov_matrix)
              if dist == 0:
                  actor_actor_similarity_matrix[actor_index][actor_index_2] = 1
              else:
                  actor_actor_similarity_matrix[actor_index][actor_index_2] = 1/(1 + dist)

  else:

      # Euclidean distance measure
      # uses the distance function given above that returns reciprocal values of distance
      # making it mimic a similarity measure
      for actor_index in range(len(actor_tag_matrix)):
          for actor_index_2 in range(len(actor_tag_matrix)):
              dist = __get_actor_actor_euclidean_similarity__(
                  actor_tag_matrix[actor_index],
                  actor_tag_matrix[actor_index_2])
              actor_actor_similarity_matrix[actor_index][actor_index_2] = dist

  return actor_list, actor_actor_similarity_matrix
```

Explanation:
Actor actor similarity matrix is calculated using 5 different distance/similarity metrics namely Euclidean distance, Mahalanobis distance, Cosine similarity, Cosine distance and Manhattan distance.

As you can see that these distance metrics do not represent similarity, I have used a way of converting distance to similarity as given in the stackexchange answer here: https://stats.stackexchange.com/questions/158279/how-i-can-convert-distance-euclidean-to-similarity-score**.**

Formula used for conversion of distance to similarity:
If $d(p1,p2)$ represents the euclidean distance from point $p1$ to point $p2$

$$s(p1,p2) = 1/1+d(p1,p2)$$

Goal 2: performs SVD on this actor-actor similarity matrix
Code:

```
# performs SVD to get actor against latent semantic matrix
# using the actor-actor similarity metric generated above
def __get_actor_latent_matrix__(similarity_measure, consider_zero_tag_vectors=True):
  actor_list, actor_tag_matrix = \
     get_actor_actor_similarity_matrix(similarity_measure, consider_zero_tag_vectors)
  svd = sd.TruncatedSVD(n_components=3)
  svd.fit(actor_tag_matrix)
  semantics = []
```

```
    return semantics, actor_list, svd.transform(actor_tag_matrix)
```

Explanation:

We have used the following line

svd = sd.TruncatedSVD(n_components=3)

to get the function from scikit.decomposition package that performs SVD and generates output with 3 components. The code given below

svd.fit(actor_tag_matrix)

is used to generate the model on the actor tag matrix generated before. The code given below

svd.transform(actor_tag_matrix)

is used to output the transformed matrix that has 3 columns and rows same as the number of actors.

Goal 3: reports the top-3 latent semantics, in the actor space, underlying this actor-actor similarity matrix

Code:

```
    # generating the semantics in terms of actors and their importance
    # present in each of the calculated semantics
    for i in range(len(svd.components_)):
        semantics.append({})
        for actor in range(len(svd.components_[i])):
            semantics[i][actor_list[actor]] = svd.components_[i][actor]
```

Explanation:

The line below svd.components_ provides a list of latent semantics where each semantic is a list of the actual semantics, actors in our case, and a value that is a measure of strength of membership of that actual semantic in the latent semantic. In simpler words, each value defines how important an actor in defining the latent semantic.

Goal 4: partitions the actors into 3 non-overlapping groups based on their degrees of memberships to these 3 semantics

Code:

```
# k means clustering is done to put the actors into 3 clusters based on
# their distances with respect to the latent semantics generated above.
def get_clusters(similarity_measure, consider_zero_tag_vectors=True):
  semantics, actor_list, latent_matrix = __get_actor_latent_matrix__(similarity_measure, consider_zero_tag_vectors)

  # scikit clustering is used for kmean clustering
  kmeans = sc.KMeans(n_clusters=3)
  kmeans.fit(latent_matrix)
  kmeans.transform(latent_matrix)
  print "Total number of Actors: ", len(latent_matrix)
```

```
# separating clusters into 2-D arrays for easy representation later
cluster1 = []
cluster1_actors = []
cluster2 = []
cluster2_actors = []
cluster3 = []
cluster3_actors = []
for label in range(len(kmeans.labels_)):
    if kmeans.labels_[label] == 0:
        cluster1.append(latent_matrix[label])
        cluster1_actors.append(actor_list[label])
    if kmeans.labels_[label] == 1:
        cluster2.append(latent_matrix[label])
        cluster2_actors.append(actor_list[label])
    if kmeans.labels_[label] == 2:
        cluster3.append(latent_matrix[label])
        cluster3_actors.append(actor_list[label])

# clusters is a list of three clusters
# each of these three clusters are a list of actorids
clusters = [cluster1, cluster2, cluster3]

# cluster_actors is a list of three clusters
# each of these three clusters are a list of actor names
cluster_actors = [cluster1_actors, cluster2_actors, cluster3_actors]
return semantics, cluster_actors, clusters
```

Explanation:

The code given below

```
kmeans = sc.KMeans(n_clusters=3)
kmeans.fit(latent_matrix)
kmeans.transform(latent_matrix)
```

uses scikit's clustering package to perform kmean clustering. We used n_clusters=3 so that it outputs 3 clusters.

```
kmeans.labels_
```

is used to get a matrix that determines what cluster each actor was placed in.

Output:

The output of this is present in the Outputs folder and since it is really big, it can not be made a part of this report. Some insights are that cosine distance worked really well to get the similarity matrix. Mahalanobis distance also performed well as compared to euclidean and manhattan distance. Some sensible clusters could be seen when using Mahalanobis distance and cosine distance metrics and when the actors with 0 tfidf weights for all tags were ignored. Otherwise all these actors just became part of one of the clusters who were closest to the origin point (zero vector) of the vector space.

TASK 2B

Goals:

- Form a co-actor – co-actor matrix.
- Perform Singular Value Decomposition on the co-actor – co-actor matrix
- Report the top 3 latent semantics in the reduced space of actors
- Partition the actors into 3 non-overlapping groups based on their membership to the latent semantics

For forming the co-actor-co-actor matrix:

1. A dictionary is formed having key values as actor id and list of movies played by that actor as values.
2. For every 2 actors in the database an intersection between their key-value pairs is computed.
3. The count of this key value pair is stored in the corresponding row of the co-actor-co-actor matrix

The function used for computing this matrix is symmetric and hence the matrix will also be symmetric, i.e.
co_co(a1,a2)=co_co(a2,a1)

Singular valued Decomposition(SVD) reduces the space by preserving data. Eigen decomposition reduces the dimensionality of data without scaling
Let D= Data matrix
Then SVD will decompose it into 3 matrices:

D= U * S * VT

U= Left factor matrix that represents objects in the reduced space of latent semantics
S= A diagonal matrix having squares of eigenvalues corresponding to latent semantics arranged in descending order
VT= Right Factor matrix that represents the latent semantics in terms of the features.

Since the matrix is symmetric,
The columns in U and the rows in V will be the same.
We use the eigenvalues in S matrix for choosing the top latent semantics we want to preserve.

We implement the SVD decomposition using the scikit-learn svd package.
svd = TruncatedSVD(n_components=3,algorithm='arpack')

This will form a model for Singular Valued Decomposition. Since n_components=3 we indicate that we want to retain the top 3 latent semantics of data.

The we use,
svd.fit(co_co)

This function fits the Latent Semantic Indexing model to our co-actor-co-actor matrix.

Further, we perform
dec=svd.fit_transform(co_co)

This function will perform dimensionality reduction on the matrix and return the data matrix that has 3 components.
I.e. if input= [n_objects, n_features]
output=[n_objects,n_components]

Since we have included the number of components as the input to the model,the function will give us back the mapping of actors onto the reduced space represented by top 3 latent semantics

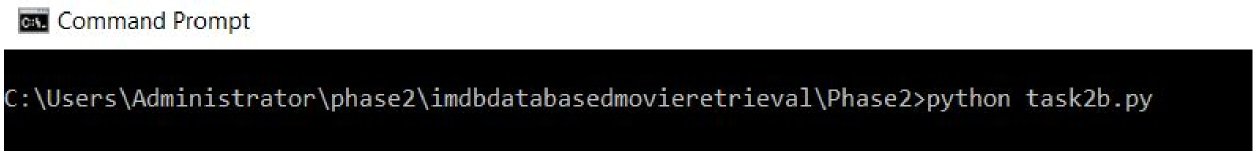
Further we need to form non- overlapping groups according to membership of objects to these 3 latent semantics.
We will use k-means clustering for forming these non- overlapping groups.

kmeans = KMeans(n_clusters=3, random_state=0).fit(dec)

We pass number of clusters as 3 and seed as 0. This will group data based on membership to these latent semantics.

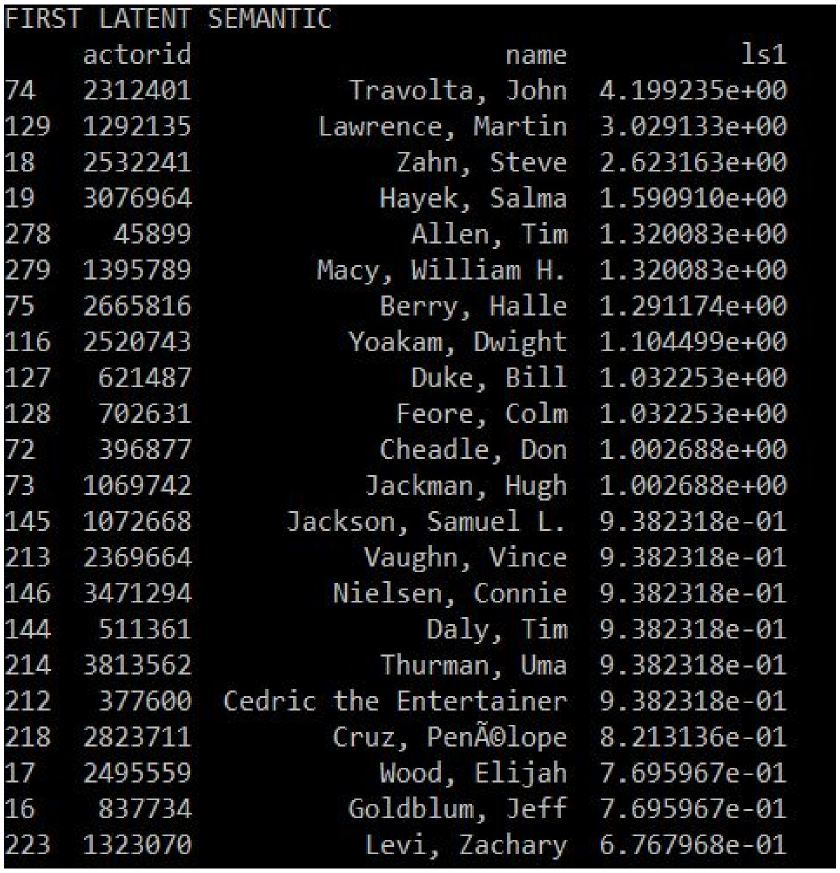
Finally we will display the top 3 latent semantics and the 3 grouping we have formed

Input command :

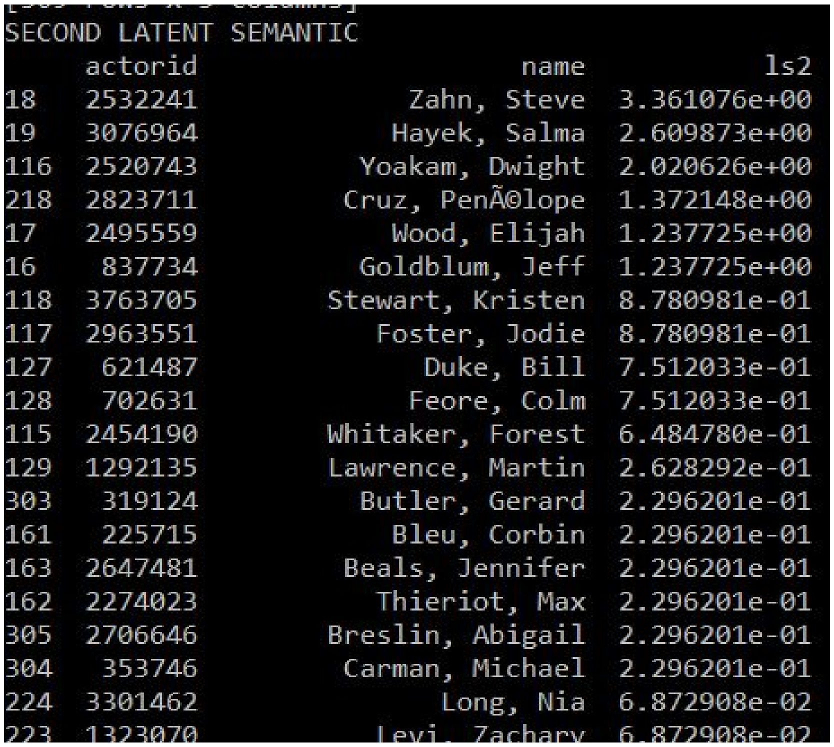


```
C:\Users\Administrator\phase2\imdbdatabasedmovieretrieval\Phase2>python task2b.py
```

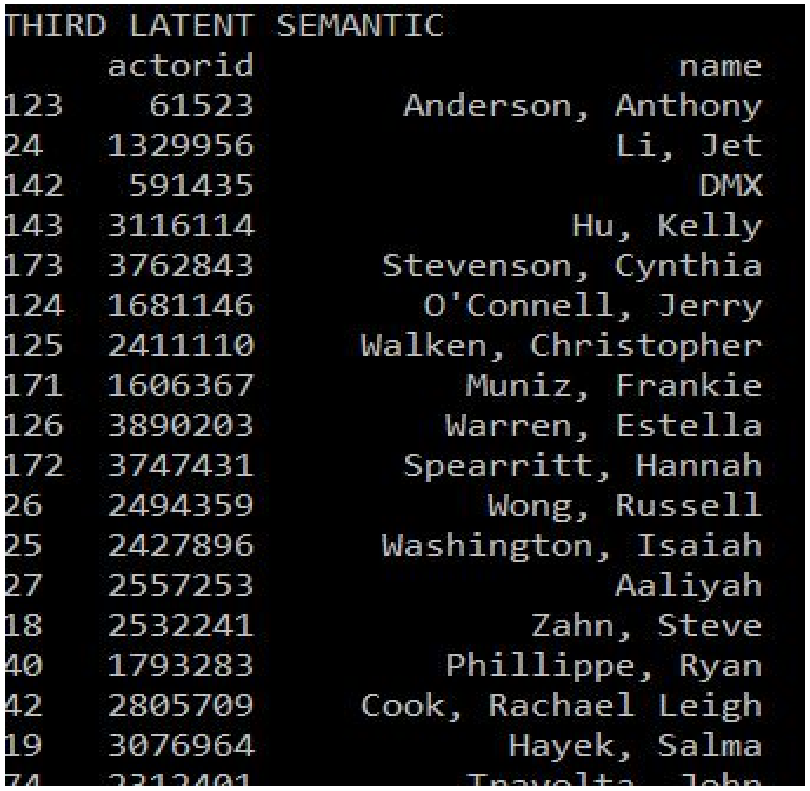
Sample Output:



```
FIRST LATENT SEMANTIC
     actorid                   name           ls1
74   2312401         Travolta, John  4.199235e+00
129  1292135       Lawrence, Martin  3.029133e+00
18   2532241            Zahn, Steve  2.623163e+00
19   3076964           Hayek, Salma  1.590910e+00
278    45899              Allen, Tim  1.320083e+00
279  1395789       Macy, William H.  1.320083e+00
75   2665816           Berry, Halle  1.291174e+00
116  2520743         Yoakam, Dwight  1.104499e+00
127   621487             Duke, Bill  1.032253e+00
128   702631            Feore, Colm  1.032253e+00
72    396877           Cheadle, Don  1.002688e+00
73   1069742          Jackman, Hugh  1.002688e+00
145  1072668     Jackson, Samuel L.  9.382318e-01
213  2369664          Vaughn, Vince  9.382318e-01
146  3471294        Nielsen, Connie  9.382318e-01
144   511361              Daly, Tim  9.382318e-01
214  3813562           Thurman, Uma  9.382318e-01
212   377600  Cedric the Entertainer  9.382318e-01
218  2823711        Cruz, PenÃ©lope  8.213136e-01
17   2495559           Wood, Elijah  7.695967e-01
16    837734         Goldblum, Jeff  7.695967e-01
223  1323070          Levi, Zachary  6.767968e-01
```

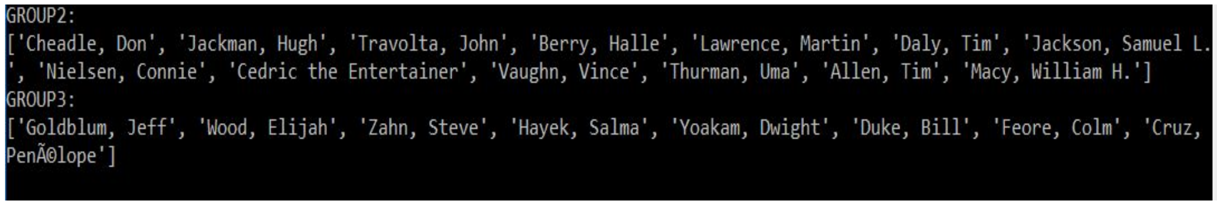


```
SECOND LATENT SEMANTIC
     actorid                   name           ls2
18   2532241            Zahn, Steve  3.361076e+00
19   3076964           Hayek, Salma  2.609873e+00
116  2520743         Yoakam, Dwight  2.020626e+00
218  2823711        Cruz, PenÃ©lope  1.372148e+00
17   2495559           Wood, Elijah  1.237725e+00
16    837734         Goldblum, Jeff  1.237725e+00
118  3763705       Stewart, Kristen  8.780981e-01
117  2963551          Foster, Jodie  8.780981e-01
127   621487             Duke, Bill  7.512033e-01
128   702631            Feore, Colm  7.512033e-01
115  2454190       Whitaker, Forest  6.484780e-01
129  1292135       Lawrence, Martin  2.628292e-01
303   319124         Butler, Gerard  2.296201e-01
161   225715           Bleu, Corbin  2.296201e-01
163  2647481       Beals, Jennifer  2.296201e-01
162  2274023            Thieriot, Max  2.296201e-01
305  2706646        Breslin, Abigail  2.296201e-01
304   353746        Carman, Michael  2.296201e-01
224  3301462              Long, Nia  6.872908e-02
223  1323070          Levi, Zachary  6.872908e-02
```



```
THIRD LATENT SEMANTIC
     actorid                    name
123    61523       Anderson, Anthony
24   1329956                Li, Jet
142   591435                    DMX
143  3116114              Hu, Kelly
173  3762843     Stevenson, Cynthia
124  1681146       O'Connell, Jerry
125  2411110    Walken, Christopher
171  1606367         Muniz, Frankie
126  3890203         Warren, Estella
172  3747431      Spearritt, Hannah
26   2494359          Wong, Russell
25   2427896     Washington, Isaiah
27   2557253                Aaliyah
18   2532241            Zahn, Steve
40   1793283        Phillippe, Ryan
42   2805709     Cook, Rachael Leigh
19   3076964           Hayek, Salma
74   2312401         Travolta, John
```

Output for clusters:

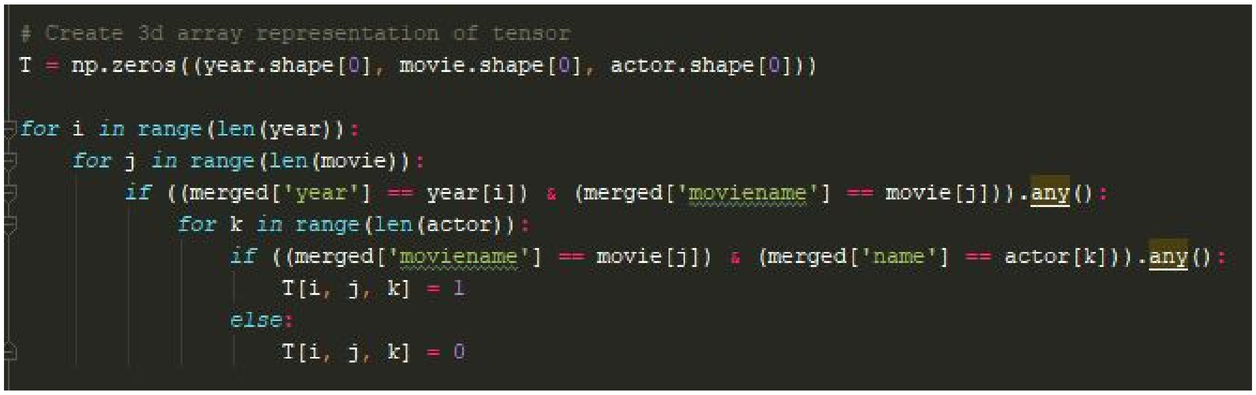


```
GROUP2:
['Cheadle, Don', 'Jackman, Hugh', 'Travolta, John', 'Berry, Halle', 'Lawrence, Martin', 'Daly, Tim', 'Jackson, Samuel L.
', 'Nielsen, Connie', 'Cedric the Entertainer', 'Vaughn, Vince', 'Thurman, Uma', 'Allen, Tim', 'Macy, William H.']
GROUP3:
['Goldblum, Jeff', 'Wood, Elijah', 'Zahn, Steve', 'Hayek, Salma', 'Yoakam, Dwight', 'Duke, Bill', 'Feore, Colm', 'Cruz,
PenÃ©lope']
```

TASK 2C

Create tensor

To create the actor-movie-year tensor, data from the csv files are used. Pandas is used to merge the files, and the dataframe is converted into a 3D matrix, where actor name, movie name and year are the features. This matrix is converted into 1D numpy arrays and duplicates are removed. The tensor element value is stored as 1 if the actor-movie-year triple exists, otherwise 0 is stored. The code below implements this:

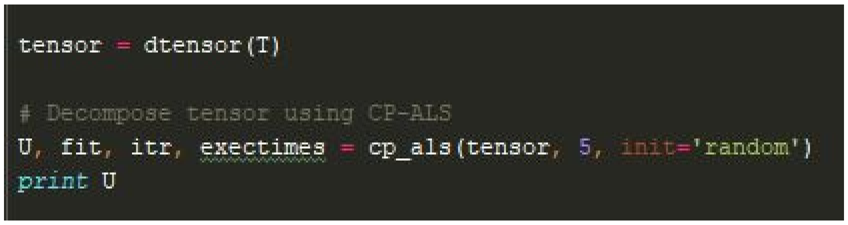


```
# Create 3d array representation of tensor
T = np.zeros((year.shape[0], movie.shape[0], actor.shape[0]))

for i in range(len(year)):
    for j in range(len(movie)):
        if ((merged['year'] == year[i]) & (merged['moviename'] == movie[j])).any():
            for k in range(len(actor)):
                if ((merged['moviename'] == movie[j]) & (merged['name'] == actor[k])).any():
                    T[i, j, k] = 1
                else:
                    T[i, j, k] = 0
```

This numpy array is converted into a dense tensor using the scikit-tensor (sktensor) library.

Performing CP

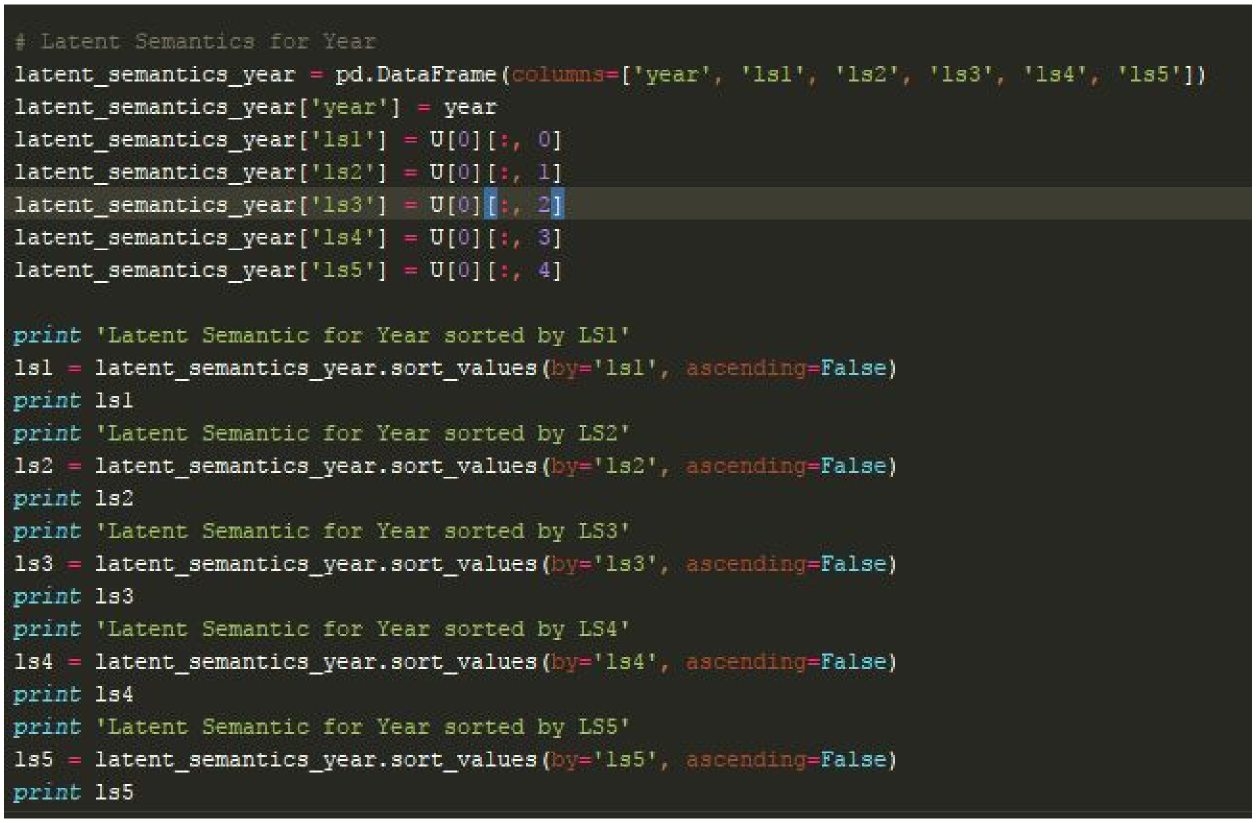
Once the tensor is formed, CP is performed on it, using the sktensor library function cp_als. The input parameters given are the tensor, its target rank (equal to 5), and init=random indicates that the factor matrices are initialized randomly. This function returns the 3 factor matrices, the fit of the transformation, number of iterations taken to reach this output, and the time taken during each iteration.



```
tensor = dtensor(T)

# Decompose tensor using CP-ALS
U, fit, itr, exectimes = cp_als(tensor, 5, init='random')
print U
```

Displaying the latent semantics

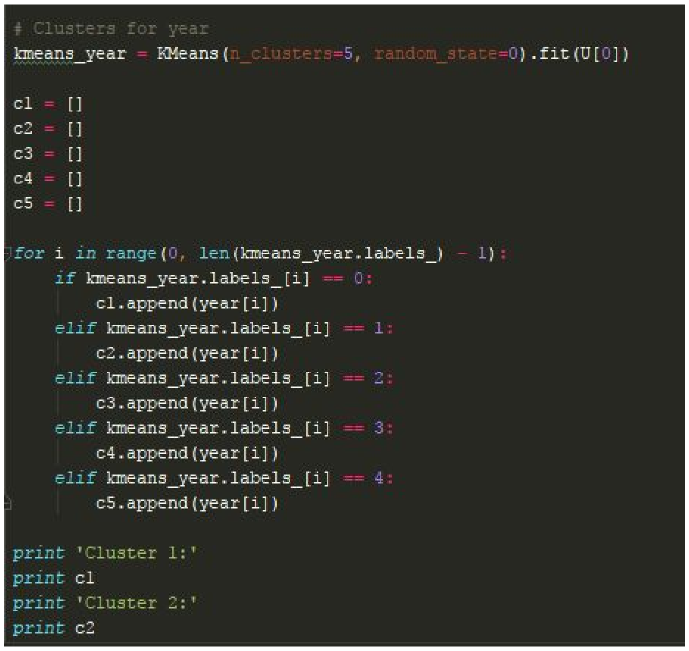
The top-5 latent semantics for actor, movie and year have to be reported. For this, we consider the three factor matrices given by CP and display them in descending order of their latent features. We perform this for all three factor matrices, actor, movie and year.



```
# Latent Semantics for Year
latent_semantics_year = pd.DataFrame(columns=['year', 'ls1', 'ls2', 'ls3', 'ls4', 'ls5'])
latent_semantics_year['year'] = year
latent_semantics_year['ls1'] = U[0][:, 0]
latent_semantics_year['ls2'] = U[0][:, 1]
latent_semantics_year['ls3'] = U[0][:, 2]
latent_semantics_year['ls4'] = U[0][:, 3]
latent_semantics_year['ls5'] = U[0][:, 4]

print 'Latent Semantic for Year sorted by LS1'
ls1 = latent_semantics_year.sort_values(by='ls1', ascending=False)
print ls1
print 'Latent Semantic for Year sorted by LS2'
ls2 = latent_semantics_year.sort_values(by='ls2', ascending=False)
print ls2
print 'Latent Semantic for Year sorted by LS3'
ls3 = latent_semantics_year.sort_values(by='ls3', ascending=False)
print ls3
print 'Latent Semantic for Year sorted by LS4'
ls4 = latent_semantics_year.sort_values(by='ls4', ascending=False)
print ls4
print 'Latent Semantic for Year sorted by LS5'
ls5 = latent_semantics_year.sort_values(by='ls5', ascending=False)
print ls5
```

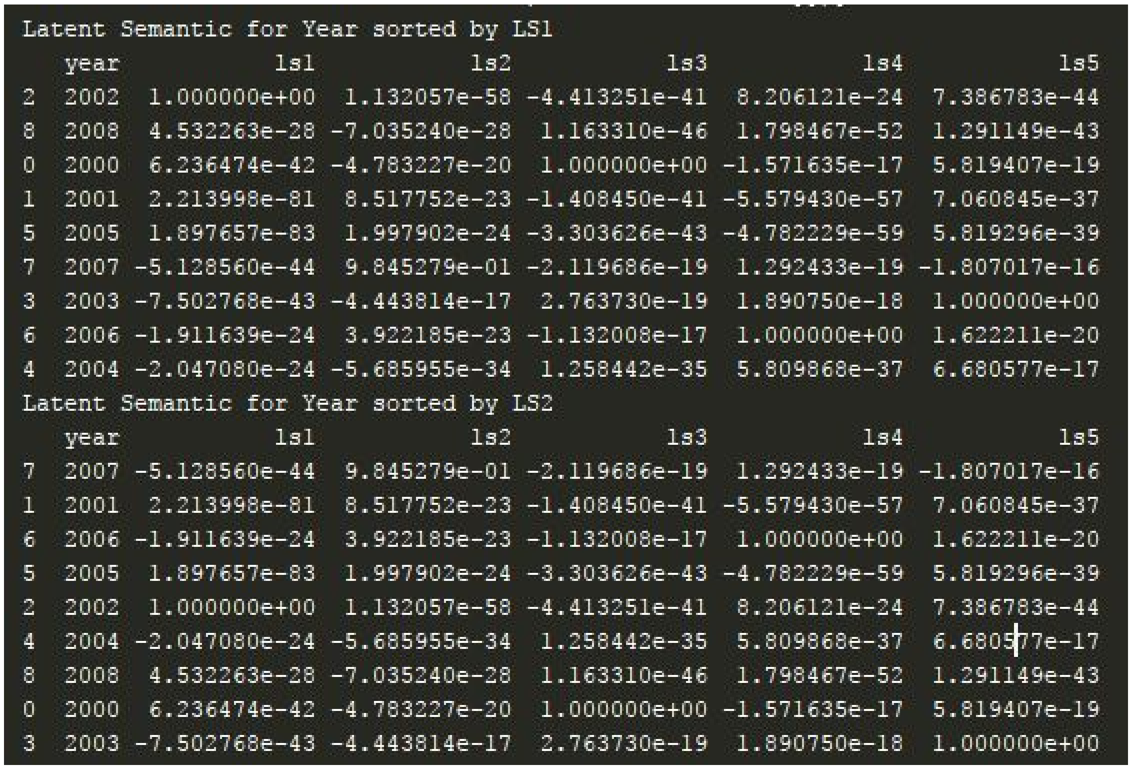
Find non-overlapping groupings

To find the non-overlapping groupings for the factor matrices, we have used KMeans. We implemented KMeans using the sklearn library, giving the number of clusters (equal to 5), and giving seed as 0. The output would be a list of the cluster each element would belong to, with range being 1-5. We then display the different clusters in separate lists.

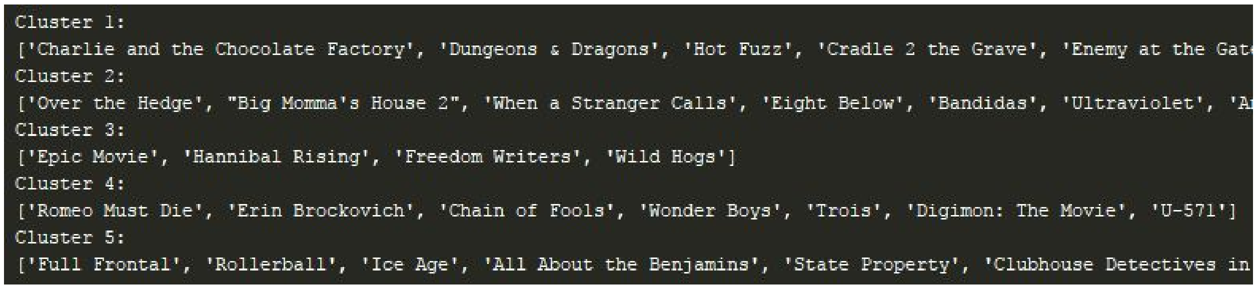


```
# Clusters for year
kmeans_year = KMeans(n_clusters=5, random_state=0).fit(U[0])

c1 = []
c2 = []
c3 = []
c4 = []
c5 = []

for i in range(0, len(kmeans_year.labels_) - 1):
    if kmeans_year.labels_[i] == 0:
        c1.append(year[i])
    elif kmeans_year.labels_[i] == 1:
        c2.append(year[i])
    elif kmeans_year.labels_[i] == 2:
        c3.append(year[i])
    elif kmeans_year.labels_[i] == 3:
        c4.append(year[i])
    elif kmeans_year.labels_[i] == 4:
        c5.append(year[i])

print 'Cluster 1:'
print c1
print 'Cluster 2:'
print c2
```

Output
Partial output for latent semantics for year


```
Latent Semantic for Year sorted by LS1
   year           ls1           ls2           ls3           ls4           ls5
2  2002  1.000000e+00  1.132057e-58 -4.413251e-41  8.206121e-24  7.386783e-44
8  2008  4.532263e-28 -7.035240e-28  1.163310e-46  1.798467e-52  1.291149e-43
0  2000  6.236474e-42 -4.783227e-20  1.000000e+00 -1.571635e-17  5.819407e-19
1  2001  2.213998e-81  8.517752e-23 -1.408450e-41 -5.579430e-57  7.060845e-37
5  2005  1.897657e-83  1.997902e-24 -3.303626e-43 -4.782229e-59  5.819296e-39
7  2007 -5.128560e-44  9.845279e-01 -2.119686e-19  1.292433e-19 -1.807017e-16
3  2003 -7.502768e-43 -4.443814e-17  2.763730e-19  1.890750e-18  1.000000e+00
6  2006 -1.911639e-24  3.922185e-23 -1.132008e-17  1.000000e+00  1.622211e-20
4  2004 -2.047080e-24 -5.685955e-34  1.258442e-35  5.809868e-37  6.680577e-17
Latent Semantic for Year sorted by LS2
   year           ls1           ls2           ls3           ls4           ls5
7  2007 -5.128560e-44  9.845279e-01 -2.119686e-19  1.292433e-19 -1.807017e-16
1  2001  2.213998e-81  8.517752e-23 -1.408450e-41 -5.579430e-57  7.060845e-37
6  2006 -1.911639e-24  3.922185e-23 -1.132008e-17  1.000000e+00  1.622211e-20
5  2005  1.897657e-83  1.997902e-24 -3.303626e-43 -4.782229e-59  5.819296e-39
2  2002  1.000000e+00  1.132057e-58 -4.413251e-41  8.206121e-24  7.386783e-44
4  2004 -2.047080e-24 -5.685955e-34  1.258442e-35  5.809868e-37  6.680577e-17
8  2008  4.532263e-28 -7.035240e-28  1.163310e-46  1.798467e-52  1.291149e-43
0  2000  6.236474e-42 -4.783227e-20  1.000000e+00 -1.571635e-17  5.819407e-19
3  2003 -7.502768e-43 -4.443814e-17  2.763730e-19  1.890750e-18  1.000000e+00
```

Partial output for movie groupings

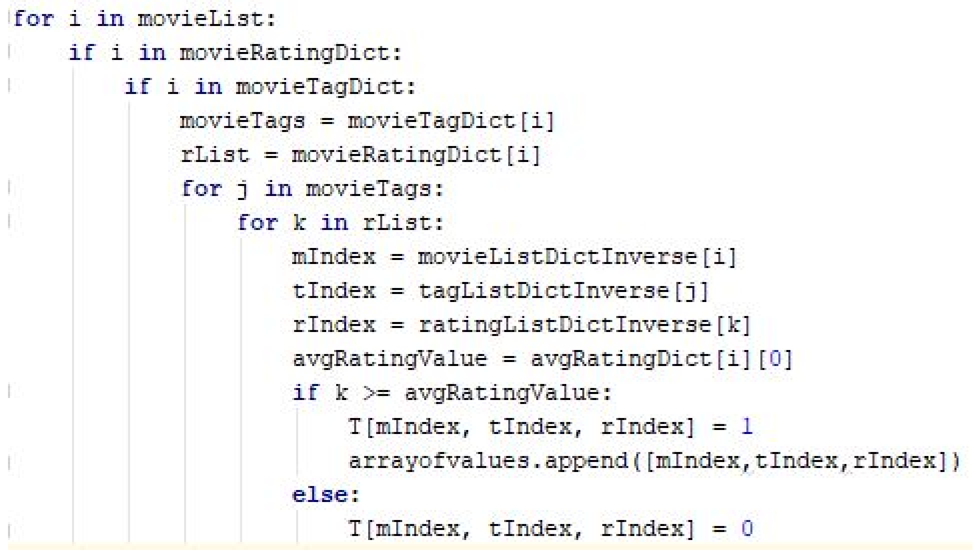

```
Cluster 1:
['Charlie and the Chocolate Factory', 'Dungeons & Dragons', 'Hot Fuzz', 'Cradle 2 the Grave', 'Enemy at the Gat
Cluster 2:
['Over the Hedge', "Big Momma's House 2", 'When a Stranger Calls', 'Eight Below', 'Bandidas', 'Ultraviolet', 'A
Cluster 3:
['Epic Movie', 'Hannibal Rising', 'Freedom Writers', 'Wild Hogs']
Cluster 4:
['Romeo Must Die', 'Erin Brockovich', 'Chain of Fools', 'Wonder Boys', 'Trois', 'Digimon: The Movie', 'U-571']
Cluster 5:
['Full Frontal', 'Rollerball', 'Ice Age', 'All About the Benjamins', 'State Property', 'Clubhouse Detectives in
```

TASK 2D

In this task, a tensor should be created, in which for any triple of movie-tag-rating, if a movie has that tag and the rating in greater than the average rating of the movie, the value at that index is 1 or else, it's 0. CP-decomposition also has to be performed on this tensor to report the top five latent semantics related to the three modes - movie, tag and rating. And finally, to create five non-overlapping groups of movie, tag and rating have to be formed, based on the latent semantics. The implementation of this task is done in Python 2.7 and the major libraries used are sktensor for tensor creation and CP-decomposition and sklearn.cluster for forming non-overlapping groups in the last step using KMeans.

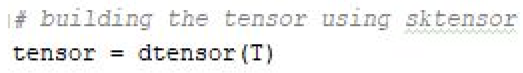
Step 1 - Formation of the tensor

In order to create a tensor, dictionaries are used to create a 3D matrix. First all the unique movies, tags and ratings are taken. The average rating for each movie has also been calculated, based on the ratings given by all the users. Then two dictionaries are created, both of which have movie as the key. As for the values for these dictionaries, for each movie in the dictionary, all possible unique tags associated with it are taken for the first dictionary and all unique ratings a movie has received from all users for the second one. Using these dictionaries, combinations of all possible triples for movie-tag-rating are made and these triples are appropriately assigned 1's and 0's, also considering the average rating of a movie.



```
for i in movieList:
    if i in movieRatingDict:
        if i in movieTagDict:
            movieTags = movieTagDict[i]
            rList = movieRatingDict[i]
            for j in movieTags:
                for k in rList:
                    mIndex = movieListDictInverse[i]
                    tIndex = tagListDictInverse[j]
                    rIndex = ratingListDictInverse[k]
                    avgRatingValue = avgRatingDict[i][0]
                    if k >= avgRatingValue:
                        T[mIndex, tIndex, rIndex] = 1
                        arrayofvalues.append([mIndex,tIndex,rIndex])
                    else:
                        T[mIndex, tIndex, rIndex] = 0
```

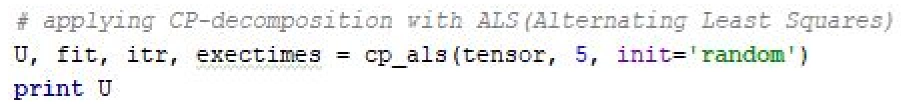
Sktensor's dtensor is used to convert the 3D numpy array to a tensor.



```
# building the tensor using sktensor
tensor = dtensor(T)
```

Step 2 - CP-Decomposition

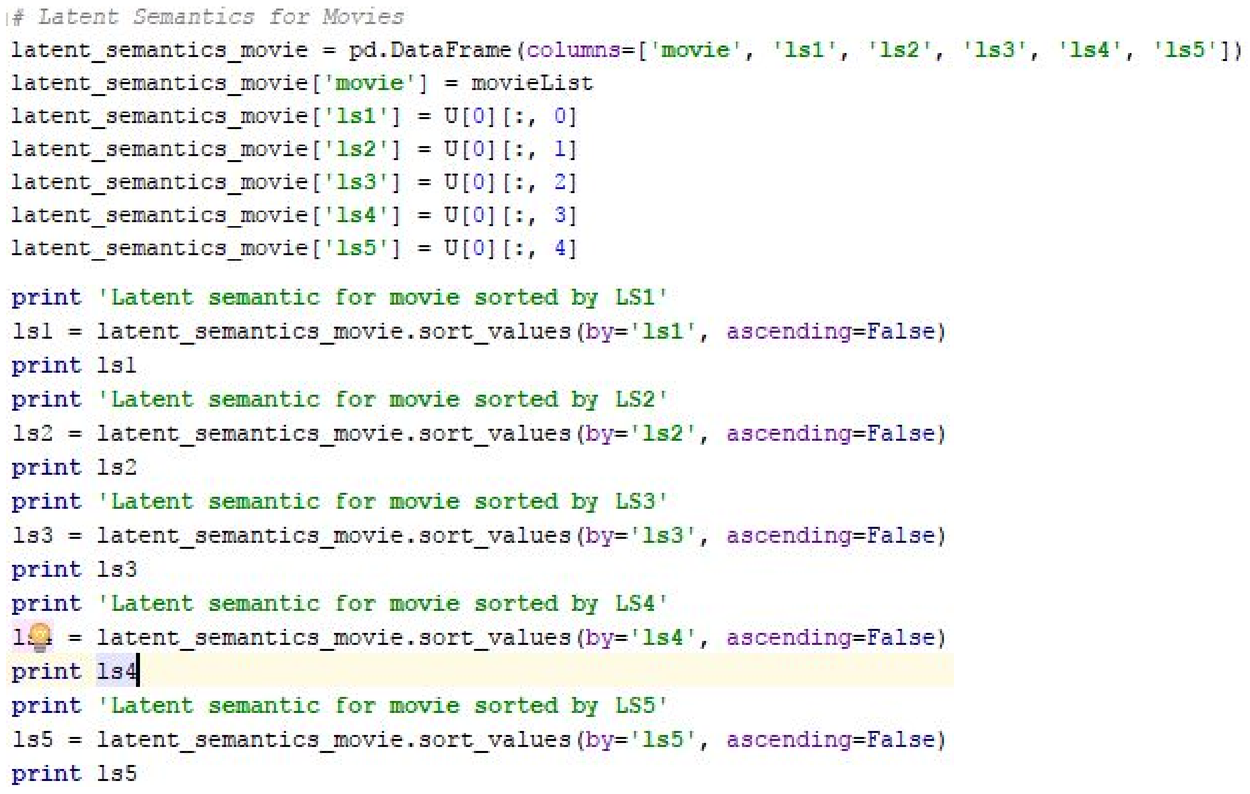
CP is performed using ALS(Alternating Least Squares). The function cp_als (from sktensor) is used to perform CP on the tensor obtained from step 1, with the rank set to be 5. The initialization of factor matrices takes place randomly.



```
# applying CP-decomposition with ALS(Alternating Least Squares)
U, fit, itr, exectimes = cp_als(tensor, 5, init='random')
print U
```

Step 3 - Report top 5 latent semantics

The factor matrices obtained in step 2 are used in this step to obtain the top 5 latent semantics. We have displayed the latent semantics for all the three modes - movie, tag and rating. For each of them, we have extracted the latent semantics from their corresponding factor matrix. For a particular mode, we considered each of the latent semantics, and displayed the results, sorted by each one of them.

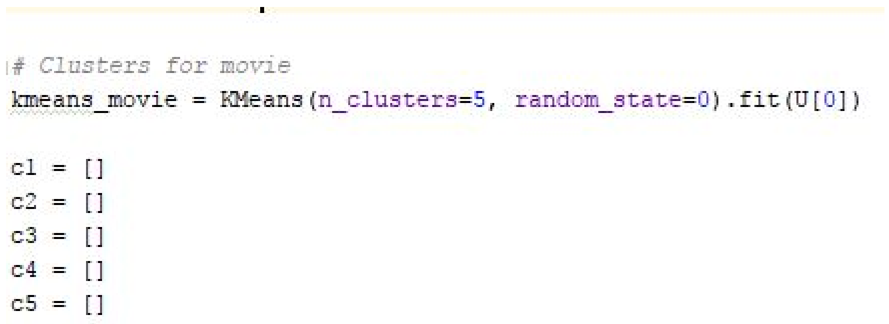


```
# Latent Semantics for Movies
latent_semantics_movie = pd.DataFrame(columns=['movie', 'ls1', 'ls2', 'ls3', 'ls4', 'ls5'])
latent_semantics_movie['movie'] = movieList
latent_semantics_movie['ls1'] = U[0][:, 0]
latent_semantics_movie['ls2'] = U[0][:, 1]
latent_semantics_movie['ls3'] = U[0][:, 2]
latent_semantics_movie['ls4'] = U[0][:, 3]
latent_semantics_movie['ls5'] = U[0][:, 4]

print 'Latent semantic for movie sorted by LS1'
ls1 = latent_semantics_movie.sort_values(by='ls1', ascending=False)
print ls1
print 'Latent semantic for movie sorted by LS2'
ls2 = latent_semantics_movie.sort_values(by='ls2', ascending=False)
print ls2
print 'Latent semantic for movie sorted by LS3'
ls3 = latent_semantics_movie.sort_values(by='ls3', ascending=False)
print ls3
print 'Latent semantic for movie sorted by LS4'
ls4 = latent_semantics_movie.sort_values(by='ls4', ascending=False)
print ls4
print 'Latent semantic for movie sorted by LS5'
ls5 = latent_semantics_movie.sort_values(by='ls5', ascending=False)
print ls5
```

This is done for tag and rating as well.

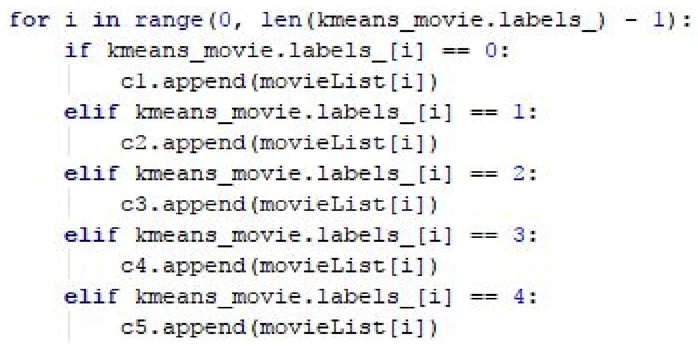
Step 4 - Formation of non-overlapping groups

In the final step, clusters of movie, tag and ratings are formed. KMeans algorithm is used for this purpose. The library used is sklearn. The number of clusters is given as 5. For each of this five clusters, a set of values are taken which belong to that group. No values are repeated.The output is shown in the form of arrays for each cluster.



```
# Clusters for movie
kmeans_movie = KMeans(n_clusters=5, random_state=0).fit(U[0])

c1 = []
c2 = []
c3 = []
c4 = []
c5 = []
```

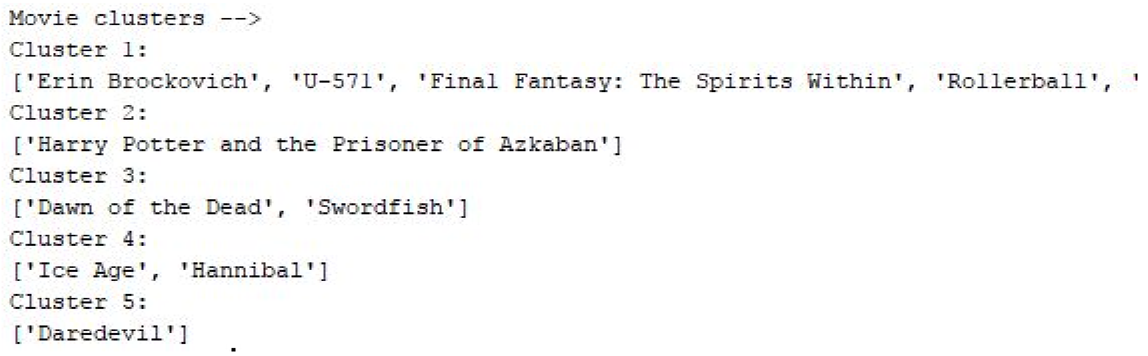
The first line in the above snapshot actually forms the clusters. A value in the range of 1-5(0-4 to be technical), has been assigned to all the values in the list of movies(tags/rating). But for better viewing of results and for better understanding, the 5 arrays are taken. Using simple comparison, the set is partitioned into five different groupings. The same thing is done for the case of tags and ratings.



```
for i in range(0, len(kmeans_movie.labels_) - 1):
    if kmeans_movie.labels_[i] == 0:
        c1.append(movieList[i])
    elif kmeans_movie.labels_[i] == 1:
        c2.append(movieList[i])
    elif kmeans_movie.labels_[i] == 2:
        c3.append(movieList[i])
    elif kmeans_movie.labels_[i] == 3:
        c4.append(movieList[i])
    elif kmeans_movie.labels_[i] == 4:
        c5.append(movieList[i])
```

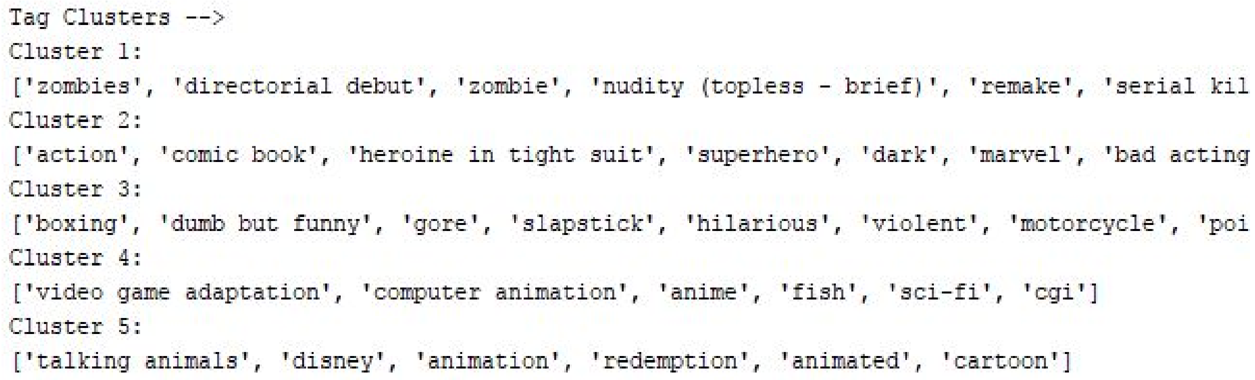
Outputs:

Cluster for Movies(Partial for better viewing)



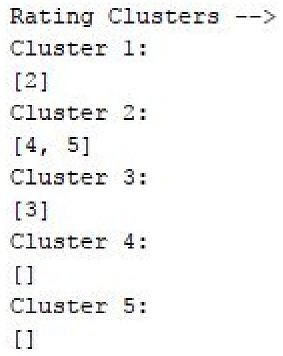
```
Movie clusters -->
Cluster 1:
['Erin Brockovich', 'U-571', 'Final Fantasy: The Spirits Within', 'Rollerball', '
Cluster 2:
['Harry Potter and the Prisoner of Azkaban']
Cluster 3:
['Dawn of the Dead', 'Swordfish']
Cluster 4:
['Ice Age', 'Hannibal']
Cluster 5:
['Daredevil']
```

Cluster for Tags



```
Tag Clusters -->
Cluster 1:
['zombies', 'directorial debut', 'zombie', 'nudity (topless - brief)', 'remake', 'serial kil
Cluster 2:
['action', 'comic book', 'heroine in tight suit', 'superhero', 'dark', 'marvel', 'bad acting
Cluster 3:
['boxing', 'dumb but funny', 'gore', 'slapstick', 'hilarious', 'violent', 'motorcycle', 'poi
Cluster 4:
['video game adaptation', 'computer animation', 'anime', 'fish', 'sci-fi', 'cgi']
Cluster 5:
['talking animals', 'disney', 'animation', 'redemption', 'animated', 'cartoon']
```

Cluster for Ratings (Only 5 ratings[1,2,3,4,5])



```
Rating Clusters -->
Cluster 1:
[2]
Cluster 2:
[4, 5]
Cluster 3:
[3]
Cluster 4:
[]
Cluster 5:
[]
```

TASK 3

TASK 3A

Goals:

- Create an actor-similarity matrix using tag vectors in the data.
- Given a seed of set actors that user is interested in, we have to find 10 most related actors to these actors using random walk with restarts.

Task 3b
Goals:

- Create a co-actor-co-actor matrix based on number.of movies two actors have acted together in.
- Given a seed of set actors that user is interested in, we have to find 10 most related actors to these actors using random walk with restarts.

For both task 3a and task 3b we have to rank actors based on their relationships which could be either:

1. Similarity between actor
2. Co-acting relationship between actors.

In addition since we have a set of seed actors, we will have to use the concept of personalized pagerank(PPR) which is implemented using random walk with restarts.(RWR)

First let’s examine a research paper that uses a similar concept for another application:

In the paper titled Automatic multimedia cross-modal correlation discovery by J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu the technique of random walk is used for assigning captions to images, videos or audio clips[7]

Suppose there are 3 images :
A: captions c1,c2,c3,c4
B: captions c3,c4,c5,c6
C :no captions

- We need to assign captions to C
- We know relationships between A,B,C and based on those relationships we need to assign captions.

Now, we can achieve this using a random walk with restarts approach

- We store the similarity values of A, B and C in a matrix called as affinity matrix.
- By normalizing the affinity matrix we can convert it into a transition matrix that signifies the contribution of each caption to its affinity measure.
- Finally we can implement a random walker that takes C as the seed node and computes the ranks of captions of A and B with respect to C.
- Among those ranked captions, we can then choose the top k desired captions for the image C.

The computation of personalized pageranks using random walk with restarts is done using the formula:

P = (alpha) G P+(1-alpha) S

- In a random walk for personalized pagerank, the matrix G is the transition matrix that stores the normalized affinity values for all nodes in the data.
- ‘A random walker must always keep walking’- this is the idea used for construction of the formula.
- With a probability of alpha the random walker follows the graph and finds pageranks of nodes.
- But there may arise a situation that there are no more paths possible if a random walker reaches a dead end.
- Hence with a probability of 1-alpha the random walker jumps to any arbitrary node
- But for Random walk with restarts we are provided with a set of seed nodes that a user cared about .
- Hence the jumps with probability of 1-alpha are made to the seed nodes instead of any arbitrary nodes.
- Alpha is known as the teleportation probability and S is the teleportation seed vector
- Thus a random walk will indicate to us the importance of other nodes in the graph with respect to the seed nodes.
- We usually iteratively refine the pagerank using random walker until the error has converged to minimum.
- In the paper mentioned before it was experimentally found that alpha values between 0.8 and 0.9 give better results for RWR.[7]

Description of implementation :

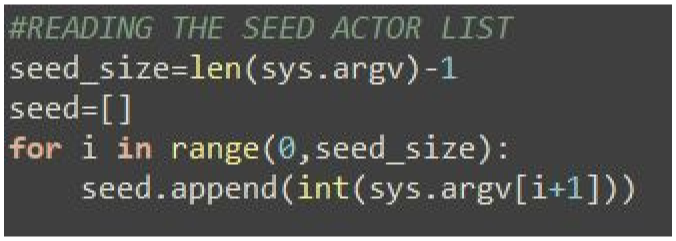
Task 3a and b:

For forming the actor-actor similarity matrix:
The actor-actor similarity matrix was taken as output from part 2a

For forming the co-actor-co-actor matrix:
1. A dictionary is formed having key values as actor id and list of movies played by that actor as values.
2. For every 2 actors in the database an intersection between their key-value pairs is computed.
3. The count of this key value pair is stored in the corresponding row of the co-actor-co-actor matrix

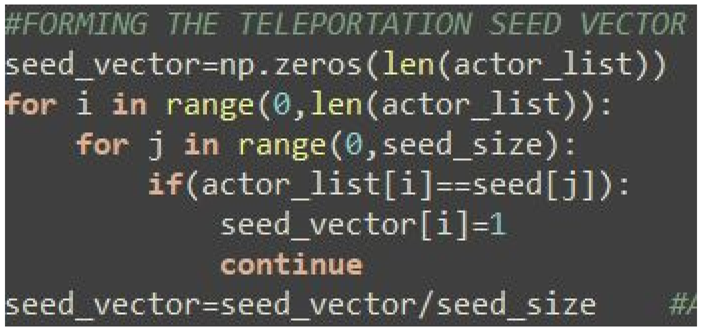
The function used for computing this matrix is symmetric and hence the matrix will also be symmetric, i.e.
co_co(a1,a2)=co_co(a2,a1)

We take seed nodes from the command line using the code segment:



```
#READING THE SEED ACTOR LIST
seed_size=len(sys.argv)-1
seed=[]
for i in range(0,seed_size):
    seed.append(int(sys.argv[i+1]))
```

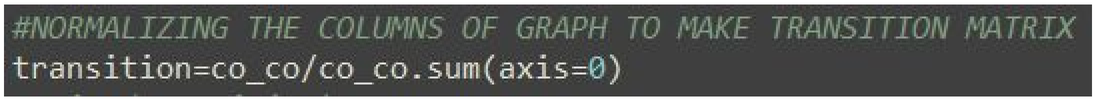
We form a seed vector by assigning the values to the seed teleportation vector:



```
#FORMING THE TELEPORTATION SEED VECTOR
seed_vector=np.zeros(len(actor_list))
for i in range(0,len(actor_list)):
    for j in range(0,seed_size):
        if(actor_list[i]==seed[j]):
            seed_vector[i]=1
            continue
seed_vector=seed_vector/seed_size    #A
```

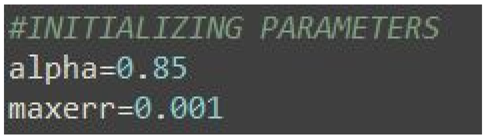
For computing personalized pageranks we will need to convert the co-actor-co-actor matrix into a transition matrix

This can be done by normalizing the columns of the co-actor-co-actor matrix:



```
#NORMALIZING THE COLUMNS OF GRAPH TO MAKE TRANSITION MATRIX
transition=co_co/co_co.sum(axis=0)
```

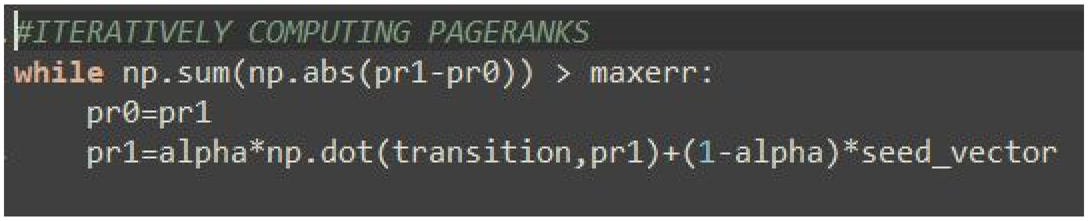
As mentioned before alpha should be between 0.8 and 0.9. Thus we have chosen it as 0.85 and maximum allowed error as 0.001:



```
#INITIALIZING PARAMETERS
alpha=0.85
maxerr=0.001
```

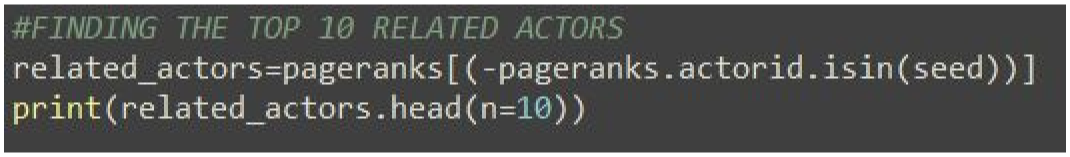
Now we find pageranks using the parameters and transition matrix and keep using the random walker until the error is tolerable.

Thus we are refining the pageranks by restarting at our seed nodes:



```
#ITERATIVELY COMPUTING PAGERANKS
while np.sum(np.abs(pr1-pr0)) > maxerr:
    pr0=pr1
    pr1=alpha*np.dot(transition,pr1)+(1-alpha)*seed_vector
```

Finally we display the top 10 related actors to the seed actors:



```
#FINDING THE TOP 10 RELATED ACTORS
related_actors=pageranks[(-pageranks.actorid.isin(seed))]
print(related_actors.head(n=10))
```

3a Input: python task3a.py 1342347 396877 623809 3704908 2392683

3a Ouput:

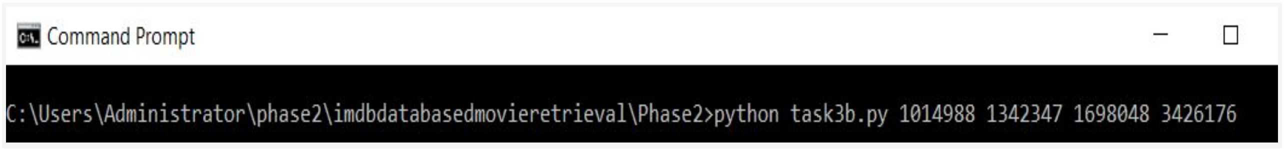
```
Please input 1-5 to choose the similarity/distance measure to use for finding similarity:
1: Manhattan Distance
2: Cosine Similarity
3: Cosine Distance
4: Mahalanobis Distance
5: Euclidean Distance
Note: Anything besides these will automatically consider Cosine Similarity
Enter here: 4
```

```
Do you want to ignore actors with whom no tags were associated?
0 to ignore, anything else to include: 0
        actorid  pagerank          name
48  1698048  0.010895      [Oldman, Gary]
75  2665816  0.010869      [Berry, Halle]
80  2946264  0.010820      [Ferris, Pam]
20   692749  0.010713   [Farrell, Colin]
22   713207  0.010689  [Fiennes, Joseph]
8         316365  0.010686   [Buscemi, Steve]
41  1297720  0.010684      [Leary, Denis]
50  1792455  0.010605      [Phifer, Mekhi]
49  1726318  0.010551   [Paetkau, David]
24   721841  0.010509      [Fisher, Tom]
```

This output was chosen on the bases of only actors that had tags associated with it and Mahalanobis distance was used.

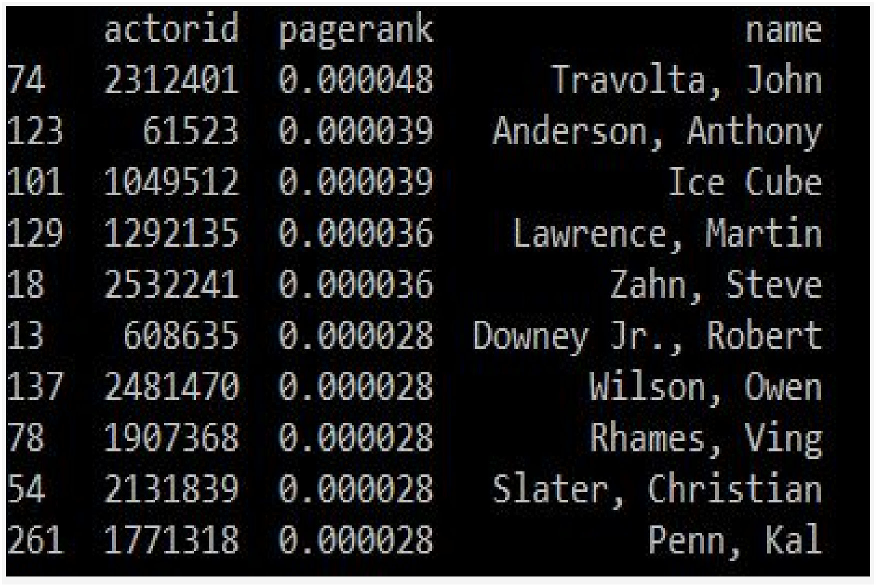
Sample input:

```
python task3b.py <actorid1> <actorid2>.....
```



```
C:\Users\Administrator\phase2\imdbdatabasedmovieretrieval\Phase2>python task3b.py 1014988 1342347 1698048 3426176
```

Sample output:



```
     actorid  pagerank               name
74   2312401  0.000048     Travolta, John
123    61523  0.000039  Anderson, Anthony
101  1049512  0.000039           Ice Cube
129  1292135  0.000036    Lawrence, Martin
18   2532241  0.000036        Zahn, Steve
13    608635  0.000028  Downey Jr., Robert
137  2481470  0.000028       Wilson, Owen
78   1907368  0.000028       Rhames, Ving
54   2131839  0.000028  Slater, Christian
261  1771318  0.000028          Penn, Kal
```

TASK 4

To recommend top similar movies to users based on his previous history we need 2 types of input matrix which compares movies in the space of tags and actors in the space of tags. From this we will be able to get both similar movies and also via similar actor.

Get Input matrix

For any given new data set Actor's TF-IDF tag vectors and Movie's tag vectors are calculated first using methods discussed in Phase 1 project. This model was stored in a csv file which has actor id/movie id, tags associated with this actor and its corresponding TF-IDF weights. This is used to calculate actor's or movie's tag vector. The cell where an actor/movie has no tag will be assigned as zero. Thus, our input matrix will have all actors in rows and all tags in column and its corresponding TF-IDF values. Then we apply PCA method to reduce the dimensionality of the data.

Apply PCA on this input matrix

We have decided to go with PCA for dimensionality reduction because when we consider similarities between objects, it makes more sense to preserve the variance than the data. Step by step implementation of PCA [6] is as follows

- Find the covariance matrix from both actors and movie's tag vectors matrix
- Get eigenvalues and eigenvectors as pairs from this covariance matrix.
- These pairs are sorted based on eigenvalues in descending order. We cannot assume always the Eigen decomposition will give sorted Eigen values and Eigen vector pair. If the TF-IDF values are not normalized then we get random or unsorted Eigen value vector pair
- We decide number of dimensions to keep based on the values obtained from a factor called "explained variance". This is calculated as follows
    - Total_sum = sum(eig_vals)
    - Exp_var = [(index / tot) * 100 for index in eig_vals]
    - cum_var_exp = np.cumsum(var_exp)
- The explained variance tells us how much information (variance) can be attributed to each of the principal components. In order to hold 100% of variance we keep the threshold as 100 for this cum_var_exp and keep adding top dimensions until this number is reached
- Here, we are reducing the m-dimensional feature space to a k-dimensional feature subspace (k<<m), by choosing the "top k" eigenvectors with the highest eigenvalues

Find Similar Movies

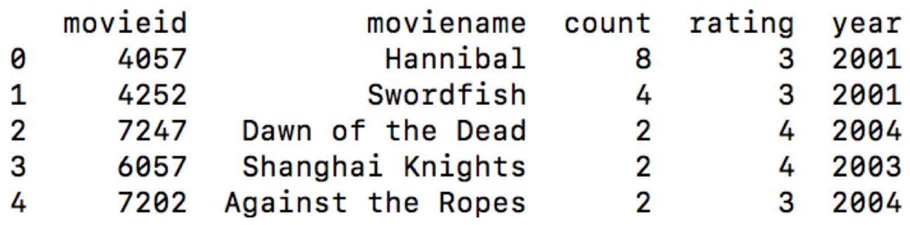
- For each given movie id as input we calculate distance (Euclidean) between each point which will be a movie object in the reduced dimension space
- For each given movie id as input we get the actor with highest rank. Keeping this as base point we calculate the distances (Euclidean) between each point which will be an

actor object in the reduced dimension space. We then reverse map these related actors to their movie

- Combine the movies obtained from above two steps and combine into a single set without duplicates. Now these movies are somehow related to the inputted movies either by actors or by movie content
- This set of movies might contain movies under genre that were of no interest to the user. This information can be extracted by retrieving genres of input movie ids and remove movies that do not fall under this set of genres.
- We keep track of number of times a movie is selected as related movie. This means if user has watch two movies and a new movie is selected as similar to both these two movies then count of the new movie becomes two.
- We also retrieve rating of all selected movies and year of release of these selected movies
- Based on these 3 factors (count, rating, year) we sort the list and retrieve top 5 movies. If two movies have same count then we compare it with rating and if this is also equal we go for year of release.

Output

· Here user has watched “Enemy at the Gates” and “Collateral Damage” movie which was given as input. The movies are ranked based on count, rating and year as shown in the output and all movies are related to either Action movies or related based on actor.



```
   movieid            moviename  count  rating  year
0     4057             Hannibal      8       3  2001
1     4252            Swordfish      4       3  2001
2     7247     Dawn of the Dead      2       4  2004
3     6057     Shanghai Knights      2       4  2003
4     7202   Against the Ropes      2       3  2004
```

SYSTEM REQUIREMENTS

- Windows/ mac OS X/ Linux
- RAM – 4GB or higher to run Python
- Computer should have Python installed in it
- Computer should have scikit-learn, scikit-tensor, numpy, numpy+mkl and pandas libraries installed in it. These can be installed using pip on terminal/command line, or with direct download of respective .whl files.
- The execution of the project was done on mac OS X El Capitan, Windows 10 and Linux Kali 17.2.

RELATED WORKS

- According to author Kuan Liu [8] Similarity Learning for High-Dimensional Sparse Data talks about an efficient approach to get similarity functions for sparse high dimensional data. Similarity can be perceived as a combination of sparse basis elements that only operate on two features then Frank-Wolfe algorithm is used. This approach showed to be robust, efficient and effective on real world data. This can be used a more general preprocessing technique before applying other machine learning algorithms.

- According to Sebastian Raschka [6] in most of the machine learning algorithms the size of the dataset is the main problem or bottleneck for the performance. Using PCA in these datasets we can identify patterns in data and also determine how one data is related to other data. Also, when strong correlation between data points exists, the attempt to reduce the dimensionality comes into picture. PCA will play a fine role in this area. According to the author finding the directions of maximum variance in high-dimensional data and project it onto a smaller dimensional subspace while retaining most of the information is all about PCA. According to the author the steps involved in PCA are Standardize the data, obtain eigenvalues and eigenvector from the covariance matrix, sort the eigenvalues in descending order and chose k eigenvalues out of m dimensions, construct and new projection matrix from the selected k eigenvectors.
- According to author in Huang, Feiping Nie, Heng Huang [9] A New Simplex Sparse Learning Model to Measure Data Similarity for Clustering has suggested the use of properly defined constraints on the dataset to compute similarity/distance/afiniti between two datasets. This parameter free setting automatically produces the Laplacian graph and provides an efficient algorithm to solve the optimization problem. They proposed a spectral clustering (non parameterized) method robust to scale inconsistency and data noise. The projected gradient method was accelerated using a combination of Newton method for root finding and auxiliary variable.
- Hu, D.J., 2009. Latent dirichlet allocation for text, images, and music. *University of California, San Diego. Retrieved April*, *26*, p.2013.
  The paper is concentrated around exploring approaches rather than just bag-of-words to achieve more realistic models for classification of multimedia objects.
- The paper by D. Surian, N. Liu, D. Lo, H. Tong, E. P. Lim and C. Faloutsos, titled "Recommending People in Developers' Collaboration Network," *2011 18th Working Conference on Reverse Engineering*, Limerick, 2011, pp. 379-388. doi: 10.1109/WCRE.2011.53 proposes an application of Random Walk with restarts technique Based on an input developer,a list of top developers is recommended, that are most compatible based on their programming language skills, past projects and project categories they have worked on before, via a random walk with restart procedure.

CONCLUSION

The goal of the project for phase 2 is met. The information extracted from given data set is used to learn different concepts such as feature selection or dimensionality reduction in multimedia data retrieval and query optimization.

BIBLIOGRAPHY

[1] Gerald Salton and Christopher Buckley, “Term Weighing Approaches in Automatic Text Retrievals”, Information Processing and Management, 24(5):513–523, 1988.
[2] K. Selcuk Candan and Maria Luisa Sapino, “Data Management for Multimedia Retrieval”, Cambridge University Press, UK, 2010.

[3] October 22, 2017, https://pandas.pydata.org/pandas-docs/stable/comparison_with_sql.html

[4] September 22, 2017, https://en.wikipedia.org/wiki/Tf%E2%80%93idf

[5] September 22, 2017, https://en.wikipedia.org/wiki/Normalization_(statistics)

[6] September 22, 2017, https://plot.ly/ipython-notebooks/principal-component-analysis/

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

[8] September 22, 2017, http://proceedings.mlr.press/v38/liu15.pdf

[9] September 22, 2017, https://www.ijcai.org/Proceedings/15/Papers/502.pdf