**PHASE 2 GROUP 10 FALL 2017**

**GROUP MEMBERS:**

* Abhishek Thorat
* Krishna Chaitanya Bejjipuram
* Raviteja Upmaka
* Sachin Sundar Pungampalayam Shanmugasundaram
* Shiva Tejaswi Rampally
* Shubam Gondane

**ABSTRACT:**

This project is about developing a recommender engine on the Movie+Lens database. We will be performing different tasks like movie recommendations, implementing a probabilistic relevance feedback system, Locality Sensitive Hashing tool, r-nearest neighbor based relevance feedback and movie classification. We implement Singular Value Decomposition, Principle Component Analysis, Latent Dirichlet Allocation, CP decomposition, Personalized Page Ranking and combination of these to give movie recommendations for a user based on his watched movies data. Then we take the feedback from the user for the recommended movies and implemented probabilistic relevance feedback system to improve the recommendations to the user. We implement Locality Sensitive Hashing tool by mapping each movie into a 500-dimensional latent space and the user will be able to give positive/negative feedback to the result using which we perform r-nearest neighbor based relevance feedback algorithm and improve the matches. Movie classification is done using r-nearest neighbor based classification algorithm, decision tree based classification algorithm and n-ary SVM based classification algorithm.

**KEYWORDS:**

* SVD – Singular Value Decomposition
* PCA – Principal Component Analysis
* LDA – Latent Dirichlet Allocation
* CP Decomposition
* Personalized Page Ranking
* Probabilistic Relevance Feedback System
* LSH – Locality Sensitive Hashing
* R – nearest neighbor
* Decision tree
* N – ary SVM

**INTRODUCTION:**

TERMINOLOGY:

* Singular Value Decomposition: The singular value decomposition of a matrix A is the factorization of A into the product of three matrices A = UDV T where the columns of U and V are orthonormal, and the matrix D is diagonal with positive real entries.
* Principal Component Analysis: Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components (or sometimes, principal modes of variation).
* Latent Dirichlet Allocation: latent Dirichlet allocation (LDA), a generative probabilistic model for collections of discrete data such as text corpora. LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of topics.
* CP Decomposition: CP decomposition is generalization of SVD for tensors.
* Page Rank: Page Rank (PR) measures stationary distribution of one specific kind of random walk that starts from a random vertex and in each iteration, with a predefined probability p, jumps to a random vertex, and with probability1-p follows a random outgoing edge of the current vertex
* Personalized Page Ranking: Personalized Page Rank (PPR) is the same as PR other than the fact that jumps are back to one of a given set of starting vertices. In a way, the walk in PPR is biased towards (or personalized for) this set of starting vertices and is more localized compared to the random walk performed in PR.
* Tensor: Tensors are multidimensional arrays (generalization of one dimensional array)
* Probabilistic Relevance Feedback: It is a formalism of information retrieval useful to derive ranking functions used by search engines and web search engines to rank matching documents according to their **relevance** to a given search query.
* Locality Sensitive Hashing: is an algorithm for solving the approximate or exact Near Neighbor Search in high dimensional spaces.
* Decision Tree: A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences.
* Support Vector Machines: are a set of supervised leaning methods used for classification, regression and outlier’s detection.

**GOAL DESCRIPTION**:

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

TASK 1:

We need to implement a program such that using the complete information of movies the user had watched, we need to recommend the user 5 more movies to watch.

* **1a:** Implement the above-mentioned task using SVD or PCA.
* **1b:** Implement the above-mentioned task using LDA
* **1c:** Implement the above-mentioned task using Tensor Decomposition
* **1d:** Implement the above-mentioned task using Personalized Page Rank
* **1e:** Implement the above-mentioned task using a measure that is a combination of all the above methods.

TASK 2:

Relevance Feedback task: Using users feedback, we implement a probabilistic relevance feedback system and improve the accuracy of the matches from task 1a through 1e. We output the new set of ranked results.

TASK 3:

Multi-dimensional index structure and nearest neighbor search task:

* We need to implement a program which maps each movie in the system into a 500-dimensional latent space.
* We need to implement Locality Sensitive Hashing(LSH) tool, with inputs as
  + Number of layers, L
  + Number of hashes per layer, k
  + Set of movie vectors as input and creates an in-memory index structure containing the given set of vectors.
* We need to implement a similar movie search using this index structure: for a given movie and r, outputs the r most similar movies.

Note: User should be able to provide positive/negative feedback for the returned movie, which is used for task 4.

TASK 4:

NN – Relevance Feedback: Implement a r-nearest neighbor based relevance feedback algorithm to improve the nearest neighbor matches. The system should output the revisions it suggests in the relative importance of different parts of the query.

TASK 5:

Movie Classification: We need to classify the movies by implementing the following algorithms:

* a r-nearest neighbor based classification algorithm
* a decision tree based classification algorithm, and
* an n-ary SVM based classification algorithm

which takes a set of labeled movies and associates a label to rest of the movies in the database.

**ASSUMPTIONS:**

* User has watched a movie if he has either tagged or rated it.
* Movies don’t have genres repeated. A genre appears only once for a movie.
* For tasks involving tensor we are considering the following subset:
  + Mltags has been reduced to 10 percent of the original data file.
  + Mlmovies has been reduced to 30 percent of the original data file.
  + Along with this subset we have added corresponding records from the above files for users with userid >= 71500 and movies with year >= 2004.

**IMPLEMENTATION/DESCRIPTION OF PROPOSED SOLUTION:**

**TASK 1:**

**Task 1a:**

We have used movie-genre matrix as an object feature matrix for SVD Decomposition. For PCA, we have used covariance of movie-genre matrix. Also, object feature matrix is having values that are based on TF-IDF weighted graph.

**TF = no. of times a genre has appeared in movie / total no. of genre associated with movie**

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

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

Input is user id. Based on given user id we are finding all the movies he has watched from ratings and mltags table combined. Run SVD/PCA on given object feature matrix. After calculating SVD/PCA we find out the vectors that are associated with given user’s movie-list. Compare each vector in object feature matrix with retrieved vectors from user movie-list using adjusted cosine similarity. We took the movie-genre matrix as input and passed the movie tag matrix as an input to the SVD/PCA algorithm. We then remove the movies corresponding to the inputted movie from the U matrix obtained to obtain a matrix R. We then take the top 5 latent semantics from the decomposition matrix and then compare the object – latent semantic matrix with the given input movie vector to find the 5 most similar movies.

SVD

**Input –** Matrix M representing movie-genre relationship.

**Output – U, S , Vt**

**PCA**

**Input –** Covariance matrix representing movie-genre relationship. We pass the covariance matrix into the SVD function.

**Output – U, S , Vt**

**Task 1b:**

We have used movie-genre matrix as an object feature matrix for LDA Decomposition. Also, object feature matrix is just count of a particular feature in feature set for given object.

Input is user id. Based on given user id we are finding all the movies he has watched from ratings and mltags table combined. Run LDA on given object feature matrix. After calculating LDA we find out the vectors that are associated with given user’s movie-list. Compare each vector in object feature matrix with retrieved vectors from user movie-list using adjusted cosine similarity. We took the movie-genre matrix as input and passed the movie-genre matrix as an input to the LDA algorithm. We then remove the movies corresponding to the inputted movie from the U matrix obtained to obtain a matrix R. We then take the top 5 latent semantics from the decomposition matrix and then compare the object – latent semantic matrix with the given input actor vector to find the 5 most movies.

**Task 1c:**

Here are creating a 3-mode tensor where the modes are movies, tags and genre. The tensor can be represented using the n-dimensional array of numpy module in python. The value for the triple – movie-tag-genre will be 1 if the corresponding tag and genre belong to the same movie. We denote the 3-D array by ‘**T1**’.

**T1 🡪 (Movie, Tag, Genre)**

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

**T = dtensor(T1)**

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

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

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

X ∈ IRn1×n2×n3

X = [[λ; F, G, H]]

where λ ∈ IRr, F ∈ IRn1×r, G ∈ IRn2×r, and H ∈ IRn3×r

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

The Factor matrix used for recommendation is the movie factor matrix. For a given user ‘u’ we find the list of movies the user has watched and then we remove these movies M🡪 {m1, m2, m3 …, mn} from the factor matrix.

Next, we compare each of the user movies with the remaining set of movies in the Movie factor matrix. We are using cosine similarity to find the closest movies.

For m in M:

Find cosine similarity between m and remaining movies in Factor matrix

Since the movies are in 5-dimensional latent space the vectors representing these movies will also be 5 dimensional.

Now rank the movies based on similarity. The top 5 movies from this set is recommended to the user.

**Task 1d:**

we will use the movie-movie matrix as the transition matrix for random walk with restarts algorithm. The movies watched by the given user are considered as the seeds and a teleportation matrix for movies is create using the seeds. Weights are given to the seeds in the order of the time they were watched by the user and the year the movies were released.

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

**TF = no. of times a genre has appeared in movie / total no. of genre associated with movie**

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

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

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

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

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

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

**Task 1e:**

Here are creating a 3-mode tensor where the modes are movies, tags and genre. The tensor can be represented using the n-dimensional array of numpy module in python. The value for the triple – movie-tag-genre will be 1 if the corresponding tag and genre belong to the same movie. We denote the 3-D array by ‘**T1**’.

**T1 🡪 (Movie, Tag, Genre)**

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

**T = dtensor(T1)**

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

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

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

X ∈ IRn1×n2×n3

X = [[λ; F, G, H]]

where λ ∈ IRr, F ∈ IRn1×r, G ∈ IRn2×r, and H ∈ IRn3×r

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

This decomposed tensor of rank 5 is unfolded along one mode(Matricization) in our case the mode is movie. Let X` be the decomposed tensor.

M 🡪 Matricize(X`) along mode(movie)

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

**U, s, Vh = svd (M)**

We are considering the top 10 latent semantics which describe the movies, so we need the first 10 columns of the U matrix.

**U 🡪 U[:,10]**

Next, we construct the movie-movie matrix which will be used as transition matrix for pagerank.

**Movie-movie matrix = U.UT**

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

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

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

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

**TASK 2:**

**Task 2a:**

We have used movie-genre matrix as an object feature matrix for SVD Decomposition. For PCA, we have used covariance of movie-genre matrix. Also, object feature matrix is having values that are based on TF-IDF weighted graph.

In this task, for the 5 similar movies suggested by the task 1a, give relevance feedback (y=relevant, n=irrelevant, d=unlabeled). Based on this feedback, update the movie-genre matrix and calculate SVD/PCA and update the top 5 similar movies to the user. Also remove the movies that are already suggested and given feedback yes. To find the weight for genres after giving feedback, use the formula:

For positive feedback,

**wi = log( (ri + 0.5/ (R - ri + 1)) / ((ni - ri + 0.5) / (N – R – ni + ri + 1)))**

where, wi is the positive relevant weight of that genre after feedback,

ri is the count of relevant genre in relevant movies

ni is the count of relevant genre in total number of movies

R is the count of relevant movies

N is the count of total movies

For negative feedback, the formula is same but the variables become,

ri is the count of irrelevant genre in irrelevant movies

ni is the count of irrelevant genre in total number of movies

R is the count of irrelevant movies

After the feedback weight calculation, for each genre, update its column in the movie-genre matrix. For positive feedback weight, add the weight to the column’s values and for negative feedback weight, subtract the weight. With the updated movie-genre matrix, calculate the movie-movie matrix and do pagerank again as explained in task 1d.

**Task 2b:**

We have used movie-genre matrix as an object feature matrix for LDA Decomposition. Also, object feature matrix is just count of particular feature in feature set for given object.

Input is user id. Based on given user id we are finding all the movies he has watched from ratings and mltags table combined. Run LDA on given object feature matrix. After calculating LDA we find out the vectors that are associated with given user’s movie-list. Compare each vector in object feature matrix with retrieved vectors from user movie-list using adjusted cosine similarity. We took the movie-genre matrix as input and passed the movie-genre matrix as an input to the LDA algorithm.

In this task, for the 5 similar movies suggested by the task 1b, give relevance feedback (y=relevant, n=irrelevant, d=unlabeled). Based on this feedback, update the movie-genre matrix and calculate LDA and update the top 5 similar movies to the user. Also remove the movies that are already suggested and given feedback yes. To find the weight for genres after giving feedback, use the formula:

For positive feedback,

**wi = log( (ri + 0.5/ (R - ri + 1)) / ((ni - ri + 0.5) / (N – R – ni + ri + 1)))**

where, wi is the positive relevant weight of that genre after feedback,

ri is the count of relevant genre in relevant movies

ni is the count of relevant genre in total number of movies

R is the count of relevant movies

N is the count of total movies

For negative feedback, the formula is same but the variables become,

ri is the count of irrelevant genre in irrelevant movies

ni is the count of irrelevant genre in total number of movies

R is the count of irrelevant movies

After the feedback weight calculation, for each genre, update its column in the movie-genre matrix. For positive feedback weight, add the weight to the column’s values and for negative feedback weight, subtract the weight. With the updated movie-genre matrix, calculate the movie-movie matrix and do pagerank again as explained in task 1d.

**Task 2c:**

In this task, for the 5 similar movies suggested by the task 1c, give relevance feedback (1=relevant, -1=irrelevant, 0=unlabeled). Based on this feedback, update the movie-genre matrix and calculate pagerank and update the top 5 similar movies to the user. To find the weight for genres after giving feedback, use the formula:

For positive feedback,

**wi = log( (ri + 0.5/ (R - ri + 1)) / ((ni - ri + 0.5) / (N – R – ni + ri + 1)))**

where, wi is the positive relevant weight of that genre after feedback,

ri is the count of relevant genre in relevant movies

ni is the count of relevant genre in total number of movies

R is the count of relevant movies

N is the count of total movies

For negative feedback, the formula is same but the variables become,

ri is the count of irrelevant genre in irrelevant movies

ni is the count of irrelevant genre in total number of movies

R is the count of irrelevant movies

After the feedback weight calculation, for each genre, slice the tensor corresponding to the specific genre, so you’ll get a matrix then update the values for each cell in the matrix which has a non-zero element as follows: For positive feedback weight, add the weight to the cell’s values and for negative feedback weight, subtract the weight.

With this updated tensor, now again decompose the tensor and follow the same steps as in task 1c to recommend movies to the user.

**Task 2d:**

we will use the movie-movie matrix as the transition matrix for random walk with restarts algorithm. The movies watched by the given user are considered as the seeds and a teleportation matrix for movies is create using the seeds. Weights are given to the seeds in the order of the time they were watched by the user and the year the movies were released.

In this task, for the 5 similar movies suggested by the task 1d, give relevance feedback (1=relevant, -1=irrelevant, 0=unlabeled). Based on this feedback, update the movie-genre matrix and calculate pagerank and update the top 5 similar movies to the user. To find the weight for genres after giving feedback, use the formula:

For positive feedback,

**wi = log( (ri + 0.5/ (R - ri + 1)) / ((ni - ri + 0.5) / (N – R – ni + ri + 1)))**

where, wi is the positive relevant weight of that genre after feedback,

ri is the count of relevant genre in relevant movies

ni is the count of relevant genre in total number of movies

R is the count of relevant movies

N is the count of total movies

For negative feedback, the formula is same but the variables become,

ri is the count of irrelevant genre in irrelevant movies

ni is the count of irrelevant genre in total number of movies

R is the count of irrelevant movies

After the feedback weight calculation, for each genre, update its column in the movie-genre matrix. For positive feedback weight, add the weight to the column’s values and for negative feedback weight, subtract the weight. With the updated movie-genre matrix, calculate the movie-movie matrix and do pagerank again as explained in task 1d.

**Task 2e:**

In this task, for the 5 similar movies suggested by the task 1e, give relevance feedback (1=relevant, -1=irrelevant, 0=unlabeled). Based on this feedback, update the movie-genre matrix and calculate pagerank and update the top 5 similar movies to the user. To find the weight for genres after giving feedback, use the formula:

For positive feedback,

**wi = log( (ri + 0.5/ (R - ri + 1)) / ((ni - ri + 0.5) / (N – R – ni + ri + 1)))**

where, wi is the positive relevant weight of that genre after feedback,

ri is the count of relevant genre in relevant movies

ni is the count of relevant genre in total number of movies

R is the count of relevant movies

N is the count of total movies

For negative feedback, the formula is same but the variables become,

ri is the count of irrelevant genre in irrelevant movies

ni is the count of irrelevant genre in total number of movies

R is the count of irrelevant movies

After the feedback weight calculation, for each genre, slice the tensor corresponding to the specific genre, so you’ll get a matrix then update the values for each cell in the matrix which has a non-zero element as follows: For positive feedback weight, add the weight to the cell’s values and for negative feedback weight, subtract the weight.

With this updated tensor, now again decompose the tensor and follow the same steps as in task 1e to recommend movies to the user.

**TASK 3:**

* We are creating a movie-tag matrix, which we will reduce to 500 dimensions using LDA.

Let M be that movie-tag matrix.

M` = LDA(M)

* First, we select Random Vectors using Normal Distribution (numpy library) for all the hash functions.
* We take each movie vector from M` and calculate the hash value using the formula:

hash (r, b) = Math.floor(r·x+b)/w;

Where r is the random projection vector selected for that hash function;

x is the movie vector, b is the random value selected in b -> [0, w);

w is the width of the bucket of the segment.

* We calculate the hash values of the same data point vector using all the hash functions of a layer. We take the concatenated values of the all the hash values and take that as a key and construct the index structure for that layer. By doing this for all the data points we have an index structure with keys as the hash codes and the values of the bucket would be the movies which are similar.
* We similarly calculate the hash values of all the data points for all the layers. If there are L layers then we construct ‘L’ index structures.
* When a query movie ID is given we first find the hash codes of that movie data point in all the layers. After finding the key of the given movie id in all the layers we look at each index structure for that layer and see what values does that key contain. We collect the obtained movie ID’s from all the index structures and form a movie list which are the similar movies of the given query movie id.
* We check for the number of movies obtained after collecting the similar movies, if it exceeds the number of similar movies that the user is expecting then we get the movies from the other buckets whose keys are similar to the one that is obtained for the query. This is done using Hamming Distance.
* After obtaining the similar movie list, we do the dot product of the query movie vector with the obtained similar movie vectors and retrieve the movies which are having the least similar values.

**TASK 4:**

We have created the locality sensitive hash index structure in task 3. So, for the nearest neighbor of a given input the system will suggest the recommendations. For each movie recommended we, take input whether the movie is relevant or not. If it is relevant then enter ‘y’ else ‘n’. The input movie would be represented in the reduced vector of 500 dimensions which we represent by Q.

Q = (q1, q2, q3, ...., q500)

Given the recommended movies we find the corresponding movie vectors (D) from the object feature matrix which represents each movie in reduced 500 dimensions. Then according to the following formula, we update the original query Q. [1]

The first part of this equation is the relevant movies and the second part is for irrelevant movies. So now the query will be updated so that it is closer to the relevant movies in the vector space.

Next, we will pass this query Q` to the Locality Sensitive Hash function, where it will be hashed to a bucket and we will recommend the movies which are in this bucket and are closer to our new query.

**TASK 5:**

**Task 5a:**

In this task, the input is movies and a label associated with each set. The object-feature relation is movie-genre and create the movie genre matrix as mention in task 1d. For the movies in the data set, associate the movies to one of the labels based on the k Nearest Neighbor approach. In k Nearest Neighbor classifier, **for each movie in data set, calculate the distance with the movies in the input**. **Sort them based on the distances and take the first k movies from the input. Get the labels associated with the movie. The label which has the maximum count will be associated to the movie.** For the output, display the label and the movies associated to it. Cosine similarity is used to find the distance between the movie in data set and the input movies.

**Task 5b:**

In this task, the input is movies and a label associated with each set. For the movies in the data set, associate the movies to one of the labels based on the decision tree classifier. Using the labels and the movies from the input, **create a decision tree using movie genre relationship**. For each level, find a genre for the movies in the input that can achieve maximum separation. To **find the best genre, take a genre that has equal number of movies with and without the genre**. This will help in splitting the movies set into halves approximately. Calculate the entropy value of the genres in finding the best fit. After finding the genre that has best split, **the movies that have that genre will be on one sub tree and the movies without the genre will be on another sub tree**. Do this until the movies can’t be split further.

Once the decision tree is created, take the movies in the data set and classify the movie using the created decision tree, **follow the edges of the tree based on the genre value of the tree and the genre in that node till the leaf node is reached**. For the output, display the label and the movies associated with it.

**Task 5c:**

In this task, based on the given movies and labels, classify all the movies in the data set to the input labels. SVM classifier is used to classify the movies. Using the movie genre matrix with the tf-idf values used in task 1a, get the movie-genre vector for the input movies. This will be the input to create a model. In creating the model, first create an empty co-efficient matrix for number of labels by number of number of input movies. Calculate the normalized matrix of the input movie-genre matrix over the column. For the input movies, calculate the gradient and find the optimum violations of it. Calculate the violation ratio for all the movies and do the iterations till the violation comes under 0.05. Update the co-efficient matrix with dot product of gradient and input movie samples. Once the model is created, for all the movies in the data set, predict the label. To predict, find the dot product of the movie-genre vector of the movie with the model created and find the column of the output with highest value and classify the movie to that label. For output, display the labels and the movies associated with it.

**SYSTEM REQUIREMENTS/INSTALLATION AND EXECUTION INSTRUCTIONS:**

SYSTEM REQUIREMENTS:

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

INSTALLATION STEPS:

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

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

* pip install scikit-tensor

To install in development mode

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

EXECUTION STEPS:

* TASK 1:
  + 1a: ./task1a 109 SVD/PCA
  + 1b: ./task1a 109 LDA
  + 1c:
    - Command: python task1c.py userid
  + 1d:
    - Command: python task1d.py
    - Enter user id: 109
  + 1e:
    - Command: python task1e.py userid
* TASK 2:
  + 2a: ./task2a 109 SVD/PCA
    - After that enter y, n or d for feedback
  + 2b: ./task2b 109 LDA
    - After that enter y, n or d for feedback
  + 2c:
    - Command: python task2c.py userid
    - After the output, enter 1, -1 or 0 for the movies as feedback
  + 2d:
    - Command: python task2d.py
    - Enter user id: 109
    - After the output, enter 1, -1 or 0 as feedback for the movies.
  + 2e:
    - Command: python task2e.py userid
    - After the output, enter 1, -1 or 0 for the movies as feedback
* TASK 3:
  + Command: python lshIndexing.py
    - NumberOfHashesPerLayer: 12
    - NumberOfLayers: 6
    - movieID: 1
    - NumberOfMoviesToRecommend: 10
* TASK 4:
  + Command: python lshIndexing.py
    - NumberOfHashesPerLayer: 12
    - NumberOfLayers: 6
    - movieID: 1
    - NumberOfMoviesToRecommend: 10
* TASK 5:
  + 5a:
    - Command: python task5a.py
    - Enter number of labels: 2
    - Enter label: Comedy
    - Enter movies for the label: 5259, 4026
    - Enter label: Drama
    - Enter movies for the label: 10120, 5981
    - Enter k: 2
  + 5b:
    - Command: python task5b.py
    - Enter number of labels: 2
    - Enter label: Comedy
    - Enter movies for the label: 5259, 4026
    - Enter label: Drama
    - Enter movies for the label: 10120, 5981
  + 5c:
    - ./task5-c svm
    - [‘5259’,’10120’,’4026’,’10011’,’4025’,’4020’]
    - Enter number of labels: 3
    - Enter label: t1
    - Enter movie id related to table: 5259
    - Enter label: t2
    - Enter movie id related to table: 10120
    - Enter label: t3
    - Enter movie id related to table: 4026

**RELATED WORK:**

* [1] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. Journal of the American Society for Information Science. 41, pp. 288-297, 1990 is used in calculating the feedback and implement probabilistic relevance feedback system.
* [2]” Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions” (by Alexandr Andoni and Piotr Indyk). Communications of the ACM, vol. 51, no. 1, 2008, pp. 117-122 is used in implementing the Locality Sensitive Hashing tool where each movie in the system is mapped on a 500-dimensional latent space.
* [3]R. A. Harshman, Foundations of the PARAFAC procedure: Models and conditions for an “explanatory” multi-modal factor analysis, UCLA Working Papers in Phonetics, 16 (1970): We referred this paper to know about the CP decomposition.
* [4]J. D. Carroll and J. J. Chang, Analysis of individual differences in multidimensional  scaling via an N-way generalization of “Eckart-Young” decomposition, Psychometrika: This paper describes how to implement CP decomposition.
* [5]J.B. Kruskal, *Rank, decomposition, and uniqueness for 3-way and N-way arrays*, in *Multiway Data Analysis*, R. Coppi and S. Bolasco (eds.), Elsevier, Amsterdam, 1989, 7–18. : This paper gives an idea about how to represent tensors.

**CONCLUSION:**

We have implemented a recommender system by considering the user’s watched movie data and recommend the user 5 more movies by using different methods like Singular Value Decomposition, Principle Component Analysis, Latent Dirichlet Allocation, CP decomposition, Personalized Page Ranking. Now the user will be giving a feedback to the recommended movies and using probabilistic relevance feedback system we will be improving the matches. We have implemented Locality Sensitive Hashing tool, where each movie is mapped onto 500-dimensional latent space. Again, user will be providing feedback to the results and based on the feedback we performed r-nearest neighbor based relevance feedback and improved the matches. Finally we classified the movies based on the algorithms like r-nearest neighbor based classification algorithm, decision tree based classification algorithm and n-ary SVM based classification algorithm.

**LIST OF PUBLICATIONS RELAVENT TO WORK:**

* [1] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. Journal of the American Society for Information Science. 41, pp. 288-297, 1990.
* [2]” Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions” (by Alexandr Andoni and Piotr Indyk). Communications of the ACM, vol. 51, no. 1, 2008, pp. 117-122.
* [3] R. A. Harshman, Foundations of the PARAFAC procedure: Models and conditions for an “explanatory” multi-modal factor analysis, UCLA Working Papers in Phonetics, 16 (1970), pp. 1–84. Available at http://publish.uwo.ca/ ̃harshman/wpppfac0.pdf.
* [4] J. D. Carroll and J. J. Chang, Analysis of individual differences in multidimensional  scaling via an N-way generalization of “Eckart-Young” decomposition, Psychometrika, 35  (1970), pp. 283–319.
* [5] J.B. Kruskal, Rank, decomposition, and uniqueness for 3-way and N-way arrays, in Multiway Data Analysis, R. Coppi and S. Bolasco (eds.), Elsevier, Amsterdam, 1989, 7–18.
* [6] B.W. Bader and T.G. Kolda, A Preliminary Report on the Development of MATLAB Tensor Classes for Fast Algorithm Prototyping, Technical Report SAND2004–3487, Sandia National Laboratories, Livermore, CA, July 2004.

**APPENDIX:**

* Abhishek Thorat, Shubam Gondane and Sachin Sundar implemented the Task 1 and 2.
* Krishna Chaitanya Bejjipuram, Shiva Tejaswi Rampally and Raviteja Upmaka implemented Task 3 and 4.
* Task 5 is implemented by combined ideas of all the team members.