CSE 515 Phase 3 Report

Greggory Scherer
Xiangyu Guo
Alfred Gonsalves
Chenchu Gowtham Yerrapothu
Siddhant Tanpure

Abstract

This paper describes several queries for retrieving data related to three different entities in a subset of the Internet Movie Database. These queries provide the ability to determine keyword relationships between actors and movies, genres and movies, and users and movies. Term frequency, inverse-document-frequency, and a relational model are some of the tools explored. In addition, several strategies for extracting latent semantics are used to determine hidden features and relationships between objects. These tools are then used together to create several recommendation systems and a few classification systems.

Keywords

IMDb, document, term frequency, inverse document frequency, dimension, measure, weight, vector, timestamp, rank, tag, differentiation function, dot-product, cosine similarity, random walk with restarts, personalized page rank, tensor, matrix, eigen decomposition, singular value decomposition, principle component analysis, latent Dirichlet allocation, support vector machine, k nearest neighbor, decision tree, Local Sensitive Hashing.

Introduction

The Internet Movie Database (IMDb) began as a searchable list of credits aggregated from a Usenet group and now contains over 4,517,776 titles and 8,136,918 people [1][2]. IMDb allows users join the site and vote on titles using a 1-10 scale, as well as provide feedback or comments on the titles [3]. For this project, a set of data was provided from IMDb containing relationships between titles (movies), actors, genres, and users. Movies ranked based on votes using an undisclosed formula [3]. IMDb movies are also tagged with a set of keywords.

Terminology

- **Document** refers to a collection of words, but can also be used to refer to an entity or object such as actor or genre.
- **Term Frequency (TF)** refers to a proportion of how often a term is appears in a document and the total terms in a document [4].
- **Term Frequency - Inverse Document Frequency (TF-IDF)** refers to the discriminative power a term has in a given document. In other words, how well a particular term describes a document compared to other terms.
- **Dimension** refers to any arbitrary aspect of an entity that is distinguishable from any other.
- **Measure** refers to a numerical value determined by the intensity of some dimension of an entity. For example, length is the measure of how long (intensity) something is.
- **Measurement Unit** (unit of measure) refers to a mapping of a measure discrete proportions of a predetermined referential measurement. For example, a meter is a predetermined referential measurement of length.
- **Weight** refers to the importance of particular measure. Higher weight is analogous to higher importance.
- **Vector** refers to an ordered series of entities, and in this project the entities are measures.
- **Timestamp** refers to a date and time which is logged for any arbitrary entity. A timestamp usually grants an entity the dimension of time, and in this document, a timestamp manifests itself in POSIX time, or seconds since January 1, 1970 at midnight UTC.
- **Rank** refers to the rank computed by IMDb.
- **Tag** refers to a keyword assigned to a particular movie by IMDb.
- **Differentiation Function** refers to a function that computes a single numerical measurement of difference between two vectors.
- **Cardinality** refers to the numerical relationship between two entities, in which three outcomes are possible: one to one, one to many, and many to many. It may also refer to the size of a set.
- **Array** refers to an ordered list of entities or objects with a fixed size
- **Matrix** refers to a two-dimensional array

- **Tensor** refers to a n-dimensional array
- **Latent Semantic** refers to some relationship or meaning that is not visible in data
- **PCA** refers to **Principle Component Analysis**, an algorithm for Latent Semantic Analysis
- **SVD** refers to **Singular Value Decomposition**, an algorithm for eigen decomposition and Latent Semantic Analysis
- **LDA** refers to **Latent Dirichlet Allocation**, a generative probabilistic model for topic modeling
- **Personalized Page Rank** refers to a specific version of the **Page Rank** algorithm that allows the specification of seed nodes
- **Range Search** refers to a type of query where a specified range of values is desired
- **Nearest Neighbor (NN)** refers to a type of query where a node is specified and similar nodes are desired
- **LSH** refers to **Local Sensitive Hashing**, which is a technique to perform range and nearest-neighbor queries using hash functions
- **SVM** refers to Support Vector Machines, which is a system to create spatial divisions in vector space in order to perform classifications
- **k-NN** refers to "k-Nearest Neighbor," which is a technique to use a nearest-neighbor search in order to perform classifications
- **Decision Tree** is refers to another tool that provides a tree-structure to perform classifications

Goal Description

This project phase contained 5 tasks. Henceforth, these will be referred to as Task 1, Task 2, Task 3, Task 4 and Task 5 respectively.

The goal of task 1 was to recommend five movies to the user based on information about the movie that he has already watched. In the first subtask of task 1, the goal was recommended movies by performing Singular Value Decomposition (SVD) or by using Principal Component Analysis (PCA) on the movie data. In the second subtask, the goal was to use Latent Dirichlet Analysis (LDA) to get the recommendation from the movies data. In the third subtask, the goal was to provide the recommendation by performing tensor decomposition on movies. In the fourth subtask, the goal was to perform personalized page rank on movies to get 5 recommendations for the user. In the 5th subtask, the goal was to use all the information that we calculated from above subtasks and provide the recommendation to the user.

The goal of task 2 was to improve the results of the above subtasks using probabilistic relevance feedback. In which the goal was user will provide feedback either positive, negative or no feedback for movies recommended in task 1. Based on that feedback 5 new movies will be recommended to the user using probabilistic relevance feedback.

The goal of task 3 was to convert movie data into 500-dimensional latent space and then applying Locality Sensitive Hashing (LSH) on that to get r most similar movies.

In task 4, the goal was to improve nearest neighbor matches using r-nearest neighbor based on relevance feedback mechanism.

In task 5 there were 3 subtasks. In the first subtask, the goal was to implement r-nearest neighbor-based classification to label movies based on already labeled movies. In the second

subtask, the goal was to implement decision tree-based classification to label movies based on already labeled movies. In the third subtask, the goal was to implement n-ary SVM based classification to get labeled movies.

Assumptions

1. A movie can only have one rank.
2. The cardinality of movies to tags is many to many.
3. The cardinality of users to movies is many to many.
4. The cardinality of genres to movies is many to many.
5. The cardinality of actors to movies is many to many.
6. When counting total actors, only actors in movies that have tags are important to consider.
7. When counting total genres, only genres for movies that have tags are important to consider.
8. When counting total users, only users that have tagged or voted on movies with tags are important to consider.
9. Higher ranks have lower numbers.
10. Invalid input will print error.
11. Queries with no results will return empty.

Proposed Solution

Use information retrieval techniques to determine similarities using TF-IDF and vector similarities for visible features, and PCA, SVD, LDA for latent semantic analysis. CP and PPR is also considered for cluster analysis. LSH proposed for hashing and searching. SVM, KNN and Decision Tree proposed for clustering.

Genre tag vector determination

Since movie rank is not taken into account in Task 1a, a comparatively simplified weighting function is used. The weighting function follows a similar procedure as in other tasks, however there is no multiplication with the inverse rank. Additionally, the timestamps are based on tags for movies in a given genre instead of movies in which an actor appears.

Given tag t_i, and $s \in timestamp(g, t_i)$ and $|\text{timestamp}(g, t_i)| > 0$,
s_{min} is the smallest timestamp in the entire database,
s_{max} is the largest timestamp in the entire database,
and $timestamp(g, t_i)$ is all timestamps with tag t_i for movies in genre g and tag t_i,

$$TF_{weighted}(g, t_i) = \frac{\left(\sum_{s \in timestamp(g,t_i)} \frac{s - s_{min} + 1}{s_{max} - s_{min} + 1}\right)}{\sum_{t_j \in T}\left(\sum_{s \in timestamp(g,t_j)} \frac{s_j - s_{min} + 1}{s_{max} - s_{min} + 1}\right)}$$

Given T_g is the set of tags t_i for movies in genre g,

$vector_{TF}(g) =< TF_{weighted}(g, t_1), TF_{weighted}(g, t_2), \ldots >$ for all t_i in T_g

Given a weight $w_i = weight(g, t_i)$,
the set of all genres G,
and the set G_{t_i} of all genres with movies with tag t_i
There is a function gpt of genres per tag:
$gpt(t_i) = |\{g | g \in G_{t_i}\}|$,

$$IDF(w_i, t_i) = \log\left(\frac{|G|}{gpt(t_i)}\right) * w_i$$

$$vector_{TF\_IDF}(g) =< IDF\left(TF_{weighted}(g, t_1), t_1\right), IDF\left(TF_{weighted}(g, t_2), t_2\right), \ldots >$$

Actor Tag Vector Determination

The team determined that the best way to generate a weighted TF by tag-age, tag-frequency, and movie rank was to first calculate the weight of the tag by both frequency and age and then multiply that weight by the weight of the rank. The weight of tag by age and frequency is computed by taking the distance of the timestamp from the minimum timestamp and dividing it by the total timespan offered in the database. In this way, the newest timestamps would be closer to 1 and the oldest timestamps would be closer to 0. The total timespan in the database was chosen instead of a measure of the distance from the current day in order to make the algorithm consistent regardless of the current date and to prevent the calculations from getting too small. By taking the sum of all these calculations the frequency of the tag is implicitly factored in. By adding 1 to the numerator and denominator, division by zero or other undefined cases. In order to compute rank, since lower numbers are considered higher rank, the inverse of the rank is used so that rank 1 is closer to 1 and the lowest rank (highest number) is closer to 0. The timestamps are based on the tags on movies in which an actor appears. These decisions are described in the equations below.

Given tag t_i, and $s \in timestamp(a, t_i)$and $|\text{timestamp}(a, t_i)| > 0$,
$s_{min}$is the smallest timestamp in the entire database,
$s_{max}$is the largest timestamp in the entire database,
and $timestamp(a, t_i)$ is all timestamps with tag t_i for movies with actor a,
and $rank(a, t_i)$ is the rank of the movie with actor a and tag t_i

$$TF_{weighted}(a, t_i) = \frac{\left(\sum_{s \in timestamp(a,t_i)} \frac{s - s_{min} + 1}{s_{max} - s_{min} + 1}\right)\left(\frac{1}{rank(a, t_i)}\right)}{\sum_{t_j \in T}\left(\sum_{s \in timestamp(a,t_j)}\left(\frac{s_j - s_{min} + 1}{s_{max} - s_{min} + 1}\right)\left(\frac{1}{rank(a, t_j)}\right)\right)}$$

Given T_a is the set of tags t_i for an actor a,
$vector_{TF}(a) =< TF_{weighted}(a, t_1), TF_{weighted}(a, t_2), \ldots >$ for all t_i in T_a

Given a weight $w_i = weight(a, t_i)$,
the set of all actors A,
and the set A_{t_i} of all actors in movies with tag t_i
There is a function apt of actors per tag:
$apt(t_i) = \left|\{a | a \in A_{t_i}\}\right|$,

$$IDF(w_i, t_i) = \log\left(\frac{|A|}{apt(t_i)}\right) * w_i$$

$$vector_{TF\_IDF}(a) = < IDF\left(TF_{weighted}(a, t_1), t_1\right), IDF\left(TF_{weighted}(a, t_2), t_2\right), \ldots >$$

Generic Task 1 Weighting Function

In Task 1, we used a generic function to weigh the results of each subsection. The function is as follows:

$$weight(result) = \sum_{lhs \in result, rhs \in lhs} similarity(lhs, rhs) * \frac{t_{lhs} - t_{min} + 1}{t_{max} - t_{min} + 1}$$

where t_{lhs} is the maximum timestamp for the movie tagged/rated by the user,
t_{min} is the minimum timestamp in the database for ratings and tags, and
t_{max} is the maximum timestamp in the database for ratings and tags, and
$similarity$ is an arbitrary similarity function.

It is assumed that "recency" and order is implicitly taken into account in the following way: if a user rated or tagged a movie more recently, it is likely that they have recently viewed the movie. As such, movies similar to these should be more important to the user.

Task 1a

Top 5 Movie Recommended for an user using PCA and SVD
PCA: Principal components Analysis
Approach for PCA: When the data matrix is passed in to the PCA function with the number of components as a parameter, it forms the covariance matrix (obtained from get_covarience()) and gives the decomposition of that covariance matrix. The decomposition consists of three matrices, left factor matrix (U), right factor matrix(U Transpose) and S the diagonals matrix with decreasing magnitude of the Eigen values.
The fit-transform function will give the factor matrix with n_component latent semantics as the features.
Explained variance will give the diagonal matrix with Eigen values related to the n- latent features in the decreasing order of the magnitude.
Here the data matrix passed is the Movie-tag matrix, the U matrix obtained is the movie- latent semantic matrix. U matrix is the divided in to two subsets 1) movies watched by the user and 2)

movies that are unwatched by the user then dot product similarities are calculated between each and every movie of watched and unwatched list of the users.
Once the similarities are calculated, the similarities are passed to a weight function which further adds weight to a movie based on the recency (time) of the movie watched by the user. The weighted similarities are then sorted to give the top 5 movies recommended for the user to watch

Libraries used: decompositions from sklearn
From sklearn import decompositions
PCA(n_components)
N_components : no of components to keep (no of latent features reduced from actual no of dimensions)
Parameters used for PCA:

components : array, shape (n_components, n_features)

> Principal axes in feature space, representing the directions of maximum variance in the data. The components are sorted by explained varience

explained_variance_: array, shape (n_components,)

> The amount of variance explained by each of the selected components.Equal to n_components largest eigenvalues of the covariance matrix of Data.

SVD: Singular valued Decomposition

Approach for SVD: When the data matrix is passed in to the SVD function, it gives the decomposition of that data matrix. The decomposition consists of three matrices, left factor matrix (U), right factor matrix (V Transpose) and S the diagonals matrix with decreasing magnitude of the square root of the Eigen values(singular values).

Here the data matrix passed is the Movie-tag matrix, the U matrix obtained is the movie- latent semantic matrix. U matrix is the divided in to two subsets 1) movies watched by the user and 2) movies that are unwatched by the user then dot product similarities are calculated between each and every movie of watched and unwatched list of the users.

Once the similarities are calculated, the similarities are passed to a weight function which further adds weight to a movie based on the recency (time) of the movie watched by the user. The weighted similarities are then sorted to give the top 5 movies recommended for the user to watch

Libraries used : linalg from numpy libraries

From numpy import linalg.svd :- Linalg.svd(data, full_matrices=1,compute_uv=1)

Parameters that svd function will take:

Data: data : (..., M, N) array_like
A real or complex matrix of shape (M, N)

full_matrices : bool, optional
If True (default), u and v have the shapes (M, M) and (N, N), respectively. Otherwise, the shapes are (M, K) and (K, N), respectively, where K = min(M, N).

SVD function Returns:

u : { (..., M, M), (..., M, K) } array
Unitary matrices. The actual shape depends on the value of full_matrices Only returned when compute_uv is True.
s : (..., K) array
The singular values for every matrix, sorted in descending order.
v : { (..., N, N), (..., K, N) } array
Unitary matrices. The actual shape depends on the value of full_matrices Only returned when compute_uv is True.

Task 1b

Latent Dirichlet Allocation (LDA) was employed in the following way in order to compare movies: a different model was generated using the tags for each the movies that a user has watched, and then each model is evaluated against tags of all the remaining movies. The evaluation of the model produces a probability distribution which is then compared via dot-product with the movie that the user has watched. This creates a mapping of user movies to other movies. After this, the generic task 1 weighting function is used to weigh the results.

Since model generation takes quite a long time, an assumption is made that 100 movies rated or tagged by the user is an adequate sample to determine suggestions. These 100 movies are randomly chosen based on the dictionary key iteration in Python. Assuming the distribution is uniform or normal, this should provide a fair representation of the population of movies rated or tagged by a user.

LDA: Latent Dirichlet Allocation:

LDA is a generative probabilistic model for collections of discrete data such as text corpora. There are various libraries that implement LDA.

The class genism.models.ldamodels.LdaModel provides various parameters that can be assigned at the time of declaration.
gensim.models.ldamodel.LdaModel(corpus=None, num_topics=100, id2word=None, distributed=False, chunksize=2000, passes=1, update_every=1, alpha='symmetric', eta=None, decay=0.5, offset=1.0, eval_every=10, iterations=50, gamma_threshold=0.001, minimum_probability=0.01, random_state=None, ns_conf=None, minimum_phi_value=0.01, per_word_topics=False, callbacks=None)

Parameters:
1. corpus – documents that need to be considered
2. num_topics – K number of random topics that are to be considered for assigning the words in the documents. This parameter takes the number of latent semantics required
3. id2word – this parameter takes a mapping from id (integers) to words (strings). It is used to determine the size of the vocabulary

LDA provides a probabilistic distribution upon its data. Based on this distribution feature is assigned to one of the hidden slots. This process is governed by multinomial distribution.
<doubtful about this statement>
The top_topics() will output topics based on the probability distribution.

Task 1c

Top 5 Movie Recommended for an user using CP

Approach: Once the three-mode tensor Tag-Movie-Rating is created and passed in to decomposition function with a target rank as 5, the tensor then decomposes to three factor matrices one for each mode of the tensor i.e. Tag, Movie and Rating respectively.
The initial factor matrices are chosen randomly and optimized using iterative approach till the converge is optimal, the library function uses ALS approach to optimize the decomposition.

Once we obtain the factor matrices, we select the factor matrix which contain movies as objects and is divided in to two subsets 1) movies watched by the user and 2) movies that are unwatched by the user then dot product similarities are calculated between each and every movie of watched and unwatched list of the users.

Once the similarities are calculated, the similarities are passed to a weight function which further adds weight to a movie based on the recency (time) of the movie watched by the user. The weighted similarities are then sorted to give the top 5 movies recommended for the user to watch

Libraries used : decompositions from tensorly, numpy
parafac : Library for Alternative least square approach to compute [CANDECOMP/PARAllel FACtors(PARAFAC)] CP decomposition. **parafac(tensor,rank,init)**
Result we get has a decomposed tensor with three factor matrices because it is a three-mode tensor. Factor matrices are returned with latent semantics, the number of latent semantics returned will be based on the target rank given (no of components)
The parametrs that the parafac function takes are:

Data: the input tensor to be decomposed
Rank: Tensor rank for the decomposition (no of latent features in to each factor matrices is decomposed in to).
Init: {'random','nvecs'} – optional parameter for initialization of factor matrices
-random: Factor matrices are initialized randomly.
-nvecs: Factor matrices are initialized via SVD.
Default is :'svd'
max_iter: int, this is also a optional parameter
Maximum nuber of iterations of the ALS algorithm.
Default is 100 iterations
tol: float- this parameter is optional
Tolerance: the algorithm stops when the variation in the reconstruction error is less than the tolerance
Default value is 1e-06

This library function returns:
Factors-: ndarraylist
List of factors of the CP decomposition element I is of shape (tensor.shape[i],rank)
It returns the factor matrices of the decomposed tensor along all the modes of the tensor.

Task 1d

We implemented the Personalized Page Rank (PPR) algorithm in task1d. The PPR algorithm will take a set of seeds as the teleport vector (v_q) to calculate the page ranking vector(u_q)
Transition matrix A: we construct the transition matrix by counting the outgoing degrees of each node 1/out_degree(node). And for nodes without outgoing edges, we set them to 1/number(nodes).
Initial vector u_q v_q: initially, we set each node in seed with 1/number(seeds) for both u_q and v_q.
Constant c, as we discussed in the class, the constant c is application related. And in Personalized Page Rank (PPR), we should increase the chances that our algorithm will back to the seed nodes. After few rounds experiment, we found out c = 0.2 works fine on our application.
When to stop(converged), when the max(Δu) < 1e-06, we consider our algorithm converged and stop.

Task 1e

Task 1e executes the queries derived in the previous subtasks (a-d) and compiles the results. This is done by computing the normalized weights computed in the subtask. The following function describes this:

$$weight(w_{i,j}) = \frac{w}{w_{total}}$$

where $w_{i,j}$ denotes the weighted similarity between all the user's movies and a movie i using a subtask j, and
w_{total} denotes the total similarity.

Then the weight w_i for a given movie i is given by:

$$w_i = \sum_{j \in subtasks} weight(i,j)$$

Task 2

Task 2 takes the results of any subtask of Task 1 and further refines the results using probabilistic feedback. The equation derived from [7] to compute the probabilistic feedback weight is described below:

$$prf(t_i) = \begin{cases} \log\left(\frac{\left(\frac{r+0.5}{R-r+0.5}\right)}{\left(\frac{m-r+0.5}{M-m-R+r+0.5}\right)}\right), if \begin{pmatrix} r = 0 \; or \\ M-m-R+r = 0 \; or \\ m-r = 0 \; or \\ m = r \; or \\ R = M \; or \\ R = r \end{pmatrix} \\ \log\left(\frac{\left(\frac{r}{R-r}\right)}{\left(\frac{m-r}{M-m-R+r}\right)}\right), otherwise \end{cases}$$

Where r is the number relevant movies (rated, tagged, and marked relevant) tag t_i,
m is the number of movies with tag t_i,
R is the set of all relevant movies, and
M is the set of all movies

And for a movie m,

$$weight(m) = \sum_{t \in m} prf(t)$$

Where the final ranking is given by:

$$ranking(m) = similarity(m) * weight(m)$$

Task 3

We implemented the Local Sensitive Hashing (LSH) algorithm in task3. The LSH algorithm will take a parameter L as the layer, and a parameter K as the number of the hash functions in each layer.

Reduced dimension t: since the original dimension d is 500, which is too high and will cause dimensionality curse, so we reduce it to t dimension. The value of t is equal to log(n), n is the objects(movies) in the database.

U grids of balls: as we learned from the paper, we need at least n, the number of the objects in the database of grids to cover the d space.

Constant w, as we learned from the paper, the constant w is greater than or equal to 0.5. We found out w = 0.5 works fine on our application.
Transition matrix A: we construct the transition matrix by filling the cell value using the Normal Distribution. We use this transition matrix to mapping the objects from a higher d dimension to a lower t dimension.
Offset vector Su_q: Su_q randomly select from a value from 0 to 2*w for each t dimension.
When we compute the hash value, first we convert the query point from d dimension to t dimension using the transition matrix we created above. Then we go through U grids try to find a first grid with the ball can cover out query point, combine the grid id with the coordinate of the ball return as the hash value.
The program will stop, once we collected enough data, otherwise it will use the LSH-Forest strategy to find more data.

Task 4:

Based on the user feedback, we want to know how to move our query point to a relevant direction. The user feedback for relevant data (movies he liked) is considered as the positive feedback, we want to move towards to that direction. Once the input is taken and the distance of those movies are calculated from the U(movie- latent semantic) matrix and the new_direction is updated by adding those new similarities to it.
Similarly, the user feedback for the irrelevant movies (movies the user doesn't like) is considered as the negative feedback, we want to move away from that direction. Once the input is taken and the distance is calculated and it is subtracted from the new_direction vector.
At last, we shift our query point using this new_direction vector, and rerun the search.

Task 5

Classification :
For the classification task we consider Movie-tag matrix as the primary input and upon preprocessing and dimensionality reduction using SVD, the dimensions of the Movie-tag matrix are reduced to the optimal log (n) to base 2 dimensions, where n is the total number of movies in the movie- tag matrix. We are performing the dimensionality reduction keeping in mind the ease of computability and also the discrimination of the features (tags).
Movies are labelled based on the tags given by the users. Taking the different tags given in to account movies are labelled as Adventure, thriller, comedy, action and so on. Here the assumption is that the movies are labelled based on the tags given by the user.
Once the SVD on the Movie-tag matrix is calculated, the U matrix with the movies and latent semantics as their features is sent in to the different classification functions respectively to perform SVM, r-NN and Decision tree based classification tasks

n-ary SVM description:

Though we have several max-margin classifiers like linear separators, polynomial separators and spherical separators, a linear separator model n-ary SVM(support vector machines) based

classification is used as one of the classification method for this task. In the n-ary SVM a class label is compared with the rest of the class labels in the dataset and the closest of the labels from the group to the single label is considered. Once the closest labels are found then midpoint of those two labels is calculated and a hyper plane is formulated as a decision boundary because it maximizes the margin between those class labels at the mid point. Based on whether the query point is on left or the right side of the decision boundary and majority selection of labels on each side, the classification label is given to a new movie (object).

r-NN Description:

The input to the r-Nearest Neighbor algorithm is the dimension reduced matrix formed by performing SVD over the entire movie-tag matrix. The new movie (query data) that is to be classified is taken and the distance between every other movie in the labelled movies set is calculated and based on the r, the top most r minimum distanced labels are chosen and among those r- top selections majority selection is performed again for labelling then the classification is done and label is assigned to the new movie.

Decision Trees:

The imput for the decision tree is all movies and tag vector information. The input is given to SVD to reduce the dimensions. Then the U matrix is given as an input to the decision tree builder function which then created the gini and information gain on each column to identify the features which are more informative. Then decision tree was built based on the gini and information gain. Each row from U is provided as an input to the tree and labeled using the tree.

Interface Specifications

The system-level interface is by command line. The following commands should produce the corresponding outputs for each task.

p3_task1a <user_id> <PCA|SVD> [pf]

Param 1: user_id, a valid user id without quote, e.g. 41154
Param 2: model, e.g. PCA
Param 3: pf, enable task2 probabilistic feedback.

p3_task1b <user_id> [pf]

Param 1: user_id, a valid user id without quote, e.g. 41154
Param 2: pf, enable task2 probabilistic feedback.

p3_task1c <user_id> [pf]

Param 1: user_id, a valid user id without quote, e.g. 41154
Param 2: pf, enable task2 probabilistic feedback.

p3_task1d <user_id> [pf]

Param 1: user_id, a valid user id without quote, e.g. 41154
Param 2: pf, enable task2 probabilistic feedback.

p3_task1e <user_id> [pf]

Param 1: user_id, a valid user id without quote, e.g. 41154
Param 2: pf, enable task2 probabilistic feedback.

p3_task3 <l> <k>

Param 1: l, the number of layers in LSH.

Param 2: k, the number of hash functions in each layer

For the further interface information, please follow the instruction in each command.
For the interface of task2/4, also follow the instruction indicated in each task1/3 command.

p3_task5 <NN | DT | SVM> <r>

Param 1: classification algorithm from NN|DT|SVM, e.g. NN.

Param 2: r range

reset wc

Helper function to release the memory

System requirements/installation and execution instructions

0. Running environment, any machine/OS with Python3.0 installed. (numpy, tensorly, sklearn, gensim are prerequisite Python libraries)
1. Put all csv data file under the “data” folder.
2. Go to the “Code” folder using your command line tool. (Shell or CMD)
3. Execute the program with “python3 main.py”.
4. Wait for around 20 seconds, you will see the following information during this progress.
“loading csv data into memory...
loading completed!
preprocessing data...
preprocessing completed!”
5. After you see a “query>”, you can running your query now.
6. How to run each query command, please refer the section “Interface Specifications” for more detail.

Related Work

Pan, Yang, Faloutsos, and Duygulu’s paper, *Automatic Multimedia Cross-modal Correlation Discovery*, provided an algorithm for Personalized Page Rank, which was used in Task 1d in order to provide movie recommendations. The concept in this paper is we can represent a graph using a matrix. Then for the connectivity or the transition in the graph, it can be done using a matrix multiplication operation. So, we can implement it in our project using numpy library.

Salton and Buckley's paper, *Improving Retrieval Performance by Relevance Feedback***,** provided a method and mathematical equation in order to implement probabilistic relevance feedback. This was used for Task 2 in the project. The concept in this paper is only provide a way to measure the relevance between two objects, but we can also use this for irrelevant feedback, just using the same equation and the algorithm, and flip/negate the result we get. This feedback is then incorporated into the results and the relevance values can be further updated by the user.

Andoni and Indyk's paper, *Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions*, provided the means to generate a hash family in order to perform nearest-neighbor searches. In this paper, a concept of mapping a vector space into a grid of balls to create a family of hash functions is introduced. A grid of balls replaces a random vector in other implementations and the paper describes the usage of Euclidean distance instead of Hamming distance. Also, we reduce a higher dimension to a lower dimension to avoid the dimensionality curse.

For implementing decision tree used this link to understand the concept. It explains methods to calculate gini and information gain from training data. Which helps to create decision feature and decision node. The entire tree is built first and then each movie related value is provided to the tree to label the movie.

Conclusions

Term frequency and inverse-document-frequency measurements help determine relevance of a term to an entity and the discrimination power of a term respectively. Using these measurements, it is possible to further determine similarities between objects using both observed and latent semantics. Singular Value Decomposition, Principle Component Analysis, Latent Dirichlet Allocation, and Personalized Page Rank are algorithms that can be used to analyze latent semantics. Observed features can be compared using various similarity measures, such as dot-product, cosine similarity, or even Euclidean distance. The more sophisticated algorithms help exploit hidden relationships and can be useful when trying to build recommendation systems.

CP decomposition and LDA gave nondeterministic results when used for recommendation systems. This made validation difficult as a change in result could be a result of a different random seed or a bug in the code. When using LSH, with larger l and k values, indexing quickly became an enormous task.

Bibliography

[1] Col Needham. 2010. IMDb History. (October 17, 2010). DOI: http://www.imdb.com/help/show_leaf?history
[2] IMDb Stats. Retrieved September 17, 2017 from http://www.imdb.com/stats
[3] IMDb Votes/Ratings Top Frequently Asked Questions Retrieved September 17, 2017 from http://www.imdb.com/help/show_leaf?votestopfaq
[4] How to compute TF-IDF. Retrieved September 17, 2017 from http://www.tfidf.com
[5] K. Selçuk Candan and Maria Luisa Sapino. 2010. Data Management For Multimedia Retrieval.
[6] SVD PCA and CP libraries from http://scikitlearn.org, https://docs.scipy.org andhttps://tensorly.github.io
[7] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. Journal of the American Society for Information Science. 41, pp. 288-297, 1990.
[8] "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.
[9] Jia-Yu Pan, Hyung-Jeong Yang, Christos Faloutsos, Pinar Duygulu. 2004. Automatic Multimedia Cross-modal Correlation Discovery. (August 2004) DOI: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.85.8179&rep=rep1&type=pdf
[10] Juan Ramos. 2003. Using TF-IDF to Determine Word Relevance in Document Queries. (January 2003) DOI: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.1424&rep=rep1&type=pdf

Appendix

Specific Roles of Group Members

1. Xiangyu Guo - Group discussion, Independent implementation, assistance on specific topics, assistance on debugging, testing
2. Siddhant Tanpure - Group discussion, Independent implementation
3. Chenchu Gowtham Yerrapothu - Group discussion, Independent implementation
4. Alfred Gonsalyes - Group discussion, Independent implementation
5. Greggory Scherer - Group discussion, Independent implementation, assistance on specific topics, assistance on debugging, testing