CSE 515: Multimedia and Web Databases
Project Report: Phase 3
Group 18, Fall 2018

Group Members

1. Aditya Kanchivakam Ananth
2. Aniket Rajendra Dhole
3. Shatrujit Singh
4. Siddharth Pandey
5. Siddhesh Sudesh Narvekar
6. Vasisht Sankaranarayanan

Abstract

As data grows in volume, velocity and variety it becomes increasingly difficult to extract relevant information from it. Finding internal structure of the data, partitioning it, and using this information to find similarity inside data can be used in a variety of applications such as summarizing the news, web search engines and even finding similarity between songs. In this project, we have implemented measures to group data in order to find relationship between objects, rank the importance of nodes in a data set and also decrease the complexity of similarity searching from linear to sub-linear by using approximate index structures.

Keywords

- TF – Term Frequency
- DF – Document Frequency
- IDF – Inverse Document Frequency
- TF-IDF – Term Frequency-Inverse Document Frequency
- CM – Global Color Moments on HSV Color Space
- CN – Global Color Naming Histogram
- HOG – Global Histogram of Oriented Gradients
- LBP – Global Locally Binary Patterns on Gray Scale
- CSD – Global Color Structure Descriptor
- GLRLM – Global Statistics on Gray Level Run Length Matrix
- Code 3x3 – Spatial Pyramid Representation (CN3x3, CM3x3, LBP3x3, GLRLM3x3)

- PR – PageRank
- PPR – Personalized PageRank
- LSH – Locality Sensitive Hashing
- *k*-NN – *k*-Nearest Neighbors algorithm

Introduction

- **Terminology**
  - Term Frequency (TF): Term Frequency is a measure of the weight of a given keyword *k* in a given document *d*. It is mathematically defined as

$$tf(k,d) = \frac{count(k,d)}{size(d)}$$

  - Document Frequency (DF): Document Frequency is a measure of the total number of documents in the database *D* containing the given keyword *k*. It is mathematically defined as

$$df(k,D) = \frac{number\ of\ documents\ containing(k,D)}{number\ of\ documents(D)}$$

  - Inverse Document Frequency (IDF): Inverse Document Frequency is an inverse measure of DF. It is mathematically defined as

$$idf(k,D) = \log\left(\frac{number\ of\ documents(D)}{number\ of\ documents\ containing(k,D)}\right)$$

  - Term Frequency-Inverse Document Frequency (TF-IDF): TF-IDF weight of the keyword *k* for document *d* in database *D* combines the aspect of TF and IDF:

$$tf_{idf(k,d,D)} = tf(k,d) * idf(k,D)$$

  - PageRank algorithm: PageRank is an algorithm to rank nodes in a directed graph, by their importance. PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important a node is. The main assumption is that an important node will receive more links from some other node.

  - Personalized PageRank algorithm: Personalized Page Rank is the same as PageRank with the only difference of restarting from a set of given nodes, rather than choosing a random node. In a sense, random walk is biased towards the given set of nodes and is more localized compared to the PageRank's random walk.

- Locality Sensitive Hashing: Locality Sensitive Hashing is a hashing based algorithm. It aims to preserve local relations of the original dataset while also reducing its dimensionality significantly. LSH is used to identify approximate nearest neighbors. The algorithm runs in sub-linear complexity which is made possible by reducing the number of comparisons needed to identify similar items. A hash function h is locally sensitive if for two points a and b, in a high dimensional feature space:
    - Pr(h(a) == h(b)) is high if a and b are near
    - Pr(h(a) == h(b)) is low if a and b are far

- *k*-Nearest Neighbor Classifier: The *k*-NN classification is a voting-based scheme, where an object is classified based on the majority vote of its *k* most similar objects. The efficiency of the *k*-NN classifier depends on the value of k, the dimensionality of the vector space and the index structure used to identify the nearest neighbors. Large *k* values ensure that the less representative neighbors do not affect class selection, on the other hand having very large *k* values can have an adverse effect as it may cause large classes to over vote small classes nearby.

- **Goal Description**
    - **Task 1:** Implement a program which, given a value *k*, creates an image-image similarity graph, such that from each image, there are *k* outgoing edges to *k* most similar/related images to it.

    - **Task 2**: Given the image-image graph, identify *c* clusters (for a user supplied *c*) using two distinct algorithms. You can use the graph partitioning/clustering algorithms of your choice for this task. Visualize the resulting image clusters.

    - **Task 3**: Given an image-image graph, identify and visualize *K* most dominant images using Page Rank (PR) for a user supplied *K*.

    - **Task 4**: Given an image-image graph and 3 user specified image ids identify and visualize *K* most relevant images using personalized PageRank (PPR) for a user supplied *K*.

    - **Task 5:**
        - **a.** Implement a Locality Sensitive Hashing (LSH) tool, for a and similarity/distance function of your choice, which takes as input
            (a) the number of layers, *L*,
            (b) the number of hashes per layer, *k*, and

(c) a set of vectors as input and creates an in-memory index structure containing the given set of vectors.

b. Implement similar image search using this index structure and a combined visual model function of your choice (the combined visual model must have at least 256 dimensions): for a given image and *t*, visualizes the *t* most similar images (also outputs the numbers of unique and overall number of images considered).

- **Task 6:** Implement
  - **a.** a *k*-nearest neighbor based classification algorithm, and
  - **b.** a PPR-based classification algorithm

  which take a file containing a set of image/label pairs and associates a label to the rest of the images in the database. Visualize the labeled results.

- **Assumptions**
  a. All terms are keywords.
  b. The input file with image/label pairs will always have image id as first column and image label as second column
  c. The term frequencies have been defined differently in the dataset.
     - Term frequency: Count of the term in a document.
     - Document frequency: Count of the documents in the database containing the term.
     - TF-IDF: This has been set to the value given by a division of term frequency by document frequency.

Description of the Proposed Solution

Task 1:

In this task, the visual descriptor model's data for each image is gathered and similarity measurement is done for every pair of images based on their visual descriptor vectors and this data is further pruned to keep only the top k similar images for a given image where k is input by the user. This graph is further used for other tasks.

Task 2:

For task2 we use the graph obtained from task1 which consists of k nearest/similar images to each image in the dataset.

A. bfsClustering: bfsClustering algorithm makes use of bfs algorithm to create clusters. We randomly select a node from our dataset and run bfs. Using bfs we are assured that the images/nodes that we

explore for the current cluster will be close to each other. We stop exploring new images/nodes for the current cluster when our queue becomes empty or the number of images in a cluster exceeds total (nodes or images) / k. Here, k is the number of clusters we need to create.

B. kSpanningTree: In this algorithm, we create a minimum spanning tree(MST) for our graph obtained from task1. We make use of Prim's Minimum Spanning Tree algorithm to find the MST for our graph. Once we obtain MST, we delete k - 1 random edges from our MST. This leaves us with k distinct MST's. The images/nodes in each of these k distinct MST's form our cluster.

Task 3:

Ranking of images based on PageRank is modelled around the formula

$$(\beta * M) + ((1 - \beta) * E)$$

M is considered to be the transition matrix denoting the neighbors of a node along with the probability of a given node to jump to those neighbors.

E matrix is considered to be the probability matrix for restarting where a new node is selected with equal probability.

Beta is damping factor which denotes the probability of continuing along the path of neighbors of nodes.

These parameters have been explained in detail in the implementation section.

Node B is considered to be a neighbor to a given node A if node B appears in the list of k most similar images of A.

The initial PageRank's of all pages is considered to be of equal value 1/N (where N is size of dataset).

The iterative process of finding a better estimate of the page ranks starting from the initial random page rank values is done until convergence, wherein the page ranks do not change above a certain threshold. Since page rank is a probability measure, the sum of page ranks calculated is checked for equality to 1.

Task 4:

Ranking of images based on personalized page rank is modelled around the formula

$$(\beta * M) + ((1 - \beta) * E)$$

M is considered to be the transition matrix denoting the neighbors of an image along with the probability of a given image to link to those neighbors.

E matrix is considered to be the probability matrix for restarting where only one of the images input by the user is selected with probability 1/3 (As user supplies 3 images).

Beta is damping factor which denotes the probability of continuing along the path of neighbors of nodes.

These parameters have been explained in detail in the implementation section.

Node B is considered to be a neighbor to a given node A if node B appears in the list of k most similar images of A.

The initial page ranks of all pages are considered to be of equal value 1/N (where N is size of dataset).

The iterative process of finding a better estimate of the page ranks starting from the initial random page rank values is done until convergence, wherein the page ranks do not change above a certain threshold. Since page rank is a probability measure, the sum of page ranks calculated is checked for equality to 1.

Task 5:

- **a.** Locality sensitive hashing is performed by constructing the desired number of hash tables (or layers) using the desired number of hashes per layer and the number of dimensions in the input dataset. Each hash table consists all possible buckets and images which lie inside that particular bucket. Each bucket is represented as a bitwise hash value. Observations with similar hash values are more likely to be similar to each other than those with different hash values.
- **b.** In order to determine images which are most similar to the input image, we lookup the hash tables and retrieve images which are same bucket as the input image in at at least one of the tables. Cosine angle is computed between all of these images and the input image, and the ones with highest scores are ranked as most similar.

Task 6:

- **a.** *k*-nearest neighbor based classification: The textual descriptors of each image present in the input file is stored. These terms are later used to calculate the cosine similarity between an image in the input file and a given image in the database which shares these common terms with the images in the input file. The *k* most similar images from the input file are selected for each image in the database. The labels for these *k* images is then used for a vote and the label with the most votes is the label which is thus assigned to the given image.
- **b.** Personalized Page Rank based classification: The adjacency matrix for all images has been used along with a vector containing restart probabilities for seed nodes. 100 iterations for calculating the page ranks are made with probability of restarting as 0.15. Once the page ranks are calculated, similar images from the input file are found and the most similar images' label is assigned to the given image.

Implementation

Task 1:

Preprocessing is done before similarity calculations are done to collect all the data into a manageable data structure for performing calculations.

For a given location, every image is considered in sequence and its corresponding visual descriptors are clubbed together to form a data frame.

E.g. Consider Location 1.

Location 1 Feature 1 Feature 2 Feature 3

Image1	CM	CM	CM	CM	CN	CN	CN	CN	MW	MW	MW	MW		
Image2	CM	CM	CM	CM	CN	CN	CN	CN	MW	MW	MW	MW		

The data frame has image id as an index. For all the images, Feature 1 column is first added to the data frame. Then, Feature 2 columns are added to the data frame in the manner shown above. Similarly, all features for all images are combined to form the data frame. This process is followed for every location and the final data frame consists of all the images from all the locations with their corresponding visual descriptors. The final DF looks as follows:

Loc1	Image1	CM	CM	CM	CM	CN	CN	CN	CN	MW	MW	MW	MW		
	Image2	CM	CM	CM	CM	CN	CN	CN	CN	MW	MW	MW	MW		
Loc2	Image3	CM	CM	CM	CM	CN	CN	CN	CN	MW	MW	MW	MW		
……	Image4	CM	CM	CM	CM	CN	CN	CN	CN	MW	MW	MW	MW		
	Image5	CM	CM	CM	CM	CN	CN	CN	CN	MW	MW	MW	MW		
Loc k	Image6	CM	CM	CM	CM	CN	CN	CN	CN	MW	MW	MW	MW		

For each image in this DF, its cosine distance with every other image is calculated and stored in a data structure as follows:

	Image 1	Image 2	Image 3	Image 4
Image 1	0.0	0.7	0.3	0.5
Image 2	0.3	0.0	0.2	0.6
Image 3	0.4	0.2	0.0	0.1
Image 4	0.6	0.9	0.2	0.0

Based on user input k, for each image, the top k images (closest by distance) are picked and stored in a new data structure. The final output of this task will be this data structure.

a = [

[Image1, [(Image1, 0.0), (Image7, 0.3), ….(Imagek, 0.9)]],

[Image2, [(Image2, 0.0), (Image7, 0.3), ….(Imagek, 0.9)]]

[Imagen, [(Imagen, 0.0), (Image7, 0.3), ….(Imagek, 0.9)]]

]

Task 2:

A. bfsClustering: As described earlier, we make use of Breadth First Search algorithm to create clusters. From the given dataset, we randomly select an image, add it to our queue and explore adjacent images. So, we start exploring the images/nodes for our current Cluster. As this approach can continue for as long as we have connected edges, we added a stop case in-order to stop exploring further and for the current cluster. This stop case is based on two conditions, first, is when our queue would be empty or second, when our current cluster size exceeds total(nodes or images) / k. So, we have our current graph(cluster) saved in currGraph dictionary. Once our exploration exhausts, we add currGraph to our result. The images/nodes that we add to currGraph are removed from our original graph.

We find k - 1 clusters in the approach mentioned above. Now, after finding k - 1 clusters, our last cluster will consist of images/nodes that are remaining in our original graph. If we add the remaining graph as the last cluster, we might end up adding edges that might link the current cluster to the previous clusters. So, to avoid this, we process each image/node in the remaining graph and remove the edges that have connections to the previous clusters. Post this task, we add the modified graph to our cluster as the last cluster.

B. kSpanningTree: kSpanningTree algorithm requires Minimum Spanning Tree(MST) of a given graph to divide it into k clusters. As mentioned in the description, make use of Prim's Minimum Spanning Tree algorithm to calculate the MST for our graph obtained from task1.

To calculate MST, we first converted the directed graph obtained from task1 into undirected graph, with connections between images/nodes were represented by the distance/weights between those two images/nodes.

Now, we follow the below approach to find clusters:

edgeSet → list that consists of all the edges initially

for _ in range(k - 1):

select random edge to cut

find which edgeSet does the edge belong:

remove the edge from that edgeSet

create2Partition → returns set of nodes for each partition

add the above sets to our result

create2EdgeSets → returns set of edges for the two partitions

add the above sets to our edgeSet

After completion of the above algorithm, our result consists of k sets. Each set consists of images/nodes for the k respective clusters.

Now, our result consists only of the set of images/nodes and not the edges between the nodes of each set. So, we further process the result and create graph(connections) for each of the images/nodes present in each cluster.

Task 3:

The images (identified by image IDs) are mapped to integer values (1,2,…N) for easier reference and stored in a data structure as follows:

[ImageID: 1, ImageID: 2, ……., ImageID: N]

The weighted transition matrix is then constructed. For each image in the dataset, its outgoing edges are the k most similar images. From Task 1, we get the most similar images of a given image stored in the following data structure:

[(Image1, [(Image2, 0.9), (Image3, 0.7)])]

The probability of going from Image 1 to one of its neighbors is calculated as follows.

From the above data structure, Image 7 is more similar to Image 1 than Image 9. So, probability of going from Image 1 to Image 7 should be more than probability of going from Image 1 to Image 9. The similarity score is normalized to be considered as a probability measure.

P(Image1 -> Image7) = 0.7/(0.7+0.5) and P(Image1 -> Image9) = 0.5/(0.5+0.7)

Here, we consider (1-distance) because, lesser the distance more similarity, hence more probability.

This will ensure that probability sum is 1 as well as it is weighted to give preference to more similar images.

So, in the transition matrix, the row for Image 1 will be as follows (Final transition matrix will have zeroes in the cells for images which are not similar to given image):

	Image 2	Image 3
Image 1	0.5625	0.4375

Similarly, for all images, the transition matrix is populated with the necessary probabilities.

	Image W	Image X	Image Y	Image Z
Image A	0.5	0.3	0.2	0
Image B	0	0	0.8	0.2
Image C	0.4	0	0.3	0.3
Image D	0	1	0	0

Here, a cell T has value 0 denoting 0 probability of reaching that image directly. That cell T is required as some other image is directly connected to it and has a certain probability of going to that image.

This weighted transition matrix is M.

The 1/N matrix (where N is size of dataset) is generated and stored in E. It has size NxN with every value as 1/N. This matrix is E.

Beta is considered to be 0.85

BM + 1-B * E is stored in A initially.

r matrix is the initial page rank matrix which is populated with 1/N (N is size of dataset). It is a 1xN matrix with every value as 1/N.

Until convergence, r = A*r is done. Convergence is checked by comparing current r to r from the previous iteration and seeing if there is change above a threshold value.

The R matrix finally has all the page ranks of the N nodes of the dataset. Based on user input k, the top k values in R is picked and the corresponding images from the image dictionary created are mapped and picked and they are displayed.

Task 4:

The images (identified by image IDs) are mapped to integer values (1,2,…N) for easier reference and stored in a data structure as follows:

[ImageID: 1, ImageID: 2, ……., ImageID: N]

The weighted transition matrix is then constructed. For each image in the dataset, its outgoing edges are the k most similar images. From Task 1, we get the most similar images of a given image stored in the following data structure:

[(Image1, [(Image2, 0.9), (Image3, 0.7)])]

The probability of going from Image 1 to one of its neighbors is calculated as follows.

From the above data structure, Image 7 is more similar to Image 1 than Image 9. So, probability of going from Image 1 to Image 7 should be more than probability of going from Image 1 to Image 9. The similarity score is normalized to be considered as a probability measure.

P(Image1 -> Image7) = 0.7/(0.7+0.5) and P(Image1 -> Image9) = 0.5/(0.5+0.7)

Here, we consider (1-distance) because, lesser the distance more similarity, hence more probability.

This will ensure that probability sum is 1 as well as it is weighted to give preference to more similar images.

So, in the transition matrix, the row for Image 1 will be as follows (Final transition matrix will have zeroes in the cells for images which are not similar to given image):

	Image 2	Image 3
Image 1	0.5625	0.4375

Similarly, for all images, the transition matrix is populated with the necessary probabilities.

	Image W	Image X	Image Y	Image Z
Image A	0.5	0.3	0.2	0
Image B	0	0	0.8	0.2
Image C	0.4	0	0.3	0.3
Image D	0	1	0	0

Here, a cell T has value 0 denoting 0 probability of reaching that image directly. That cell T is required as some other image is directly connected to it and has a certain probability of going to that image.

This weighted transition matrix is M.

The images input by the user are considered and their indices are found from the map created above. A 1xN matrix with value 1/3 at the positions of the given images and value 0 at other positions is created and this is stored in E.

Beta is considered to be 0.85

BM + 1-B * E is stored in A initially.

r matrix is the initial page rank matrix which is populated with 1/N (N is size of dataset). It is a 1xN matrix with every value as 1/N.

Until convergence, r = A*r is done. Convergence is checked by comparing current r to r from the previous iteration and seeing if there is change above a threshold value.

The R matrix finally has all the page ranks of the N nodes of the dataset. Based on user input k, the top k values in R is picked and the corresponding images from the image dictionary created are mapped and picked and they are displayed.

Task 5:

An aggregated data frame is constructed utilizing data from all visual models: CM, CM3x3, CN, CN3x3, CSD, GLRLM, GLRLM3x3, HOG, LBP, LBP3x3. Each hash table is constructed as a dictionary with key, value pairs which will be all possible buckets and list of images which lie inside that particular bucket. The dictionary is populated as follows:

- A random projection matrix is created of shape (k, d) where k is the number of hashes per layer and d is the total number of dimensions in the input dataset.
- Dot product is computed between the random projection matrix and each row in the aggregated data frame. If the value of the dot product is positive, bit value is assigned as 1, else the bit value is assigned as 0.
- All the k bit values computed in the above step are concatenated into a string. The result is a hash value. This hash value becomes a new key in the hash table dictionary with its value as the image ID of that particular row.
- The above two steps are repeated for all rows in the input data frame and hash values are thus computed for all images.

For example:

- Consider this input dataset consisting of four 5-dimensional vectors :

	0	1	2	3	4
A	-2.351078	1.713649	-0.970669	-0.262087	-1.352072
B	0.307681	1.015936	-0.064666	0.808742	0.050309
C	-0.310378	1.302252	-0.128743	-1.102121	0.564472
D	-2.951078	1.113649	-0.970669	-0.362087	-1.352072

- A random projection matrix is generated from the 'standard normal' distribution with shape as the desired number of layers and number of dimensions in the input dataset. Let us assign the number of layers as 2:

	0	1	2	3	4
r1	-0.228035	-0.027617	1.323876	3.504398	0.328532
r2	-0.178271	-0.739520	-0.177264	0.946814	0.817578

- Dot product is computed between each vector in the input dataset and the transpose of the random projection matrix. The resulting computations will be as follows:
    - For 'A':
        - Dot product: [-2.15890039, -2.02965574]
        - Hash value: [0, 0]
    - For 'B':
        - Dot product: [2.66685385, 0.01216775]
        - Hash value: [1, 1]
    - For 'C':
        - Dot product: [-3.81244876, -1.46689246]
        - Hash value: [0, 0]
    - For 'D':
        - Dot product: [-2.35594782, -1.57366286]
        - Hash value: [0, 0]

From the above calculations, it is clear that 'A', 'C' and 'D' have the same hash value. This means that they are more likely to be similar.

The reasoning behind this logic is that if two vectors are aligned completely (or they have exactly same correlation from origin) then they will be in the same hash bucket. Following the same logic, if two vectors are completely different (or separated by 180 degrees) then will be in different hash buckets. If two vectors are separated by 90 degrees then there is a 1/2 probability that they will be in the same hash bucket.

However, since there is randomness involved it isn't likely that all similar images will fall into the same hash bucket using just one hash table. We will end up with a lot of misses. To overcome this issue, we create multiple hash tables and consider the results from all the hash tables. For high dimensional data, it is much better if we use higher number of tables. Hash buckets from all the tables can then be taken

into consideration and used in conjunction to reduce the number of false positives. The following steps are followed:

- Construct new dictionaries which act as hash tables according to the desired input using the steps described above.
- Append these dictionaries to a list.
- For a given input image ID, generate the hash value for that input image's vector in all the hash tables.
- Take a union of all the image IDs which lie inside the dictionaries in the hash values generated above.
- Compute cosine distance between the images received above and rank in order of increasing similarity.

For example:

- The first hash table is a dictionary which looks something like this:

```
{
    '[0, 0]': ['A', 'C', 'D'],
    '[1,1]': ['B']
}
```

- Suppose the user input for the number of hash tables to be used is 2, then we go about creating another dictionary similarly. Using another random projection steps, we end up with another dictionary which looks like this:

```
{
    '[0, 1]': ['A', 'D'],
    '[0, 0]':['C'],
    '[1, 1]': ['B']
}
```

- Suppose the input image for which we have to find the similar image is 'A'. We look up both hash tables and take a union of all the images which lie in the hash bucket of 'A'. In this case the list will be: ['A', 'C', 'D'].

- Now that we have narrowed down on the possible similar images we will compute the cosine angle between 'A' and 'C', and 'A' and 'D':
    - For 'A' and 'C': 0.4249674966667078
    - For 'A' and 'D': 0.9717729195892064

From the above calculation it is clear that 'A' and 'D' have a cosine angle closer to 1.0 and are therefore more similar to each other.

By extending this logic on much bigger datasets we will find that the number of comparisons performed on a much smaller subset of the input dataset. This helps to have speedy computations. Additionally, hash tables only store the labels and not the entire feature vectors saving a lot of memory.

Task 6:

a. *k*-nearest neighbor based classification: We read the input file containing image/label pairs and store the image id and the image label in a dictionary "inputFile". The image dataset file is then read and the image id, the textual descriptor terms and tf-idf values of the terms are stored in a dictionary "image_dict". The same dictionary is used to store a list of terms of the images from the input files into another dictionary named "inputFileTerms".
This dictionary is used to find the database images having matching terms after which the cosine similarity is calculated for each image in the database (image_dict) with respect to the images in the input file (inputFileTerms). This cosine similarity score is used to find the *k*-nearest neighbors (from images in input file) for a given image in the database. The images with higher similarity scores are considered more similar to a given image than images with lower similarity scores. The image labels for the *k*-nearest neighbors are read from "inputFile" dictionary and the *k* labels are counted with the label having the most votes being assigned to the given image. This process is repeated for all images in the database. All the images with same label are appended to a single file. Thus, the number of files created will be equal to number of distinct labels in the input file.

b. Personalized Page Rank based classification: We read the input file containing image/label pairs and store the image id and the image label in a dictionary "inputFile". Similarity between a given user and input file images is computed using Personalized PageRank algorithm on basis of the most similar image being the closest ranked image. This similarity is used to assign the label to all the images in the database. All the images with same label are appended to a single file. Thus, the number of files created will be equal to number of distinct labels in the input file.

System Requirements

- Operating System: Windows x64 architecture
- Python version: 3.7
- Packages: NumPy, Pandas

Installation

- We have created a zip file which contains program codes for each task along with another program that can make a sub process call to each of the above-mentioned programs.
- Install NumPy: pip install numpy
- Install Pandas: pip install pandas
- Install Matplotlib: python -mpip install matplotlib

Execution Instructions

- Tasks:
    - Command: python tasks.py
    - Enter choice of task: 0, 1, 2, 3, 4, 5 or 6
- Pre-processing Task:
- Task 1: Enter value of k: 5
- Task 2: Input the algo number
    1. Bfs Clustering
    2. K spanning tree

    1

    Input the number of clusters needed: 6
- Task 3: Enter value of k: 5
- Task 4: Enter value of k: 7, image_id1, image_id2, image_id3
- Task 5:
    - Number of layers: 7
      Number of hashes: 2
    - Image id: <image_id1>
      T: 4
- Task 6: Enter value of 'k' for k-Nearest Neighbours: 7

Conclusion

In this project, we used different kinds of information retrieval algorithms such as Page rank, graph clustering and locality sensitive hashing using approximate index structures. We saw the applications of general algorithms such as page rank which consider that any image would be visited with the same probability, as well as personalized page rank which helps to obtain images similar to a given set of images, where we assume that some images have a higher probability of being visited than others. We also explored the use of clustering with respect to graph clustering algorithms, which help us divide our database of images into coherent groups. Usage of LSH showed that we can decrease time complexity of k-Nearest Neighbors from linear to sublinear and therefore make it more efficient.

Bibliography

- Data Management for Multimedia Retrieval, K. Selçuk Candan and Maria Luisa Sapino, Cambridge University Press, 2010.
- S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems 30: 107-117, 1998.
- J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. ACM SIGKDD, 653-658, August 2004.
- Alexandr Andoni and Piotr Indyk. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. Communications of the ACM, vol. 51, no. 1, pp. 117-122, 2008.

Appendix

- Pre-processing task has been implemented by Aniket
- Task 1 has been implemented by Aniket
- Task 2 has been implemented by Siddhesh
- Task 3 has been implemented by Siddharth and Aditya
- Task 4 has been implemented by Aditya, Aniket and Siddharth
- Task 5 has been implemented by Vasisht
- Task 6 has been implemented by Shatrujit