**RECOMMENDER SYSTEM (PHASE 3)**

**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](https://en.wikipedia.org/wiki/Vector_space) of identifiers, such as, for example, index terms. It is used in [information filtering](https://en.wikipedia.org/wiki/Information_filtering), [information retrieval](https://en.wikipedia.org/wiki/Information_retrieval), [indexing](https://en.wikipedia.org/wiki/Index_(search_engine)) 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 recommend movies to user using various dimensionality reduction techniques like and to incorporate user feedback into the results. The other part of this project was used to classify movies based on labels given by user. The goal of each task is as follows

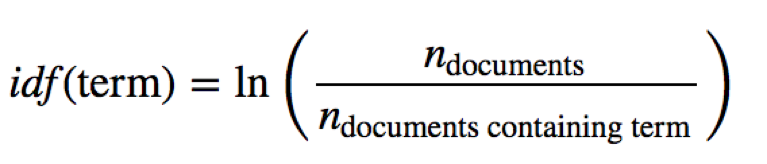
* The goal in task 1a is to recommend movies to user using PCA and SVD technique
* The goal of task 2a is to incorporate user feedback on the recommended movie and fine tune the result in next iteration
* The goal of task 5a is classify all unclassified movies in the dataset using list of sample labels provided by the user

**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

**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:



**TASK 1A**

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. Input matrix will have all movies in rows and all tags in column and its corresponding TF-IDF values. Then PCA is applied on this input matrix to reduce the dimension. Cosine similarity is applied to this reduced dimension matrix and all similar movies to the movies watch by user are ranked and top 5 movies is recommended to user.

**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.
* 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
* Reducing the m-dimensional feature space to a k-dimensional feature subspace (k<<m), by choosing the "top k" eigenvectors with the highest eigenvalues

**Apply SVD on this input matrix**

* The function ‘fit’ is used to load the SVD model with number of latent features (n\_components) selected as number of movies
* Function transform applies the SVD model to the N x M original dataset where N = number of objects and M = number of old features
* Left factor matrix (N x K) or the U matrix is returned where K = number of latent semantics after dimensionality reduction
* Eigen value is given by (K x K) or the S matrix
* Right factor matrix (K x M) or the V matrix is returned

**Apply Cosine Similarity on this reduced dimension matrix**

Cosine similarity is calculated for all pairs of movies which is rows in the matrix. Cosine similarity between two movies id the dot product of their corresponding vector. ‘sklearn’ **[10]** package is used to calculate this measure. The formula for cosine similarity is given by

Save this similarity matrix in ‘npz’ format for optimization purpose and use this computed similarity matrix in next successive iterations.

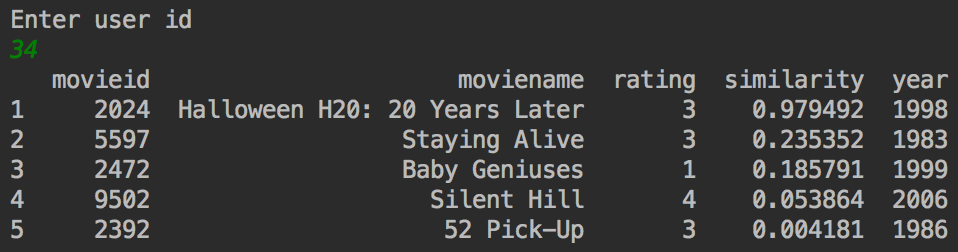
**Get top 5 movies for recommendation**

* For each movie watch by user get 10 similar movies by comparing their cosine similarity of that particular movie and all other movies
* Remove all movies watch by the user from this similarity list
* This list is sorted and first 10 movies is returned as similar movies
* In order to give preference to recent movies watch by the user sort the watched movies list based on timestamp
* Keep adding the most similar movie for this sorted watch movie list until 5 recommendation movies is obtained

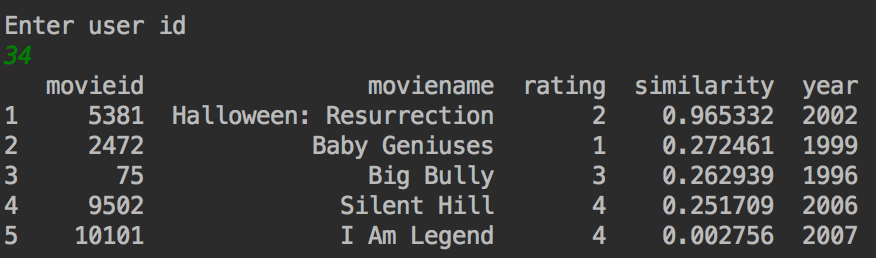
The same approach is used with SVD dimensionality reduction technique to get 5 recommendation movies.

**Output:**

* For user id 8 and PCA as dimensionality reduction technique top 5 recommended movies are as follows



* For user id 8 and SVD as dimensionality reduction technique top 5 recommended movies are as follows



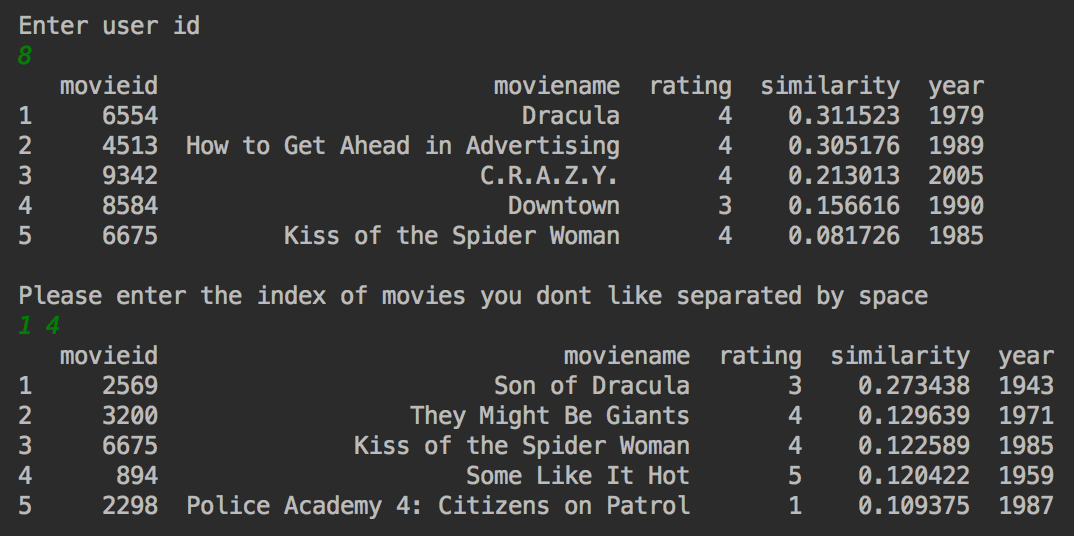
**TASK 2A**

Feedback for recommended movies in task 1a is obtained from the user where he inputs the movies that he doesn’t like. Based on this feedback the probability of similar movies to the movies he doesn’t like is calculated and those movies are avoided in the next successive iteration. Also, the probability of the similar movies to the movies he likes are increased in the next successive iteration.

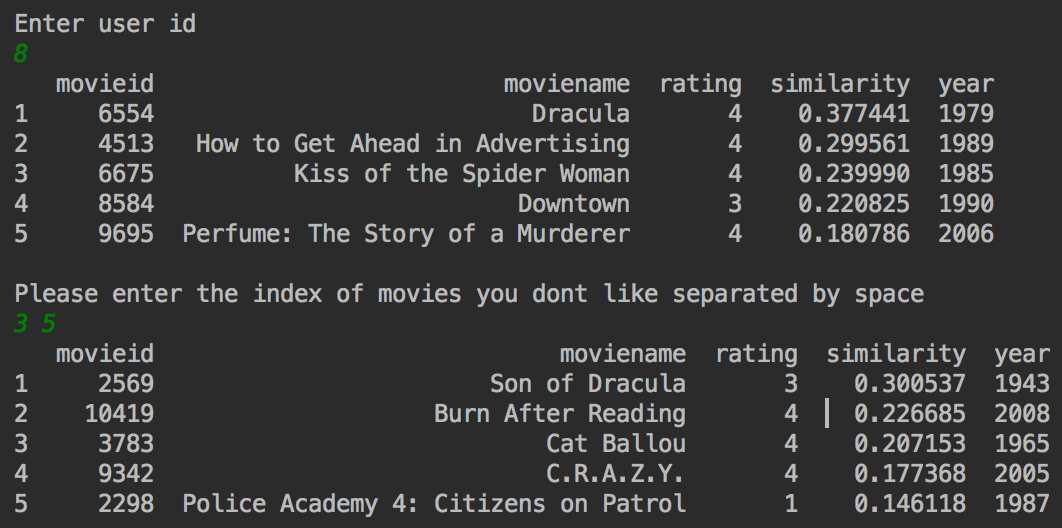
* Get list of movies user doesn’t like from the recommendation
* Get 10 similar movies to each of these movies and add it to list of movies to be ignored in the next iteration
* Get list of movies user has liked and add it to list of movies to be added in the next iteration.
* In each successive iteration when user dislikes the movie he previously liked then sway movies between these two lists

**Output:**

* For user id 8 and PCA as dimensionality reduction technique top 5 recommended movies are as follows



* For user id 8 and SVD as dimensionality reduction technique top 5 recommended movies are as follows



**TASK 1D:**

**Problem Statement:**

* For a set of movies watched by a given user, we must recommend 5 more movies to watch using Personalized PageRank algorithm.
* The user can then give feedback related to those movies which will facilitate the execution of Task 2.

Firstly, we need to form the movie-movie similarity matrix:

* Retrieve information about all movie watched by a user
* The movie ids will be used in the seed vector for PageRank seed vector
* Form the movie-tag matrix. The values in the movie tag matrix will be time -weighted tf-idf values:

In this method, we calculate term frequency using the formula:

Tf = no. of times tj occurs in a document

Total no. of terms in the document

Here document is a movie

We can calculate inverse document frequency using the formula:

Here, N = total no. of documents in the corpus

n = no. of documents the term is associated with the term

For time weighing, time stamps associated with the tag were converted to epoch values. Then those epoch values are converted into weight by using the formula:

Weight=epoch-min(epoch)/(max(epoch)-min(epoch))

We fill the entire matrix with time weighted tf-idf values of each movie and each tag.

Now, we need to form movie-movie similarity matrix

Let D=movie-tag matrix,

D.Transpose=Tag-movie matrix

Thus (D **.** ( D **.** Transpose ) ) will give us the movie-movie similarity matrix

Now we will have to use 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.

**Description of Implementation:**

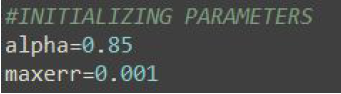
First we need to convert the movie\_movie similarity matrix to a transition matrix,

This can be done by normalizing the columns of the movie-movie similarity matrix:



As mentioned before alpha should be between 0.8 and 0.9. Thus we have chosen it as 0.85 and

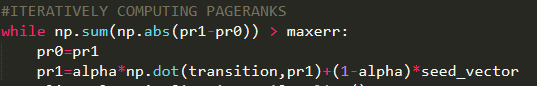
maximum allowed error as 0.001:



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:

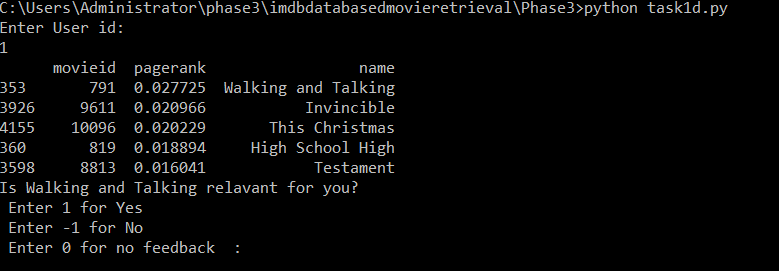


Finally we display the top 5 related movies to the seed movies:



**Sample input and output for this task :**

**Command:** python task1d.py



As we can see the output provides and interface for user to give feedback regarding the movies retrieved which will be evaluated for modification of results in Task 2d

**TASK 2D:**

**Problem statement:**

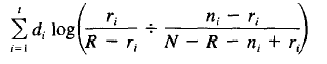
* Based on the movies retrieved in Task 1d evaluate user feedback regarding those movies.
* Based on user feedback refine the results to display an improved ranking of movies.

**User feedback:**

* We have allowed the user to give 3 types of feedback:
  1. The user can say that the movie was relevant
  2. The user can say that the movie was irrelevant
  3. The user does not label movie as relevant or irrelevant

We try to alter the tf-idf weights for all tags contained in previously retrieved movies.

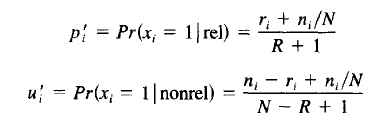
We calculate probabilistic relevance feedback using the formula:



Here we are considering term as the tags of a movie and document as the movie.

* di is the tf-idf weight of term in the movie
* ri is the no. of relevant movies in which the term occurs
* R is the total no. of movies that are relevant
* ni is the total no. of movies in which the term occurs
* N is the total no. of movies retrieved

For dealing with edge cases we have used:



Relevance feedback modification= ∑di log(pi’/ui’)

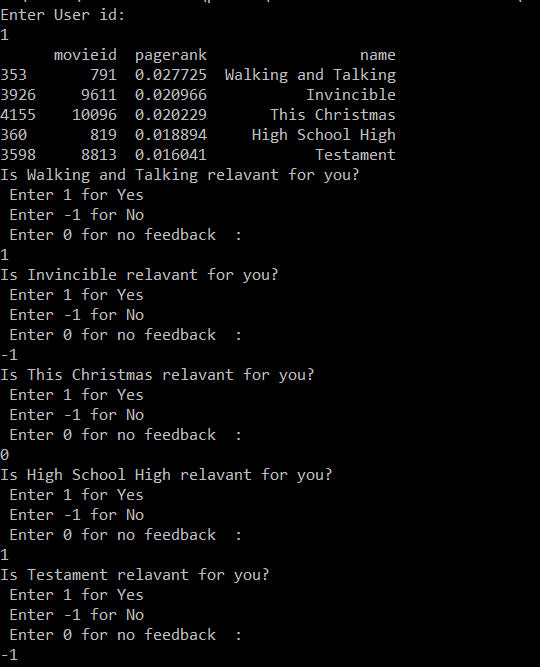
* The idea we have used is to give higher weights to tags that are present in relevant movies.
* After receiving user feedback, the weights of tags in the movie-tag matrix will be improved for tag which is more present in relevant movies.
* For tags present more in irrelevant movies, the weight values will be penalized so that these movies fall lower in ranking
* For no feedback, the change in ranking will occur if the tags of that movie appear in other relevant or irrelevant movies.
* We keep on asking user for feedback and refining results

Sample input and output:

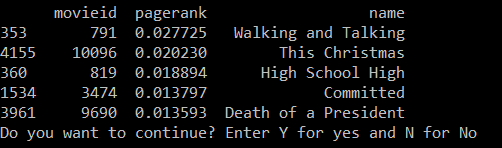
Command: python task1d.py

This task starts after PageRank finishes for the first time.

Sample output:



After applying feedback:



**TASK 3:**

* Primarily the main role of any information retrieval system is searching through data for relevant results
* There can be two types of searches:
  + Nearest Neighbor Search
  + Range Search

For searching through a space, we can iterate through every point through a brute force approach. However, this is not feasible since there may be large number of points and it will take very large amount of time to examine each and every one of them.

Thus, there is a need for pruning the search space to reduce the number of lookups,

For pruning, we need no have some total order, which is achievable using indexing techniques

Techniques like R-trees can be used for that. But the problem is that the method does not scale to very high dimensions.

In very high dimensions, searching using these index structures is equivalent to a sequential scan through the data. Thus, the probability of misses is very high in higher dimensions.

Thus, we need a method that is locality sensitive and requires fewer lookups.

**Locality Sensitive Hashing:**

This is used to form a data structure that enables search through the data space using an approximation algorithm

Although we lose on accuracy by using approximation, we significantly gain on performance on search. Also for all the inaccurate results we can cross verify produce corrected results.

**Some concepts:**

**Hash function:**

Function that maps data onto a hash value. This hash value can then be used to look for data in linear time.

**Hash Family:**

Let o1 and o2 be two objects in the database,

A hash family is a group of functions that follows these properties:

Probability[h(o1)=h(o2)] is directly proportional to similarity(o1,o2)

Thus similar objects must fall into the same bucket

**For approximation algorithm:**

Suppose we are looking for nearest object within a radius R, Then an approximate algorithm introduces an approximation factor c .

Thus the algorithm puts into the same bucket, all the objects that are within cR.

Then we have to perform a brute force search in the object returned.

dist(o1,o2) <=r probability of collision is high

dist(o1,o2)>=r probability of collision is low

However, using a single hash function may not be accurate since it may include false positives.

Thus, we use a conjunction of hash functions

Suppose h1 has false positive rate of 0.2 and h2 has false positive rate of 0.2 ,

Then a conjunction h1^h2 will have false positive rate of 0.04

Thus, we will get an improvement in results.

But sometime due to bucket size we might end up missing some values,

To avoid this, we use layering of hash functions. We would create multiple hash table and then take a union of all results for results obtained by lookup and then find nearest neighbors between them

L1 h11 ^ h12 ^ h13….^h1k

L2 h21 ^ h22 ^ h23….^h2k

L3 h31 ^ h32 ^ h33….^h3k

L4 h41 ^ h42 ^ h43….^h4k

**Description of Implementation:**

First we need to reduce dimensionality of the space to 500 dimensions.

For achieving this we have used Singular Valued Decomposition.

Since the movie-tag matrix is a sparse matrix we have used ‘svds’ which is a sparse svd package.

from scipy.sparse.linalg import svds, eigs

u, s, vt = svds(movie\_tag\_matrix, k=500)

Thus, we get a reduced space with latent dimesions.

For the implementation of hashing we have used random projection Locality sensitive hashing.

We first take input from the user regarding the number of layer L and the number of hash functions per layer k

For a hash function in a hash family, a hash value is derived using the following steps:

1. Take a random vector whose dimensionality is 500 and whose entries are taken from a p-stable distribution. We have used the gaussian normal distribution for our implementation
2. Take the dot product of this vector with the point in the 500 dimensional space
3. The bucket hash value is obtained by conjuction of hash values of all dot products
4. Form multiple layers using varying bucket sizes

The hash families and the hash tables are stored in a in -memory data structure.

For searching nearest neighbors of a query point we execute the following steps:

1. Find the hash bucket value of the query point in every table using hash families.
2. Find the elements contained in the same bucket as the query point and retrieve them
3. Search within the retrieved results for nearest neighbors

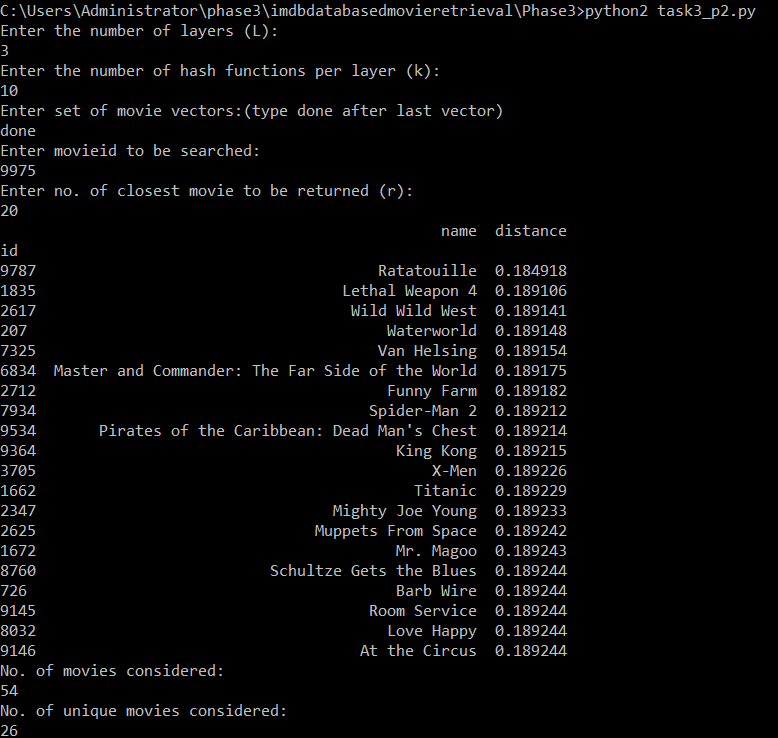
**Key point:**

Locality sensitive hashing is an index structure that can only be used for range searching. It cannot be directly be used for nearest neighbor searching. In order to use it for nearest neighbor search we confine our search to a neighborhood cR around the query point and then search for nearest neighbors within that range.

Sample input and output:

Command: python task3.py

Test run:



**Key note:**

To improve the accuracy of algorithm, we can perform some amount of preprocessing

We can find some tight clusters within data using k-means clustering and remove the clustered points from the dataset

We then apply LSH on the remaining points

Also, we apply LSH within the clusters.

This algorithm performs LSH in a data dependent way and does not assume that data is uniformly distributed.

Thus, we can find better results using this algorithm

Thus, without actually calculating distances of all points within database, we can perform nearest neighbor search using LSH.

**TASK 5A**

For any given data set and a sample of labeled movies a dictionary is created with movie id as key and label as value. For all non-labeled points in the dataset all neighbor data points within a radius is chosen and a collective response is taken to label this point. The similarity between two vectors is calculated by Euclidean distance formula given by

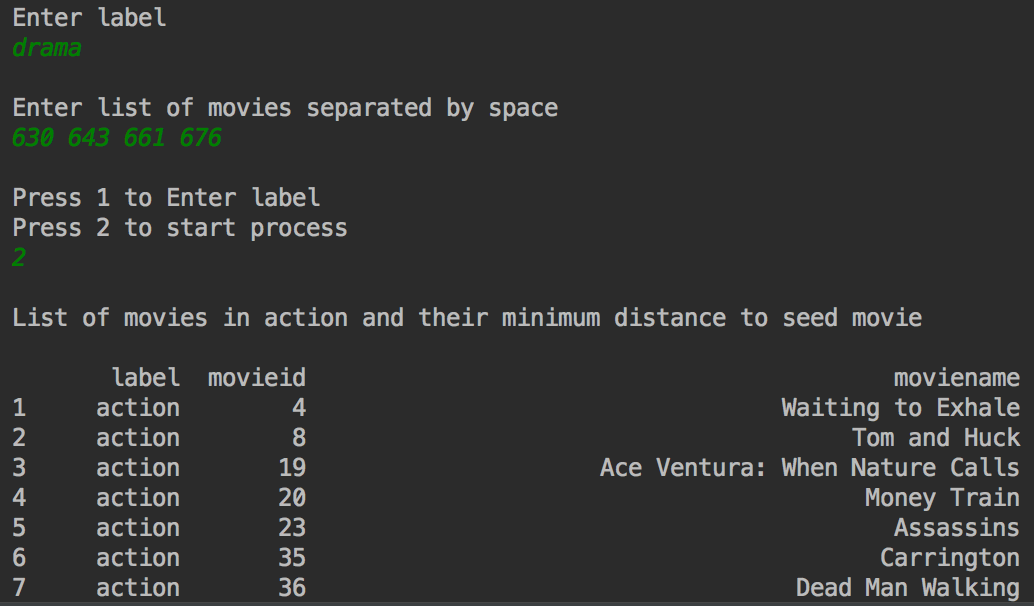
**Get Input matrix:**

Input matrix will have all movies in rows and all tags in column and its corresponding TF-IDF values. This matrix is inputted to ‘sklearn’ pairwise Euclidean distance function to calculate similarity matrix.

**Classify movies:**

* For each movie get all similar movies sorted in ascending order of distance
* Initial radius is set to unit 1
* Get all movies under this radius and check if any of these movies are labeled by user
* If not then increase the radius until a neighbor is labeled by user.
* Get votes of all labeled movies under this radius and the majority label will be assigned to this movie
* If a tie break happens then radius is further increased to get more candidates for voting
* When final radius is reached and still tie break happens assign any one among them randomly

**Output:**



**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 3 is met. The information extracted from given data set is used to learn different concepts such as indexing and classification 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>

[10]  September 22, 2017, <http://scikit-learn.org/stable/index.html>

[11]