Project Report: Multimedia and Web Databases

Phase III

Instructor
K. Selçuk Candan

Submitted by
Giriraj
Sharma Vartika
Chikkala Hasitha
Dhumantarao Sai Koushik
Kanteti Satya Srinija

ARIZONA STATE UNIVERSITY

29 November, 2017

Contents

Abstract

Fast retrieval methods are paramount for many large-scale and data-driven applications. Recent work has explored ways to embed high-dimensional multimedia features and complex distance functions into a low-dimensional space where items can be efficiently searched. For the multimedia data management using content-based and object-based query processing, and how media objects affect users, we require better understanding of the underlying perceptive and cognitive processes in media processing. In this project, we will experiment with vector models and graph models to implement techniques for media transformation, so that we can retrieve and de-correlate our input data. Some techniques implemented in this project phase are Principal Component Analysis, Cosine Similarity, Singular Value Decomposition, Tensor Decomposition, Personalized PageRank, Relevance Feedback, Locality Sensitive Hashing, r-nearest neighbor based classification algorithm, decision tree based classification algorithm, and N-ary Support Vector Machine based classification algorithm.

Keywords Locality Sensitive Hashing, PageRank, Principal Component Analysis, Singular Value Decomposition, Tensor decomposition, Support Vector Machine, Decision Trees and Nearest Neighbor Search.

1 Introduction

The final phase of the project implements various advanced techniques for recommending movies to the user based on their movie-watching patterns. In Phase 1 of the project we created the tag vectors for actors, genres, and users, using TF and TF-IDF models. Phase 2 of the project involved extracting top latent semantics from space of tags and actors, by making use of decomposition techniques like Principal Component Analysis (PCA), Singular-Value Decomposition (SVD), Latent Dirichlet allocation (LDA) and tensor decomposition. We also implemented the extraction of top latent semantics on object-object similarity matrices using SVD and tensor decomposition. Principal component analysis is one method of reducing the number of dimensions in the raw data. [1] In linear algebra, the singular-value decomposition (SVD) is a factorization of a real or complex matrix. It is the generalization of the Eigen Decomposition of a positive semi-definite normal matrix (for example, a symmetric matrix with positive eigenvalues) to any m x n matrix via an extension of the polar decomposition. It has many useful applications in signal processing and statistics. Latent Dirichlet allocation (LDA) is a topic model that generates topics based on word frequency from a set of documents. LDA is particularly useful for finding reasonably accurate mixtures of topics within a given document set. In multi-linear algebra, the tensor rank decomposition or Canonical Polyadic decomposition (CPD) may be regarded as a generalization of the matrix singular value decomposition (SVD) to tensors. Along with it, we incorporated Personalized PageRank algorithm to find the most related actors in the space.

These results from the previous two phases are used in Phase 3 of the project for the purpose of recommending movies to users. Phase 3 also involves implementing Relevance Feedback from users to get improve the accuracy of the recommendations. We also employ Locality Sensitive Hashing to search for similar movie search and extend this to improve the search using user feedback.

Apart from recommending movies to users, we implemented classification algorithms like the r-nearest neighbor, decision tree, and N-ary Support Vector Machine. K nearest neighbors is an elementary algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., Euclidean Distance Function). A decision tree classifier has an easy form which can be compactly stored and that efficiently classifies new data. Decision tree classifier can perform automatic feature selection and complexity reduction, while the tree structure gives easily understandable and interpret-able information regarding the predictive or generalization ability of the data.

1.1 Goal description (problem specification)

Phase 3 of the MWD project works on the various vector and tensor decomposition methods, relevance feedback, multidimensional index structures, nearest neighbor search and several classification algorithms. We use the Term Frequency-Inverse Document Frequency (TF-IDF) weighted tag vector for different terms in a document (from the phase one of the project) from Phase 1. Tags are classified and information is retrieved according to different criteria such as all the movies an actor has played in, all movies of a given genre, all the movies watched by a particular user and the difference between two genres. Hence, we do the classification of the documents using a vector-based model for document representation in Phase 2.

Phase 3 goal is to implement vector decomposition techniques like SVD, PCA, LDA, tensor decomposition, and an integrated measure of all these to recommend movies to a given user based on his movie-watching patterns. It also involves taking feedback from the user and use this to revise the recommendations. Another goal is to put into effect Locality Sensitive Hashing tool and perform a similar movie search. This search is refined using user's relevance feedback and thus obtain an improved set of nearest neighbors. The final goal is to apply classification techniques to label all the remaining movies, given a set of labeled movies, i.e, put them into classes.

For Phase 3 ofour project, we are using the same $IMDB$ and $MovieLens$ Data including ml-movies(movieId; movieName; genres), mltags(userId; movieId; tagid; timestamp), mlratings(movieId; userId; imdbId; rating; timestamp), genome-tags(tagId; tag), movie-actor (movieId; actorId; actorMovieRank), imdb-actor-info (actorId; name; gender), mlusers(userId) in the form of csv files.

1.2 Terminology

Objects A database object is any defined object in a database that is used to store or reference data. Some examples of database objects include actors, movies etc. These are the objects which are described using features like genres, tags, timestamps etc.

Features Features are attributes related to the objects which describe the objects of a database. Some examples of features include tags, genres, timestamps etc.

Latent Semantics/ Topics Latent semantics are the hidden features or unobserved features in a database unlike the observed features like tags, genres etc. These can be usually by applying some kind of decomposition on the object-feature or object-object or feature-feature data matrices. They give information about the relationships between the objects.

Tensors A data tensor is a 3-dimensional representation of a database. Tensor is mainly useful when the database has a temporal dimension that cannot be captured in a single snapshot. Similar to vector decomposition, tensor can be decomposed to obtain factor matrices.

TF-IDF Term Frequency-Inverse Document Frequency is a text mining technique used to categorize the given set of documents. TF-IDF classifies documents into categories inside the documents. [2] This gives us knowledge of the reviews.

K-means Clustering K means Clustering classifies a set of observations into k clusters using the k-means algorithm. The algorithm attempts to minimize the **Euclidean distance** between observations and centroids. Several initialization methods are included.

Relevance Feedback It is an automatic process designed to produce improved query formulations following an initial retrieval operation. The user gives marks a movie as relevant or irrelevant and based on this, the improved results can be produced for recommending to the user.

Multidimensional index structures The data structures which are used to store multidimensional data. If the given data set has k features, it can be represented using a k-dimensional space.

Locality Sensitive Hashing It reduces the dimensionality of high-dimensional data. LSH hashes input items so that similar items map to the same "buckets" with high probability. LSH differs from conventional hash functions because it aims to maximize the probability of a "collision" for similar items.Locality-sensitive hashing has much in common with data clustering and nearest neighbor search.

Nearest Neighbor Search This involves searching for a given number (k) of nearest neighbors to a query.

R-nearest neighbor classification A type of classification method where the query point is classified into the class in which its nearest neighbor is present.

Decision-tree based classification A type of classification method where a decision tree is used to classify the query point. The nodes of the tree are used to represent the classes and the branches are used to verify whether a feature can classify the query into one of the child nodes or not. The root node will always have the entire data set.

N-ary SVM based classification method For each query point, the best class suitable for it is selected and we make sure that it is as far as possible from the separating hyper plane.

1.3 Assumptions

For our implementation, we will take the following assumptions:
1) For SVD Implementation, the number of movies in the given genre must be greater than the number of Latent Semantics needed.
2) If a user has rated a movie (mlratings.csv), then the user has has also watched the movie.
3) If no user has rated the movie, then its rating is 0.
4) Ratings are taken as integers 1,2,3,4,5.
5) For Personalized PageRank implementation,

$$T = aM + (1 - a)\left[\frac{1}{N}\right]$$

Here, α value is 0.15 because we wish to give more weight to the Seed Vectors.
6) The value of parameter α in LDA implementation is '1' and it follows a symmetric distribution. By assuming so, the peak of Dirichlet distribution will be in the center.
7) We do not consider the similarity factor for the term if it is not differentiating:

a) occurs same number of times in relevant and irrelevant objects.
b) does not occur in any of the objects.
8) If objects are not discriminating, proper weights to the features/semantics cannot be given.

2 Description of the proposed solution/implementation

2.1 First Task: Movie Recommendation

The first task of phase 3 deals with the implementation of a program which given all the information about the sequence of movies a given user has watched, recommends the user five more movies to watch. To obtain the top 5 recommended movies, we consider the $time - stamp$ of the movies the user has watched and the recommendations should reflect the most recently watched movies. The first four parts of the first task deals with various methods to implement the task namely: SVD, PCA, LDA, Tensor Decomposition and Personalized PageRank, and the fifth part of the first task combines all these measures to give the movie recommendation to the given user.

2.1.1 Task 1a

For the first part, we use Singular Value Decomposition (SVD) and Principal Component Analysis (PCA) for getting movie recommendations.

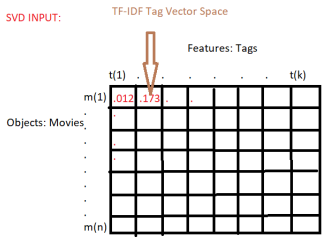

Figure 1: Singular Value Decomposition Input

As already performed in Phase 2, we decompose the data matrix $(movie - tag)$ into U, S and V matrices to get the latent semantics. Here, we use the movie-latent semantic matrix and use it along with the query vector to obtain a list of movies related to each user's watching patterns. A *queryvector* is formed by taking the movies a user has watched and giving higher weights to the most recent movie in the vector. The procedure is to take the $movie - semantic$ matrix, and find the similarity of each movie with the query vector. This gives the measure of how well each movie is related to the user's movies. Based on this, the 5 movies with the most similarity are recommended to the user.

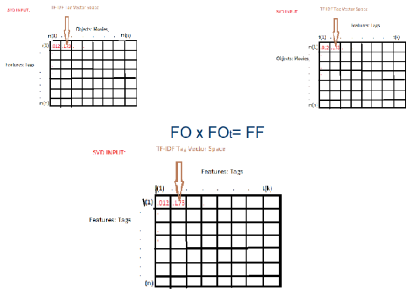

Figure 2: Principal Component Analysis Input

The '$movieFacts'$ is the movie-semantic matrix and '$queryVect'$ is the query vector taken from the user. 'res' contains the Dot Product similarity between them to get all the similarities. The top 5 movies from the sorted 'res' are produced as recommendations for the user.

2.1.2 Task 1b

We use Latent Dirichlet Analysis (LDA) for getting the movie recommendations from the user. From Phase 2, we decompose the data matrix $(movie - actor)$ to get the movie - latent semantics. Here, we use the $movie - semantic$ matrix and use it along with the *queryvector* to obtain a list of movies related to each user's watching patterns (same as Task 1a). A query vector is formed by taking the movies a user has watched and giving higher weights to the most recent movie in the vector. The procedure is to take the movie-semantic matrix, and find the similarity of each movie with the query vector. This gives the measure of how well each movie is related to the user's movies. Based on this, the 5 movies with the most similarity are recommended to the user.

The '$movieFacts'$ matrix is the movie-semantic matrix and '$queryVect'$ is the query vector taken from

the user. '*res*' contains the Dot Product similarity between them to get all the similarities. The top 5 movies from the sorted '*res*' are produced as recommendations for the user. The '*lda*' module from phase 2 is imported to '*ldaRecommendations.py*' to perform LDA and get the topics.

2.1.3 Task 1c

We use tensor decomposition for getting the recommendations. The $actor - movie - year$ tensor is taken from Phase 2 and it is decomposed to get $actor - semantic$, $movie - semantic$ and $year - semantic$ matrices. In the tensor decomposition implementation of the task, the $movie - semantic$ matrix is used along with the query vector to obtain a list of movies related to each user's watching patterns (same as Task 1a). A query vector is formed by taking the movies a user has watched and giving higher weights to the most recently watched movies. The procedure is to take the movie-semantic matrix, and find the similarity of each movie with query vector. This gives the measure of how well each movie is related to the user's movies. Based on this, the top 5 movies with the most similarity are recommended to the user.

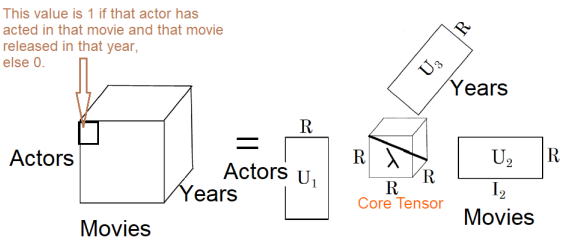

Figure 3: Tensor Input

The '*movieFacts*' matrix is the movie-semantic matrix and '*queryVect*' is the query vector taken from the user. '*res*' contains the Dot Product similarity between them to get all the similarities. The top 5 movies from the sorted '*res*' are produced as recommendations for the user.

The '*tensorDecomp*' module is imported to the '*task*$1c_3$*.py*' to implement tensor decomposition and get the movie-semantic matrix (*movieFacts*).

2.1.4 Task 1d

In the fourth part of the task, we use Personalized Pagerank for getting movie recommendations. We performed PPR algorithm on '$movie - movie$' similarity matrix. The Random Walk algorithm is implemented as Personalized PageRank where we take the tf-idf of movies in the tag-space:

It is defined by the formula:

$$P' = \alpha MP + (1 - \alpha)S$$

where, it is important to note that we take the value of α as 0.15 to give **more weight** to the seed vectors.

Instead of a query vector (like in the previous three parts), we consider *seed* movies. Based on this, the 5 movies with the most similarity with the seeds are recommended to the user. The '*rankedRes*' contains the recommended movies. This is given as input for the Task 2 to get revised recommendations by taking relevance feedback also.

2.1.5 Task 1e

The final part of the first task, combines all the above measures to give a more significant movie recommendation to the user. We use the mode function from the *scipy.stats* library. *Mode* function of Scipy library returns an array of the modal (most common) value in the passed array. We use this function to find the most common top ranks from the list of recommended movies we received from the previous four parts of the first task. The final list will result in the combined recommendation from all four tasks.

2.2 Second Task: Relevance Feedback Task

The goal of the second task is to produce improved results for the obtained results in the first task. We use a probabilistic relevance feedback mechanism to improve

the matches from the first task. [3]. In probabilistic relevance feedback, an optimal retrieval rule is used to rank the documents (movies) in decreasing order.

$$log\frac{(P_r(x|Relevant)}{P_r(x|Irrelevant)}$$

where, $P_r(x|Relevant)$ represents the probabilities that a relevant item has a vector representation x and $P_r(x|Irrelevant)$ represents the probability that a non-relevant item has a vector representation x.

The rank for each movie is calculated using the equation:

$$sim(D,Q) = \sum d_i log\frac{p_i(1-u_i)}{u_i(1-p_i)}$$

where d_i represents the weight of the term i. This gives us the ranks of all the movies in the $movie - semantics$ matrix. These 'sim' values cannot be computed without knowing the values of p_i and u_i for all the movies. Despite the suggestion of several methods to estimate these values, we use the equations mentioned below to compute them. For the initial search when the relevance of the movies is not known, we may assume that the p_i and u_i values are 0.5. But later, the user feedback of relevant and irrelevant movies is taken and then this feedback is applied to the movies to get revised rank for each movie. In the above formula, p_i and u_i values are calculated as follows:

$$p_i = (r_i + n_i/N)/(R+1)$$
$$u_i = (n_i - r_i + n_i/N)/(N-R+1)$$

where, r_i = the total number of relevant retrieved items that include term i,

n_i =the total number of retrieved items with term i,

R = total number of relevant retrieved items

If the latent semantic exists (i.e it is not equal to zero) for a particular relevant movie, then its r value is incremented. Similarly, if the latent semantic value exists for any of the movies, its n value is incremented. R is the total number of relevant movies, i.e, the length of the relevant vector list. N is the total number of movies. These values are substituted in the sim formula to get the revised rank for each movie. Then, the movies are sorted in decreasing order of their new ranks (weights). The top 5 movies will be are produced as output and they represent the 5 recommendations for the user. This approach is employed for all the sub-tasks of the first task - PCA, SVD (Task 1a), LDA (Task 1b), and tensor decomposition (Task 1c) and the combined measure (Task 1e).

For the Personalized PageRank task of the first task (Task 1d), relevance feedback is applied differently. The $getRevisedRanksPPR()$ function is used to get the revised movies separately from the remaining tasks. The seed movies are also passed to the feedback function in addition to the relevant and irrelevant movies. We do not consider the '$queryVect'$ in obtaining the '$rankedRes'$. We build an object-object similarity matrix of the '$semMatrix'$ and again perform PPR on this to get newly revised rank for each movie. From this, the top 5 movies are recommended to the user. If no feedback is provided by the user, the same set of recommendations from the first task is given to the user.

2.3 Third Task: Multidimensional index structures and nearest neighbor search task

The third task of Phase 3 deals with the Multidimensional index structures and nearest neighbor search task. We implement a program which maps each movie in the system into a 500 dimensional latent space. And further, implement a Locality Sensitive Hashing (LSH) tool, which takes as input (a) the number of layers, L, (b) the number of hashes per layer, k, and (c) a set of movie vectors as input and creates an in-memory index structure containing the given set of vectors [4] We then implement similar movie search using this index structure: for a given movie and r, outputs the r most similar movies (also outputs the number of unique and overall number of movies considered). The result interface is made such that it allows the movie to provide positive and/or negative feedback for the returned movie returned by the system to enable Task 4.

2.3.1 Task 3a

The goal is to map every movie in the given data into a 500-dimensional latent space. 500-dimensional data space is obtained by applying SVD on $movie - tags$ matrix.

From this, we extract the top 500 latent semantics from the S matrix.

2.3.2 Task 3b

Locality Sensitive Hashing (LSH). In LSH, we make use of a hashing scheme to that maps closely neighboring items to the same class. Locality Sensitive Hash are deterministic functions that tend to map nearby points to the same or nearby values. Let $sim()$ be a similarity function. A locality sensitive hash corresponding to $sim()$ is a function, $h()$, such that:

$$prob[h(o1) = h(o2)] = sim(o1, o2)$$

The challenge here is to find the appropriate $h()$ for a given $sim()$. We implement L2-Norm or Euclidean Norm for the implementation of LSH. The L2-Norm of a vector $x = (x_1 + x_2 + ...x_n)$ is given by:

$$|x| = \sqrt{(x_1)^2 + (x_2)^2 + ...(x_n)^2}$$

The input file for our LSH implementation is *csv* file with movie ids (TF-IDF Values).

2.3.3 Task 3c

Using the index structure designed in Task 3b, we perform a similar movie search. The input is taken as a movie and r (the number of movies) and output of r similar movies to the given movie is produced. This is extended to Task 4 and relevance from the user is taken in order to improve the nearest neighbors of the movie.

2.4 Fourth Task: NN-based relevance feedback

NN-based relevance feedback. We 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. The result interface from Task 3 should allow the movie to provide positive and/or negative feedback for the returned movie returned by the system to enable this task.

In this task, we will shift the query vector, Q, in direction of relevant movies and away from the irrelevant movies to form a new query vector, Q', which is closer to the relevant movies and farther from the irrelevant movies. This is in accordance with the problem statement. This way we will get the required revised recommendation using the given formula:

$$Q' = Q + \frac{\sum R - \sum I}{T}$$

where, R = Relevant Movie Vector,

I = Irrelevant Movie Vector, and

N = Total Number of Movies (Relevant and Irrelevant)

In the module *relFeedback.py*, we apply the above formula to obtain a new query vector. We now use LSH of task 3 to find the movies which are relevant to the user (based on user feedback). In this manner, we get r closest movies to the query.

2.5 Fifth Task: Movie classification

We implement 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 the rest of the movies in the database.

2.5.1 Task 5a: Nearest Neighbor based Classification

The first part of the Task 5 deals with the movie classification using k- nearest neighbor. The implementation of our k-Nearest Neighbor algorithm searches through the training data set for the k-most similar instances. The prediction attribute of the most similar instances is summarized and returned as the prediction for the unseen instance. The similarity measure is dependent on the type of data. We use Euclidean distance for the implementation as given:

$$EuclideanDistance = \sqrt{\sum (x_i - y_i)^2}$$

It should also be noted that euclidean distance measures are only valid for continuous variables. In the case of classification, the most prevalent class may be returned.

Our input to the classifier is a set of movie tag vectors (based on TF-IDF values) and corresponding labels (given as test data). Now we need to split the data into a training data set that k-NN can use to make predictions and a test data set that we can use to evaluate the accuracy of the model.

2.5.2 Task 5b: Decision Tree based Classification

The second part of this task deals with the movie classification based on the decision trees. A decision tree recursively partition a data set into smaller subdivisions on the basis of tests applied to one or more features at each node of the tree. Decision trees is great for data with a simple mix of numerical and categorical as in our case. Our input here again is a set of movie tag vectors (based on TF-IDF values) and corresponding labels (given as test data).

For our implementation, we first construct a root node by considering all the observations in the given movie data set and determine which variable or feature would subset the data the most. We check the outcome of spliting on each variable. When constructing our root node we should chose a variable that creates two sets with the least possible amount of mixing.

2.5.3 Task 5c: SVM based Classification

In the final part of the fifth task, we deal with Support Vector Machine based classification to classify the movies according to given labels (we give two labels). The objective of the Support Vector Machine is to find the best splitting boundary between the movies of the two labels. In the 2-D space, you can think of this like the best fit line that divides our movie data set.

With a Support Vector Machine, we're dealing in vector space, thus the separating line is actually a separating hyper plane. The best separating hyper plane is defined as the hyper plane that contains the maximum margin between the support vectors. A hyper plane is also known as a decision boundary. In the Task 5a, we classify a new data point based on what it is closest to or the proximity. The main problem with that implementation is that, per data point, you have to compare it to every single data point to get the distances, thus the algorithm does not scale properly, even though it is quite accurate. Support Vector Machine generates the $bestfit$ line that properly classifies our movies according to the given labels. Thus, this algorithm scales well, unlike the K-NN classifier in Task 5a. Our input here like Task 5a and 5b is a set of movie tag vectors in movie space (based on TF-IDF values) and corresponding labels (given as test data).

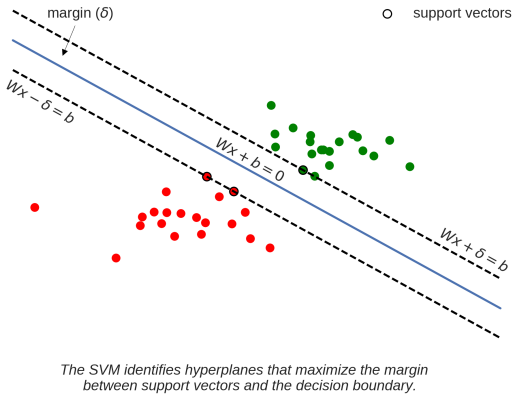

Figure 4: SVM maximizes the margin between support vectors and decision boundary.

For our implementation, we first define a function to train the given training data according to the labels and then give the test movie data to classify them.

3 Interface specifications (with focus on the goal and problem specification)

We used the following libraries and functions for the implementation of our program:

1) **TensorLy** TensorLy is a fast and simple Python library for tensor learning. It builds on top

of NumPy, SciPy and MXNet and allows for fast and straightforward tensor decomposition, tensor learning and tensor algebra. To install it, we can simply run: $pipinstall - Utensorly$. Alternatively, we can pip install it from the git repository: $pipinstallgit + https : //github.com/tensorly/tensorly$. [5]

2) **SciPy** We used the Scipy library in several tasks for the usage of kmeans, Euclidean distance, cosine distance ($fromscipy.spatialimportdistance$) [6]

3) **NumPy** [7] Functions from the NumPy library have been incorporated in the tasks involving SVD and similarity matrix

4) **Scikit-Learn** It is a simple and efficient tools for data mining and data analysis which is accessible to everybody, and reusable in various contexts. It is built on NumPy, SciPy, and matplotlib Open source, commercially usable - BSD license. We can use functions like PCA from this library. [8]

We use the following Software Distribution:
Python 3.3+ on Ubuntu 14.04 LTS)

Further, input and output for each tasks have been mentioned in the output folder attached with the Code.

4 System Requirements/ Installation and Execution Instructions

The project is implemented in Python (Linux Environment: Ubuntu 14.04) where we have made use of various libraries such as sciPy, csv, sqlite3 (for querying data from the csv files), math, numPy and many more. We can simply install the given libraries in Linux Terminal (alternatively, we can install the Anaconda python distribution package on Windows 10).

The first step is to create a Connection object that represents the database using the sqlite3 library. We then create a file where the data will be stored; in our case it is stored in *mwdproj.db*. Now, we create a cursor object, which allows us to interact with the database and add records. Here we use SQL syntax to create tables with text different fields.

We then run the python file $task1_3.py$, $task2_3.py$, $task3_3.py$, $task4_3.py$ and $task5_3.py$, etc. which contains implementation of all the given Tasks.

5 Related Work

In the recent years, there has been significant work done in the field of information retrieval and development of randomized methods for low-rank approximation. These methods target principal component analysis (PCA) and the calculation of truncated singular value decomposition (SVD). The paper by A. Szlam et al. [9] presents an essentially black-box, fool-proof implementation for Mathworks' MATLAB, a popular software platform for numerical computation. Paper by Michael E. Wall et al. [10] describes SVD methods for visualization of gene expression data, representation of the data using a smaller number of variables, and detection of patterns in noisy gene expression data. Ranking methods like PageRank assess the importance of Web pages based on the current state of the rapidly evolving Web graph. The dynamics of the resulting importance scores, however, have not been considered yet, although they provide the key to an understanding of the Zeitgeist on the Web. The paper by Klaus Berberich et al. [11] proposes the BuzzRank method that quantifies trends in time series of importance scores and is based on a relevant growth model of importance scores. The implementation for Personalized PageRank has extended beyond search on social networks. Personalized PageRank has found use in many other domains, including friend recommendation on Facebook [12], who to follow on Twitter [13], graph partitioning [14], community detection [15], and other applications [16]. On the web graph there is significant work on using Personalized PageRank to rank web pages (e.g. [17] [18]). PPR is also useful on heterogeneous graphs; for example it can be used to rank items in a bipartite user-item graph, in which there is an edge from a user to an item if the user has liked that item. Random walks on a user-item graph have proven useful for YouTube when recommending videos [19].

Randomized approximate similarity search algorithms have been designed to preserve query time guarantees, even for high dimensional inputs. The recent work in Locality-sensitive hashing [20] and [21] offers sub linear time search by hashing highly similar examples together

in a hash table. Prior to our work, LSH functions that accommodate Hamming distance [22], inner products [21], 'p norms [23], normalized partial matching [24], normalized set intersection [25], learned Mahalanobis metrics [26], and particular kernel functions such as the Gaussian kernel [27], [28] have all been developed. Vision researchers have shown the effectiveness of this class of methods for various image search applications, including shape matching, pose inference and bag-of-words indexing [29], [30], [24], [26], [25]. However, thus far arbitrary kernel functions remain off limits for LSH.

The recent work in nearest neighbor is multi-label learning. In this approach, each instance in the training set is associated with a set of labels, and the task is to output a label set whose size is unknown a priori for each unseen instance. In the paper [31], a multi-label lazy learning approach named ML-kNN is presented, which is derived from the traditional k-nearest neighbor (kNN) algorithm. For each new instance, its k-nearest neighbors are firstly identified. After that, according to the label sets of these neighboring instances, maximum a posteriori (MAP) principle is utilized to determine the label set for the new instance. Experiments on a real-world multi-label bioinformatic data show that ML-kNN is highly comparable to existing multi-label learning algorithms.

Further, the recent paper by [32], shows an interesting study to map fringe and basin mangrove forests at the species level using decision tree and support vector machine. The incorporation of spatial information either in the form of OBIA or pixel-based image texture (e.g., grey-level co-occurrence matrix or lacunarity) improves the classification accuracy. Spatial information seeks to extract repeated patterns in canopy structure that can be indirectly related to species. K-NN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.

6 Conclusion

In the MWD project, we worked on vector models and graph models to retrieve data from the high dimensional multimedia and web databases. In the first phase of project, we successfully implemented the task of taking all the movies an actor played and creates and stores a weighted tag vector for each actor using (time-weighted) TF as well as TF-IDF models. We also implemented a program which considers all movies of a given genre to create a combined tag vector for the genre. Moving on the second phase, we worked on the audacious task of implementing various techniques for reducing the number of dimensions in the raw data. We successfully implemented PCA, SVD and LDA for the tasks of the retrieving information given a certain Object such as genre, actor, movie, etc from various Object-Object, Object-Feature and Feature-Feature matrices. We also implement CP Tensor decomposition to calculate top latent semantics. Using PageRank, we were able to order search results so that more important and central movies are given preference. Finally moving on to our final and third Phase, we worked on movie recommendation using all the tasks done in Phase 2 and use normalized average to get the combined measure for recommending a movie to the user. Our results from Task 1 are giving proper movie recommendations and even taking user feedback (relevant and irrelevant movies) for improving the results we get. Further, in Task 2, we successfully use probabilistic relevance feedback to improve the results in Task 1. Our results in Task 2 shows significant improvement in movie recommendations from Task 1. In Task 3, we successfully implement LSH tool for Multidimensional index structures and nearest neighbor search task in High Dimensions for outputting the most similar movies given an input movie. The Task 4 implements a r-nearest neighbor based relevance feedback algorithm to improve the nearest neighbor matches. Our code outputs the revisions it suggests in the relative significance of different parts of the query. The final task of the project Phase 3, deals with movie classification. We test the three implemented classification algorithms by associating two labels to a given data set and then classify the test movie data in the two different classes of labels. From these three phases of this project we are successfully able to learn and implement various vector and graph models to understand and retrieve data from high dimensional multimedia and web databases. Although, some challenges remain when it comes while dealing with very large size input files and there is much scope of optimization still left but for most parts, we were successfully able to deal with such challenges of optimization.

Appendices

A Project Team Members

A.1 Giriraj, Arizona State University

Giriraj did the implementation of the creation of Tensor and doing CP decomposition of the Tensor. He also worked on the implementation of the Task 1c and Task 3 along with Hashita. He was also one of the major technical support of the group.

A.2 Sharma Vartika, Arizona State University

Vartika was responsible for the implementation of Task 5 including K-Nearest Neighbor, Support Vector Machine and Decision Trees. She also implemented Task 1e of the project. She also worked majorly on the report for the project.

A.3 Chikkala Hasitha, Arizona State University

Hasitha worked on the SVD and PageRank implementation in the project (Task 1a, Task 1d). He also worked on the implementation of the Task 3 along with Giriraj. She was also one of the major technical support in the group.

A.4 Dhumantarao Sai Koushik, Arizona State University

Koushik worked on the LDA implementation in Task 1b. He also worked Task 4, Relevance Feedback. He also made the Output Folder and Readme file for the project.

A.5 Kanteti Satya Srinija, Arizona State University

Srinija worked on the similarity matrix implementation in the various tasks of the project.She also worked on Task 2 of the project, that is Relevance Feedback Task. She also worked on making project report. She also worked on making project report.

References

[1] K. Selçuk Candan and Maria Luisa Sapino. *Data Management for Multimedia Retrieval.* Cambridge University Press, 2010.

[2] Buckley C. Salton G. Term weighting approaches in automatic text retrieval. information processing and management, 1988, 24(5), 513-523.

[3] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. journal of the american society for information science. 41, pp. 288-297, 1990.

[4] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. communications of the acm, vol. 51, no. 1, 2008, pp. 117-122.

[5] Python Software Foundation. Tensorly. http://tensorly.github.io.

[6] The Scipy community Copyright 2008-2009. Scipy. https://docs.scipy.org/doc/scipy-0.7.x/reference/.

[7] SciPy. Numpy. https://docs.scipy.org/doc/numpydev/user/quickstart.html.

[8] Mathieu Blondel Et al. Scikit-learn: Machine learning in python. http://scikit-learn.org/stable/.

[9] Mark Tygert Arthur Szlam, Yuval Kluger. An implementation of a randomized algorithm for principal component analysis, 2014. acm toms, 43(3): 28:1-28:14.

[10] Luis M. Rocha. Michael E. Wall, Andreas Rechtsteiner. Singular value decomposition and principal component analysis". in *A Practical Approach to Microarray Data Analysis.* d.p. berrar, w. dubitzky, m. granzow, eds. pp. 91-109, kluwer: Norwell, ma (2003). lanl la-ur-02-4001.

[11] M. Vazirgiannis K. Berberich, S. Bedathur and G. Weikum. Buzzrank . . . and the trend is your friend. in www06: Proceedings of the 15th international conference on world wide web, pages 937–938, new york, ny, usa, 2006. acm.

[12] J. Leskovec L. Backstrom. Supervised random walks: predicting and recommending links in social networks. in proceedings of the fourth acm international conference on web search and data mining. acm, 2011.

[13] J. Lin A. Sharma D. Wang R. Zadeh P. Gupta, A. Goel. Wtf: The who to follow service at twitter. in proceedings of the 22nd international conference on world wide web. international world wide web conferences steering committee, 2013.

[14] K. Lang R. Andersen, F. Chung. Local graph partitioning using pagerank vectors. in foundations of computer science, 2006. focs'06. 47th annual ieee symposium on, 2006.

[15] J. Leskovec J. Yang. Defining and evaluating network communities based on ground-truth. in proceedings of the acm sigkdd workshop on mining data semantics, page 3. acm, 2012.

[16] J.-Y. Pan H. Tong, C. Faloutsos. Fast random walk with restart and its applications. 2006.

[17] T. H. Haveliwala. Topic-sensitive pagerank. in proceedings of the 11th international conference on world wide web, pages 517–526. acm, 2002.

[18] J. Widom G. Jeh. Scaling personalized web search. in proceedings of the 12th international conference on world wide web. acm, 2003.

[19] D. Sivakumar Y. Jing J. Yagnik S. Kumar D. Ravichandran M. Aly S. Baluja, R. Seth. Video suggestion and discovery for youtube: taking random walks through the view graph. in proceedings of the 17th international conference on world wide web, pages 895–904. acm, 2008.

[20] P. Indyk A. Gionis and R. Motwani. Similarity search in high dimensions via hashing. proc. 25th int'l conf. very large data bases, 1999.

[21] M. Charikar. Similarity estimation techniques from rounding algorithms. proc. acm symp. theory of computing, 2002.

[22] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. proc. 30th symp. theory of computing, 1998.

[23] P. Indyk M. Datar, N. Immorlica and V. Mirrokni. Locality sensitive hashing scheme based on p-stable distributions. proc. symp. computational geometry, 2004.

[24] K. Grauman and T. Darrell. Pyramid match hashing: Sub-linear time indexing over partial correspondences. proc. ieee conf. computer vision and pattern recognition, 2007.

[25] J. Philbin O. Chum and A. Zisserman. Near duplicate image detection: Min-hash and tf-idf weighting. proc. british machine vision conf., 2008.

[26] B. Kulis P. Jain and K. Grauman. Fast image search for learned metrics. proc. ieee conf. computer vision and pattern recognition, 2008.

[27] M. Raginsky and S. Lazebnik. Locality sensitive binary codes from shift-invariant kernels. proc. advances in neural information processing systems, 2009.

[28] A. Andoni. Nearest neighbor search: The old, the new, and the impossible. phd dissertation, mit, 2009.

[29] P. Viola G. Shakhnarovich and T. Darrell. Fast pose estimation with parameter-sensitive hashing. proc. ieee int'l conf. computer vision, 2003.

[30] T. Darrell G. Shakhnarovich and eds P. Indyk. Nearest-neighbor methods in learning and vision: Theory and practice, the mit press, 2006.

[31] Zhang Xia-Huan Zhang Chen-Guang. Graph-based semi-supervised multi-label learning method", mechatronic sciences electric engineering and computer (mec) proceedings 2013 international conference on, pp. 1021-1025, 2013.

[32] Benjamin W. Heumann. An object-based classification of mangroves using a hybrid decision tree—support vector machine approach. remote sens. 2011, 3(11), 2440-2460.