

CSE 515 Multimedia and Web Databases

Page Ranking, LSH and classification of
Images in a Dataset

PROJECT REPORT

Abstract

The abstract of this phase of the project is to understand and experiment with graph analysis, clustering, indexing and classification. Task 1 concentrates on creating a image-image similarity graph and task 3 use this graph as input to page ranking algorithm to find significant nodes. Also given a seed image ids, personalised page ranking based on those ids also has to be done. Task 2 concentrates on applying clustering to the image-image similarity graph and creates clusters. Task 5 concentrates on creating an in-memory index structure and queries the t similar image given a image id. Task 6 does classification, supervised learning based on the labelled input data set.

The entities are represented as textual descriptors and visual descriptors. Visual Descriptors are represented in the form of various visual descriptors like Global Color Naming Histogram, Global Histogram of Oriented Gradients, Global Color Moments on HSV Color Space.

Keywords

PCA, Vector, Euclidean, Frobenius norm, Cosine Similarity, TF(term frequency), TF-IDF(term frequency–inverse document frequency), DF(document frequency), KNN, Spectral Clustering, K-means, Locality Sensitive Hashing(LSH), Page Rank(PR), Personalized Pagerank(PPR), Random Walks, Global Color Naming Histogram(CN), Global Histogram of Oriented Gradients(HOG), Global Color Moments on HSV Color Space(CM), Global Local Binary Patterns on gray scale(LBP), Global Statistics on Gray Level Run Length Matrix(GLRLM), sklearn, numpy, Norm, Scaling, Transformation, Frobenius Distance, Euclidean Distance.

Introduction

Terminology

TF – Term Frequency : The count of a term in a document

DF – Document Frequency : The number of documents which contains this term

TF-IDF - term frequency–inverse document frequency : how important a word is to a document

Global Color Naming Histogram (code CN - 11 values): maps colors to 11 universal color names: "black", "blue", "brown", "grey", "green", "orange", "pink", "purple", "red", "white", and "yellow" [v1];

Cosine Similarity: It is a measure of similarity between two vectors that calculates the cosine of angle between them

PCA: It is a algorithm that uses an orthogonal transformation to convert a data having possibly correlated features into a set of values of linearly independent features[5]

kNN Algorithm: It is a commonly used non-parametric and lazy classification algorithm. It classifies a given data by memorizing the entire data. Hence the name lazy classifier.

Spectral clustering: It is a clustering algorithm used on the similarity matrices to form specified number of clusters. It uses laplacian matrices for the calculation of the feature vectors.

Laplacian Matrix: It is a matrix representation of the graph data. It is defined as:

$$L = D - A$$

where D is a Degree matrix and A is a adjacency matrix of the graph data.

Degree Matrix: It is a diagonal matrix, which has information about the degree of each row which tells about the connectivity. It is used along with adjacency matrix to form the laplacian matrix.

K-Means Clustering: It will fit the data into k clusters where the inter cluster distances are maximum and intra cluster distances are minimum.

Frobenius norm: The Frobenius norm is the Euclidean norm on and comes from the Frobenius inner product on the space of all matrices.

PageRank algorithm: It is used to determine importance of a image or a webpage by giving it a rank.

Personalized PageRank algorithm: It is a modified version of PageRank algorithm. Here, the importance of a image or webpage is determined with respect to one or more given query points.

Normalized-cut partitioning algorithm: It is a graph partitioning algorithm which is used to create groups using minimum cuts.

Goal Description

The goal of this project is to reduce the dimensionality of the dataset and find k similar entities to a given entity. Different features of the entity will be provided in the form of visual descriptors and textual descriptors.

Task 1: Given a value k , This task is to form a Image-image similarity graph such that K edges are pointed to K most similar images, where K represents the no of outgoing edges. This similarity graph is used as the input for all the remaining tasks.

Task 2:

This task is a continuation of task 1. In this our goal is to create C number of clusters (where “ C ” is a user specified value) using 2 clustering algorithm techniques namely Spectral clustering and normalized cut partitioning algorithm. Then after forming the clusters using these algorithms we visualize the clusters using the web browser.

Task 3:

Given a user specified k value (No. of dominant images) and the image-image similarity graph obtained from task 1, this task is to identify and visualize K most dominant images by using Pagerank algorithm.

Task 4:

Given 3 user specified image ids, a user specified k value, and the image-image graph obtained from task 1, this task is to use Personalized PageRank(PPR) algorithm and then identify and visualize K most relevant images

Task 5:

Given the number of layers, L , the number of hashes per layer, k and a set of vectors, the task is to implement a Locality Sensitive Hashing (LSH) tool, for a and similarity/distance function and creates an in-memory index structure containing the given set of vectors

The task is to implement similar image search using the previously created index structure. The data with which the index structure is to be created is to be made from a combined visual model function with at least 256 features. With a given image and t as input, t most similar images has to be visualised. Also the numbers of unique and overall number of images considered across layers also has to be found.

Task 6

Given a file containing a set of image/label pairs and a value k , associate a label to the rest of the images in the database. In addition to this, This task has also been implemented without providing any input k value. The algorithms used for this task are K- nearest neighbor based algorithm and Personalised Page Ranking algorithm(PPR).

Assumptions

1. Each entity have all the textual descriptors like TF, DF, TF-IDF present for their corresponding terms.
2. Every images should have all the visual descriptors in terms of CM, CM3x3, CN3x3, CN, CSD, GLRLM, GLRLM3x3, LBP, HOG present as a csv file.
3. Dataset is in the same format.
4. Input is provided as expected. While executing each task, sample input is specified which has to be followed.
5. Data is assumed to be true and accurate.
6. The teleportation factor in Pagerank and Personalised Pagerank algorithm is considered to be 0.15.

Proposed Solution

The following section provides the solution for the problem and the reasoning for the solution

Preprocessing:

The data is preprocessed and loaded into the mongoDb before executing any tasks.

The data for this phase is obtained from 1 xml file (locations), 3 space separated text files containing visual descriptors for users (devset_textTermsPerUser.txt), images (devset_textTermsPerImage.txt) and locations (devset_textTermsPerPOI.txt) and 300 comma separated files containing visual descriptors for 30 locations, each file contains data for 1 location k and 1 visual descriptor of which there are 10. The data is processed and loaded on to MongoDB.

Textual Descriptors (3 documents – users, images and locations) [{id:"10117222@N04", desc:[{term:"1",TF:"1",DF:"53",TF-IDF:" 0.018867924528301886"}.....]}.....]

Visual Descriptors (10 documents – one for each location)

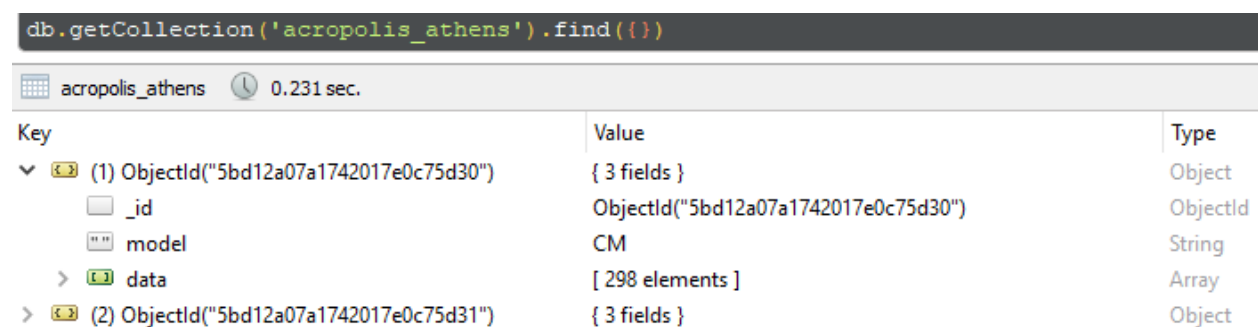
[{model:"CM",data:[image_id,data1.....]}.....],{model:"CM3x3",data:[[]]}.....]

Devset_topics.xml has to be read and all its field like location id, title are stored in locations collections in database.

Locations collections in database has the following format:

```
{
  "_id" : ObjectId("5bd129f2a1742017e0c737be"),
  "id" : "1",
  "title" : "angel_of_the_north",
  "latitude" : "54.9141",
  "longitude" : "-1.58949",
  "wiki" : "http://en.wikipedia.org/wiki/Angel_of_the_North"
}
```

All the visual descriptors are read from csv with filename from title field of locations collection in database and inserted in collections named "title field" from location collections. These collections in database will contain the visual descriptors for all the models.



In the above figure, each document represents visual descriptors of all the images the "acropolis_athens" place for respective visual models like 'CM', 'CM3x3', 'CN', 'CN3x3', 'CSD', 'GLRLM', 'GLRLM3x3', 'HOG', 'LBP', 'LBP3x3'.

For the we read all the locations xml files from xml folder, and read each content to get the location from file name, image id from id, userid from user id and terms count from the tags and we store this data in LIU_commonterms collections. The content of each document is as belows:

```
{
  "_id" : ObjectId("5bd4fee6a1742028dc0aea25"),
  "location" : "altes_museum",
  "image" : "11367738404",
  "user" : "37601286@N06",
  "terms" : 5
}
```

Preprocessing to generate Image-Image similarity graph:

1. As the tasks are dependent on task 1. During data loading, a img-img similarity graph is generated and is stored.
2. The procedure is as follows:
 - a. **Preprocessing:** devset_topics.xml is read and the data is stored in mongodb as collection locations. Then each location name is associated with a cs for each model that is angel_of_north location will have the model values under the name angel_of_north_CN.csv, angel_of_north_CM.csv. Each of these files is read and the values (image ids and the corresponding visual descriptor model) is stored in another collection under the name of the location and the documents with two fields 1. model name and 2. values for that model. So, each location will have a collection of it's own and each collection will have a document for it's model.
 - b. After the preprocessing is done, then we represent our matrix in a way that can be decomposed using the decomposition algorithms provided by sklearn library.
 - c. **Matrix representation:** As shown in the figure below
 - i. Here we create a 2d with rows - images and columns as visual descriptor values.
 - ii. We first get all the locations from the locations collections and loop through it.
 1. In this loop, we iterate over the 10 visual descriptor models mentioned above corresponding to each location.
 - a. Then we get the data for each model in each location from the file location_name_model_name.csv
 - b. The first column in the field or document is of the format image id, value, value, value,
 - c. So we strip the first column from all the documents and these are the image ids.
 - d. We maintain a dictionary of all the image ids.

- e. Now the values are appended across for each model using `np.concatenate`
 - f. so each row consists of the values of of a single image id of a location across all the models.
2. Once, we have all the images of one location, we keep track of the start and end of these images with respect to their index locations in a dictionary of locations.
 3. Then continue to the next location.
- iii. Once we have this 2D matrix for all the locations - their images and their values for all models, we pass it to sklearn decompositions and reduce to 400 dimensions using PCA.
 - iv. Here we have decided to go with PCA and 400 dimensions after trying out several combinations. As this preserves 99.5% of variance and helps in faster computation.
 - v. With the decomposed data, euclidean distances between each image to every other image is calculated.
 - vi. These distances are then sorted in ascending order for every image and is stored in mongodb in 'image_image_vis' collection for easy processing for further tasks.
 - vii. This data typically contains distance from one image to all the images in sorted order.

Matrix representation:

		Model 1 CM			Model 2 CN		
		val1	val2	...	val1	val2	...
Location 1	img1	0.04	0.03	...	0.1	0.008	...
	img2						
	.						
	.						
Location 2	img1						
	img2						
	img3						
	.						

Algorithms:

This section talks about how the various algorithms have been implemented and what inputs these expect and outputs these return.

PCA[5]:

1. This is implemented in the file `decomposition_algorithms.py`
2. Library Used: `sklearn.decomposition`
3. Input: `k` - number of latent semantics needed and a 2D array with objects as rows and features as columns, features - all the terms.
4. We pass the above inputs to `fit_transform` function from the PCA method of the `sklearn` library.
5. Output: This returns 2 things:
 - a. A matrix with object number of rows and `k` number of columns. A fitted matrix of the data being passed.
 - b. Loading scores that's a 2D array called `loading_scores` of `k` (latent semantics) number of rows and features number of columns. So this will tell how the features contributed against latent semantics. The eigen values will be `explained_variance_ratio_` and absolute of `model.components_[i]` is eigenvectors. Multiplying these both gives the loading scores. And then the loading scores are sorted in descending order.

Sparse PCA implementation[6]:

1. Since the `sklearn` library's PCA doesn't accept the sparse representation, we have our own implementation of PCA to process the sparse representation.
2. First we compute the covariance matrix[8] and then again compress the covariance matrix using `csr_matrix`.
3. We use `linalg` from `scipy.sparse` to compute the eigenvalues and eigenvectors for the sparse representation of covariance matrix.
4. Then we multiply the data(object * feature) with the eigenvectors to get the `k` latent semantics reduced data.

K Means Clustering[7]:

Input: Number of clusters to be formed and dataset

1. Based on the `K` value, initialize random `k` centroid points with values ranging between highest and lowest of the given data values.
2. Initialise the (collection of centroid points)`C_previous` with zeros of numpy array with same dimension as input data, which will later be used previous centroid values.

3. Loop through the following points till the change between new centroid and old centroid becomes zero or in another term converges.
 - a. CLASSIFY STEP : For each row of data calculate the distance between it and all the centroids and store the index of the centroid for which there is minimal distance in clusters variable.
 - b. Copy the current centroids to C_previous variable which is to store previous centroid value.
 - c. RECENTER STEP : For each centroid of (total length = k) Find the mean of points belonging to the same cluster centroid (we got from classify step). If we find any cluster to have clusters length of zero, later after this step, we will split the largest cluster into two.

Thus eventually after the clusters centroids converge return the clusters which associates each each data point of input dataset to a certain cluster where 0 represents cluster 0. (zero-based indexing is followed)

Laplacian Matrix Formulation[8]:

This is implemented in file clustering_algorithms.py

1. An adjacency matrix using the image-image similarity graph obtained from task 1 is calculated.
2. Then on the adjacency matrix, we initialize a laplacian matrix using sparse function and then using the normalized laplacian we calculate a laplacian matrix.

$$L^{\text{sym}} := D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} ,$$

The elements of L^{sym} are given by

$$L^{\text{sym}}_{i,j} := \begin{cases} 1 & \text{if } i = j \text{ and } \deg(v_i) \neq 0 \\ -\frac{1}{\sqrt{\deg(v_i) \deg(v_j)}} & \text{if } i \neq j \text{ and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise.} \end{cases}$$

3. Where L is the laplacian matrix.
4. This laplacian matrix is used in both Spectral clustering and Normalized cut partitioning algorithms for formulating the eigenvalues and eigenvectors.

Spectral Clustering[9]:

This is implemented in file clustering_algorithms.py

Input: Number of clusters to be formed and the graph dataset.

We pass the above inputs to spectral_clustering function of class Clustering_Algorithms

1. From the given dataset, we first calculate its adjacency matrix.
2. From the adjacency matrix, it builds a Graph normalized laplacian matrix
3. It then calculates the eigenvalues and eigenvectors from laplacian matrix and degree matrix by using sparse.linalg function scipy package.
4. Then these first k eigenvectors from laplacian matrix act as the feature vectors, which will be the input for the k-means.
5. Then the k-means clustering is applied on this data to get C number of clusters.

Normalized-cut graph partitioning Algorithm [11] :

This is implemented in file clustering_algorithms.py

Input: Number of clusters to be formed and the graph dataset.

We pass the above inputs to normalised_cut function of class Clustering_Algorithms

1. From the inputted graph data set calculate the laplacian matrix.
2. Later, find the eigenvalues and eigenvectors of the laplacian matrix
3. Then find the 2nd smallest eigenvalue in the array of eigenvalues obtained and take its corresponding eigenvector
4. Now, arrange such that all the image ids corresponding to the positive components of the considered eigen vector into one cluster and all the negative values in the other cluster.
5. Repeat the process for the largest cluster among all the clusters formed. Until the user inputted number of clusters value is reached.

K- Nearest neighbor based classification algorithm(kNN)[4]:

This algorithm is implemented in file classification_algorithms.py

Input: A value k (No. of nearest neighbors) and the dataset

We pass the above inputs to find_k_nearest_friends function of class KNN

1. Calculate the similarity measure on the given dataset, using Euclidean distance as the distance measure.
2. And from these similarity measures, it would give k-nearest neighbours.

Output: It would give us k nearest neighbours as the output.

PageRank[1]:

Input: The dataset and the number of dominant images to be returned.

1. The transition probability matrix is built for the given dataset with the assumption that a random surfer will visit the connected nodes with a probability of $1-a$ or visit a random node with a probability of a , where a is the teleportation factor 0.15

- Initially, all the probabilities of the outgoing edges from a particular node are set to $1/|\text{outgoing_edges}|$.
- if there are no outgoing edges then from a node, another node is selected with the probability $1/|\text{number_of_nodes}|$
- The random walk over the inputted graph can be represented with a transition matrix given by

$$T = \beta M + (1 - \beta) \left[\frac{1}{N} \right]_{N \times N},$$

where beta is (1-a) and the first part that is beta*M represents the random walk

the second part represents that is (1-beta)*((1/N) matrix) represents the transport or the restart

N is the no of nodes of the graph

M represents the N*N where

$$M[i, j] = \begin{cases} \frac{1}{|\text{out}(p_i)|}, & \text{if there is an edge from } p_i \text{ to } p_j, \\ \frac{1}{N}, & \text{if } |\text{out}(p_i)| = 0, \\ 0, & \text{if } |\text{out}(p_i)| \neq 0 \text{ but there is no edge from } p_i \text{ to } p_j. \end{cases}$$

The pageranks are obtained by finding the left principal eigenvector of the transition probability matrix. This is done using the power iteration method till 500 iterations or until convergence whichever occurs first.

Personalized PageRank[2]:

Input: The data set and 3 user specified image Ids and number of most relevant images to be returned

- The input supplied is the adjacency matrix of the image - image graph and it should be column normalised.
 - Here the image ids inputted by the user represent the query points for the algorithm.
 - We start by doing a random walk from the query point and computing the probability vector.
 - consider a column vector ,called restart vector, with its elements initialized to zero except for the entry of the query point. Initialise the query point entry to 1.
 - Initially the probability vector is initialised to be same as the restart vector
 - In the transition matrix calculation, the second part ensures that it is done using random walk with restart. As we are using the restart vector which has the probabilities set to $1/(\text{num_query_imgs})$ for the given the query_imgs.
- The Transition vector is calculated as follows:

$$\vec{u}_q = (1 - c)A\vec{u}_q + c\vec{v}_q$$

here U_q = Transition vector.

V_q = Restart vector.

the above process is continued by coming back to query point, until the probability vector is converged.

Here the **random walk restart** is explained as follows:

The "random walk with restarts" operates as follows: to compute the affinity of node "B" for node "A", consider a random walker that starts from node "A". The random walker chooses randomly among the available edges every time, except that, before he makes a choice, with probability c , he goes back to node "A" (restart). Let p denote the probability that our random walker will find himself at node "B". Then, p is the affinity of "B" with respect to "A".

sorted_list

This will maintain a sorted list of size k by using insertion sort. This is used in respective tasks to find the top k lowest distances.

The following tasks use the above algorithms in each of the tasks and the preprocessing of the data is done before executing any tasks.

Task 1:

Input: $k = 7$

Output: Image-Image similarity graph, such that for each image id there are k outgoing edges to the k most nearest related images.

Implementation:

1. After the data loading is done, we read the mongo collection 'img_img_vis' to get the img_img_graph
2. This data typically contains distance from one image to all the images in sorted order.
3. This is reduced to get the k nearest images for each image. That is the k outgoing edges to k most related images.
4. The output is then stored to file - img_img_graph.txt

Task 2a

Input: Image-image graph from task 1, $C = 6$ (where C is no. of clusters)

Output: Get C clusters from the dataset using spectral clustering and visualize them on the web browser.

Implementation:

It is implemented in clustering_algorithms.py

1. As explained in the Spectral clustering mentioned as a part of algorithms section.
2. An adjacency matrix using the image-image similarity graph obtained from task 1 is calculated.
3. Then we calculate the laplacian matrix from adjacency matrix as specified in the laplacian matrix formulation in algorithm section.
4. Then from the laplacian matrix we find C number of eigenvalues and eigenvectors using sparse.linalg.eigs function from scipy library.
5. These computed first C eigenvectors act as the feature vector for each each object.
6. Then these eigenvectors are passed through the K-means clustering algorithm which is described in the algorithm section of the report.

The diagram shows the objective function $J = \sum_{j=1}^k \sum_{i=1}^n \|x_i^{(j)} - c_j\|^2$ with several annotations: an arrow points from 'number of clusters' to k , from 'number of cases' to n , from 'case i ' to $x_i^{(j)}$, and from 'centroid for cluster j ' to c_j . A bracket under the distance term $\|x_i^{(j)} - c_j\|^2$ is labeled 'Distance function'. An arrow points from 'objective function' to J .

7. Then the K-means clustering gives C number of clusters as the output which we visualize using the web browser.

Task 2b

Input: Image-image graph from task 1, $C = 6$ (where C is no. of clusters)

Output: Get C clusters from the dataset using normalized cut partitioning algorithm and then visualize them on the web browser.

Implementation:

1. The image-image similarity graph generated in the task 1 is used and the adjacency matrix and degree matrix are created.
2. These adjacency matrix and degree matrix are used to create the laplacian matrix as explained in the task 2a and algorithms section.
3. Eigenvectors are then created using this laplacian matrix using scipy library.
4. Then from the list of eigenvectors, a eigenvector with the second smallest eigenvalue in the list is selected.
5. This eigenvector is used for bi partitioning the graph.
6. The eigenvalues with positive values in this vector from one cluster and the values with negative ones are stored as another cluster.
7. Then, if we need to split further into cluster, then the largest cluster is picked and is splitted.
8. This process is repeated recursively until we reach a point where we get the C number of clusters which is the user input.

Task 3

Input : Image-image graph from task 1, K = 7

Output: Get k most dominant image ids and then visualize them on the web browser.

Implementation:

It is implemented in **page_rank_algorithms.py**

1. As explained in the page rank algorithm mentioned as part of algorithms section.
2. A adjacency matrix using the image -image similarity graph for the k most related images are calculated and then normalised.
3. We have assumed alpha (the probability the random surfer follows outgoing edge or jumps to a new page if no outgoing edge) as 0.85 which gives the most optimized results.
4. Then a transition matrix is calculated by the formula

$$T = \beta M + (1 - \beta) \left[\frac{1}{N} \right]_{N \times N}$$

5. Here, M is adjacency matrix explained in step 2 and beta is alpha as explained in step 3.
6. Then another vector which contains the probability scores of a reaching an image on a random walk which is 1/n for all the images in our case. Hence the 1d array with 1/n as values.
7. With the Transition matrix T, and the probability vector. These two are multiplied in each iteration and compared if they converge or not before every iteration. This is called power iteration method.

8. After an infinite runs or when the probability matrix doesn't change much whichever is earlier, we take the components of the first eigenvector.
9. This components represent the pagerank scores of each image.
10. These are then sorted in descending order to get the K highest images which are K most dominant images by the page rank algorithm.

Task 4

Input: Image-image graph from task 1, k =6, image ids = "2976144","3172496917","2614355710"

Output: Displays top 5 relevant images for the image ids specified by the user.

Implementation:

It is implemented in `page_rank_algorithms.py`

1. As explained in the personalized pagerank algorithm mentioned as part of algorithms section.
2. Adjacency matrix using the image -image similarity graph for the k most related images is calculated and then it is row normalised(sum of all the elements in one row equals to one) to bring everything to the same scale.
3. This matrix is then transposed to get a column normalised adjacency matrix according the algorithm.
4. Then a restart vector is initialised to zero.
5. The entry corresponding to the query points that is the given input images is initialised to 1/no_of_query_imgs. This takes into consideration of personalization. This makes sure "random walk with restart" is done. This is explained as part of PPR in algorithms section.
6. The damping factor here is assumed to be 0.15 because of the optimal results given by it.
7. Then a transition matrix is calculated by the formula

$$\vec{u}_q = (1 - c)\mathbf{A}\vec{u}_q + c\vec{v}_q$$

8. Here, U_q = Probability vector; V_q = Restart vector; c = damping factor.
9. The initial probability vector is initialised to be same as the restart vector
10. With the Transition matrix T , and the probability vector. The above formula is compared to the transition matrix generated from the previous iteration to check the convergence.
11. The indefinite run stops if they converge. This is power iteration method. In our case, we did 500 iterations.
12. After we stop the iteration process, we take the components of the first eigenvector.
13. This components represent the pagerank scores of each image.
14. These are then sorted in descending order to get the K highest images which are K most dominant images by the random walk with restart - personalized pagerank algorithm.

Task 5a and 5b

5a) Input: l(no of layers), k(no of hash functions) and input dataset

Output: creates in-memory index structure(LSH)

Implementation:

1. Choose L functions g_j , $j = 1, \dots, L$, by setting $g_j = (h_{1,j}, h_{2,j}, \dots, h_{k,j})$, where $h_{1,j}, \dots, h_{k,j}$ are chosen at random from the LSH family H.
2. LSH family used is **euclidean hash family**[8] where v is the random projection vector and x is the input data point and b is the value $[0, w)$ and here b is a random variable from uniform distribution and w is the length of each quantization bucket. each components in the random projection vector is from a p-stable distribution where here in this task, gaussian distribution has been used $N(0,1)$. Store this chosen b , w values as this is part of the in-memory index structure. The length of each quantization bucket has been chosen by finding the closest distance between random images and taking 90% of the average of those closest distances.

$$h_{x,b}(v) = \left\lfloor \frac{v \cdot x + b}{w} \right\rfloor$$

3. Construct L hash tables, where, for each $j = 1, \dots, L$, the j th hash table contains the dataset points hashed using the function g_j . g_j is the combination of k hash function. g_j function will give a combined hash key by concatenating all the keys by applying individual hash function of that particular j th layer to the data points.
4. Query algorithm for a query point q :
 1. For each $j = 1, 2, \dots, L$
 - a. Retrieve the points from the bucket $g_j(q)$ in the j th hash table.
 - b. For each of the retrieved point, compute the distance from q to i .
5. Print the total number of images considered across the layers and also the unique images considered by converting the list to a set and doing length of the set.

Inference

On increasing the number of layers, the number of considered images across layers for a given query point increases and decreasing the bin length restricts the number of points falling in the same bucket and thus might introduce misses but avoids false positives. increasing the k value also does the same as decreasing the bin length[10]. No of layers can be chosen by picking

$L = \frac{\log_{1-p_1^k} \delta}{\log_{1-p_1^k} \delta}$ so that $(1 - p_1^k)^L \leq \delta$ [8], then, any object within range r is found in these buckets with probability at least $1 - \delta$ where δ is failure probability, p_k is probability of collision for close points.

5b)

Input: image id and t

Output: visualises most t similar images and numbers of unique and overall number of images considered

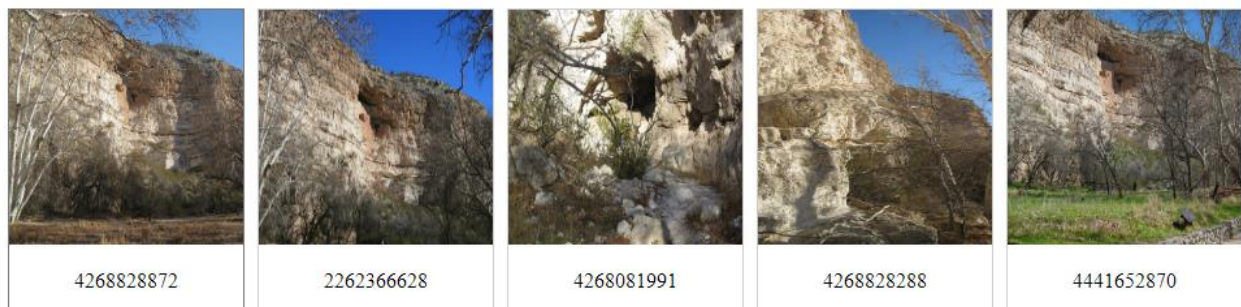
Implementation:

1. Create a combined visual model function by concatenating different visual models namely 'CM', 'CM3x3', 'CN', 'CN3x3', 'CSD', 'GLRLM', 'GLRLM3x3', 'HOG', 'LBP', 'LBP3x3' by placing each model's value side by side. All models has been combined since each of the model points out a peculiar feature of the image for example color by CM model, texture by LBP model and as such.
2. Normalise the data by applying a standardscaler from sklearn library to standardizing the individual features by removing the mean and scaling to unit variance.

$$x' = \frac{x - \bar{x}}{\sigma}$$

x bar is the mean of feature vector and sigma is standard deviation

3. Since we are going to use euclidean distance(hash family here) If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.
4. Apply PCA on the by restricting the dimensions to 400 which preserves about 98.8% of variance. PCA algorithm has been discussed in algorithm section.
5. This is the data that was used to build index structure in task 5a for a specified l and k. In this task, we get the query point from the user and t(nearest neighbour) and we will make use of the in memory index structure used in the task 5a.
6. query the lsh index structure with the given image id data and get all the considered points and sort them with the euclidean distance function to the query point and visualise the top t values. The first image is the query image and for t = 5, we got the following t similar images. Self similarity is considered.



Task 6a and 6b:

Input: filename and k (no of nearest neighbour to be retrieved)

Output: visualisation of labelled results for the rest of images not in given file

Implementation:

1. Create a combined visual model function by concatenating different visual models namely 'CM', 'CM3x3', 'CN', 'CN3x3', 'CSD', 'GLRLM', 'GLRLM3x3', 'HOG', 'LBP', 'LBP3x3' by placing each model's value side by side. All models has been combined since each of the model points out a peculiar feature of the image for example color by CM model, texture by LBP model and as such.

2. Normalise the data by applying a standardscaler from sklearn library to standardizing the individual features by removing the mean and scaling to unit variance.

a. \bar{x} is the mean of feature vector and σ is standard deviation

$$x' = \frac{x - \bar{x}}{\sigma}$$

3. Since we are going to use euclidean distance for neighbour calculation, If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.
4. The given file is read and panda library is used to read the file input and final two datasets namely training data and test data where training data consists of all images which are given in the file and labelled and test data is the rest of the images.
5. Pass the training data to the knn algorithm.
6. Since knn is an instance based learning, there is no training needed except storing the training data in memory.
7. For each of the rest of the images, pass the k value and each image row to be labelled to the knn algorithm.

$$\begin{aligned} d(\mathbf{p}, \mathbf{q}) &= d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} \\ &= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}. \end{aligned}$$

- 8.
9. KNN algorithm computes the distance between the test image and all the training data and only k smallest values(in increasing order) index of training data are alone retained by adopting min heap.
10. For the nearest neighbours, labelling the test data to be for which there is a majority of vote from the nearest neighbors. hashmap(dictionary in python) is used to save the votes from the nearest neighbors and choosing the (key)target for which there is max value.
11. Then the results are visualised by creating a html file referencing the images for each label.
12. A weighted knn has also been in case if no k value is provided as input where instead of giving whole 1 point to the neighbor, inverse distance weighting could be used, where each point has a weight equal to the inverse of its distance(euclidean) to the point to be classified. This means

that neighboring points have a higher vote than the farther points. On increasing the value of k , it is quite obvious that the accuracy will improve. Hence k is set to be n (no of data points of labelled dataset) itself.

Task 6b:

Input: filename of the file containing a set of image/label pairs

Output: visualisation of labelled results for the rest of images not in given file

Implementation:

1. **Preprocessing:** The image/label pairs are then converted into a dictionary such that key is the label name and value is array of image ids for faster execution.
2. Then, category wise the image/label pairs are passed to personalised pagerank algorithm. That is, for each label, the images are grouped together as one category and are passed to PPR(phase 4) which ranks with personalization into consideration.
3. These images are passed with a parameter not to sort the calculated page ranks, That is the returned values are the page ranks based on this label for all the images.
4. The same is repeated for all the images based on the categories and their corresponding page ranks scores are calculated.
5. So, now we have page rank values for each image based on each category.
6. The image page rank scores are compared in all the categories and that label is given according to the highest page rank score in all categories.
7. Incase, if the page rank values match then the labels are appended and the image is labelled with multiple labels.

Interface Specification

Run "python main.py"

Menu Options:

1. Task 1
 - 2a. Task 2a
 - 2b. Task 2b
 3. Task 3
 4. Task 4
 - 5a. Task 5a
 - 5b. Task 5b RUN 5a before this to build index structure
 - 6a. Task 6a With k
 - 6aa Task 6aa without k
 - 6b. Task 6b
 8. Load data
 0. Quit
- Enter option: 2b
- TASK 2b - Normalised cut partitioning
- e.g: 5

Input: 6

Sample input execution of each task

Task 1:

Input: User specified k value

Sample Input: k = 7

Output:

```
1288780397 = {1288780397; 7770574244; 5872576110; 2772036020; 36535126; 3313272320; 282276797; }
1674805035 = {1674805035; 7946872296; 3100190139; 88494678; 455623079; 3753285856; 5187088803; }
1674808621 = {1674808621; 96841974; 5038159283; 8207582957; 5213999239; 1685434866; 8556155993; }
1674810741 = {1674810741; 1674815731; 1675787466; 1675777022; 2205033827; 1674817147; 318613804; }
1674815731 = {1674815731; 1674810741; 2771193867; 318613804; 318615817; 318614132; 318613478; }
1674817147 = {1674817147; 11702609283; 2483652182; 8664737277; 2205033827; 318613478; 3184882031; }
1674819681 = {1674819681; 1675787466; 2059702220; 2355273611; 2999751672; 5456805866; 3844086536; }
1675673802 = {1675673802; 6865246830; 2873158508; 3762699767; 3185723606; 5229197365; 416554672; }
1675682028 = {1675682028; 3630226176; 3893218482; 6031579308; 6031024805; 4650277985; 6031023315; }
1675682410 = {1675682410; 6031579662; 7225670084; 416555138; 5229240089; 6031023315; 5989441049; }
1675777022 = {1675777022; 1675787466; 1674810741; 1674819681; 3398330080; 5213542757; 5213540379; }
1675787466 = {1675787466; 1674819681; 318613478; 318613804; 318614474; 1674810741; 9633836943; }
1675788826 = {1675788826; 1675789874; 2630640404; 2517318209; 12393320174; 318616047; 278548346; }
1675789874 = {1675789874; 1675788826; 8877205941; 318615210; 278548445; 4518837939; 318616047; }
1675790948 = {1675790948; 1675789874; 8877205941; 1675788826; 5200253528; 3763499450; 318616047; }
1831692999 = {1831692999; 3435157084; 10452444083; 8523163827; 1426403537; 1306855842; 4707775924; }
1831719961 = {1831719961; 5143870187; 6254887206; 2536030014; 6226636338; 3198731514; 1832534872; }
1831925575 = {1831925575; 8523171501; 1832755256; 1832534872; 1831927179; 2450162628; 7191423430; }
1831927179 = {1831927179; 7191423430; 2850330452; 2849497641; 6254874888; 2850330306; 4528155676; }
1832529568 = {1832529568; 1832533352; 7201351082; 6853242318; 3200056257; 3200096481; 4569995457; }
1832533352 = {1832533352; 1832529568; 6691855355; 4650183745; 9364960351; 3759169099; 534636363; }
1832534872 = {1832534872; 1832541240; 9838887075; 8524295618; 2356259091; 1831925575; 8523171501; }
1832536520 = {1832536520; 2987909153; 5951285958; 72855295; 7996665913; 2548794557; 9327125986; }
1832538318 = {1832538318; 3102381947; 3100192297; 10633459373; 11702190835; 2433676300; 3493366538; }
1832539790 = {1832539790; 5143870187; 2536030014; 2183308589; 298035843; 302435192; 6226636338; }
1832541240 = {1832541240; 1832534872; 2356259091; 6250108839; 7294703912; 9838887075; 5497974242; }
1832542742 = {1832542742; 2516162424; 2227175001; 2071272614; 2027020027; 218616601; 5187688210; }
```

Task 2a:

Input: User Specified C value and the image-Image graph from task 1

Sample Input: C=6, graph from task 1

Output:

1033

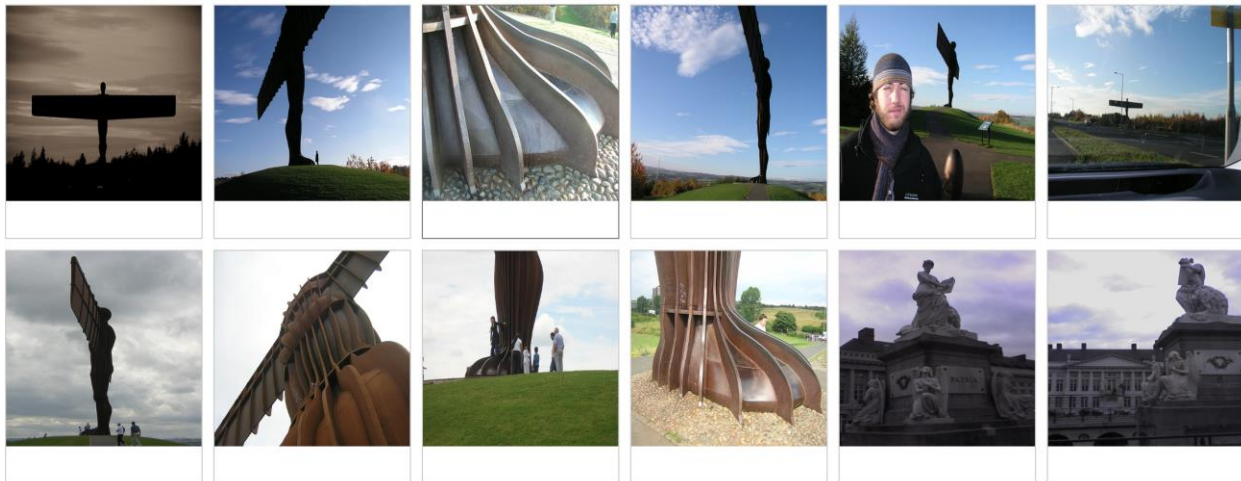
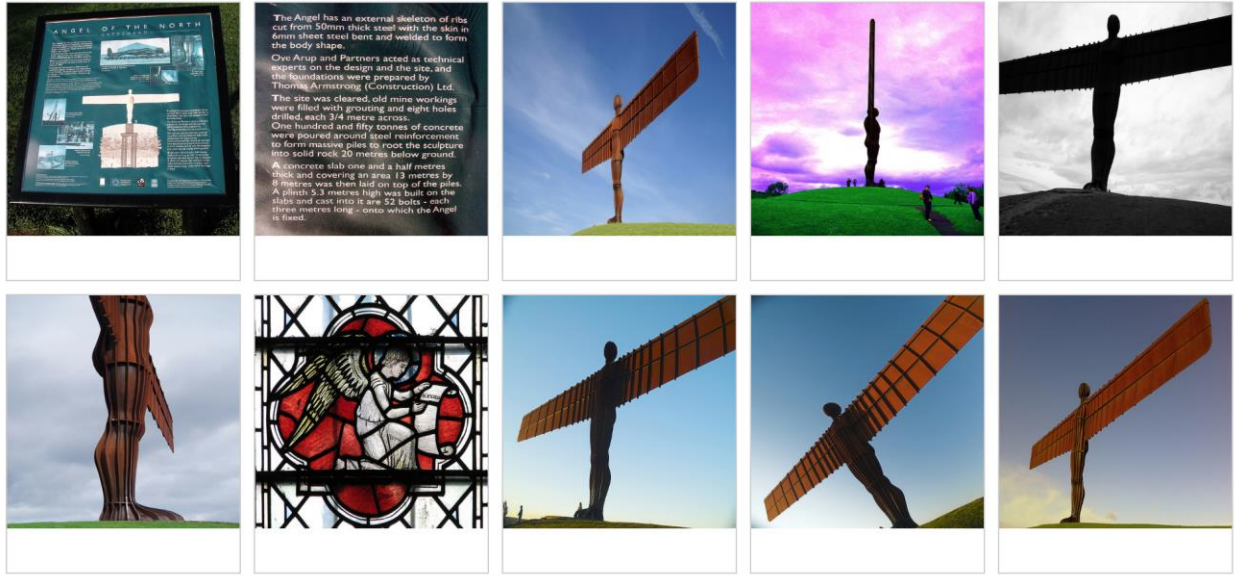
3055

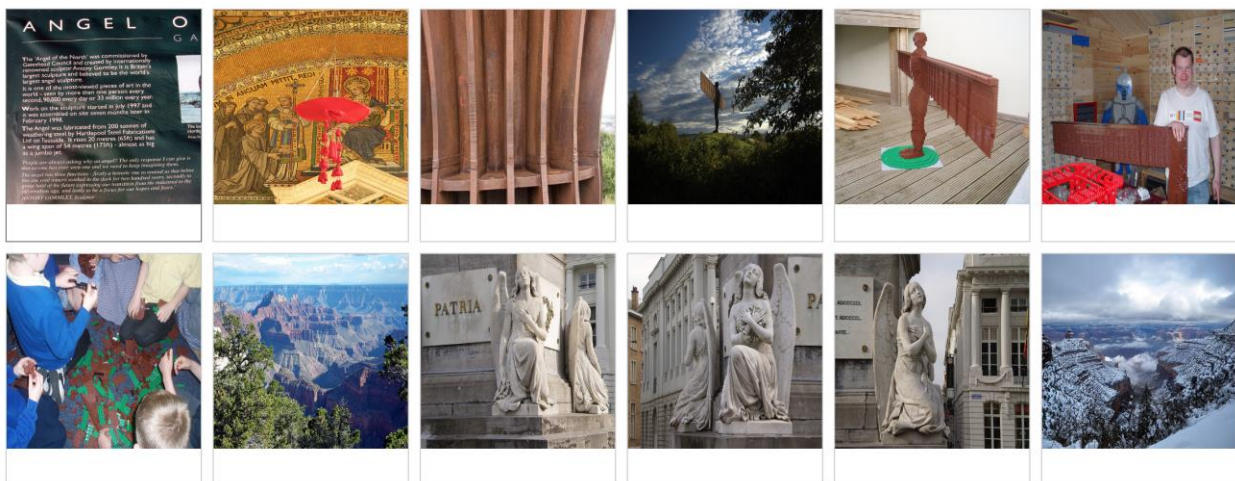
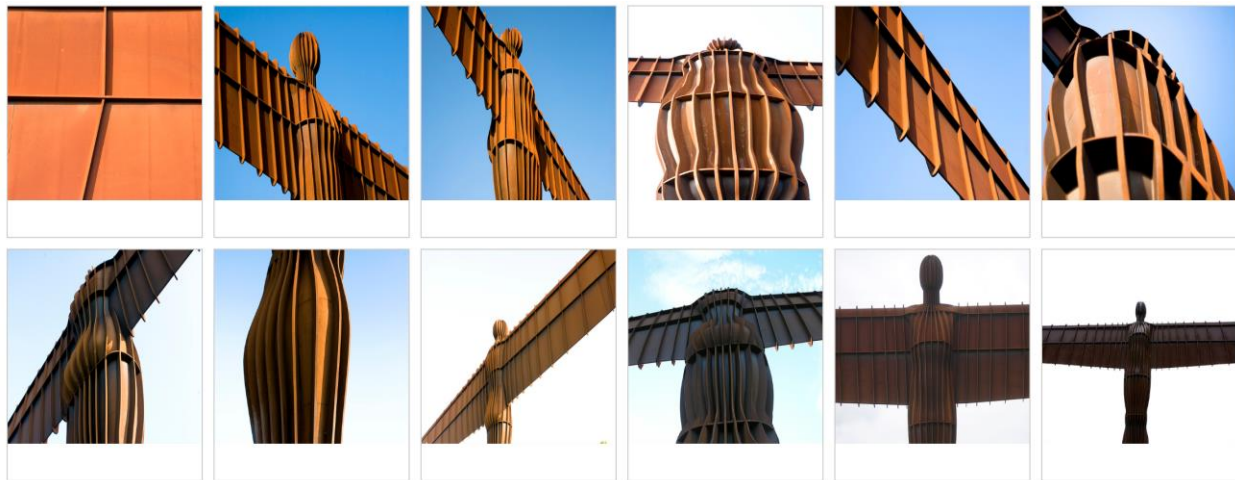
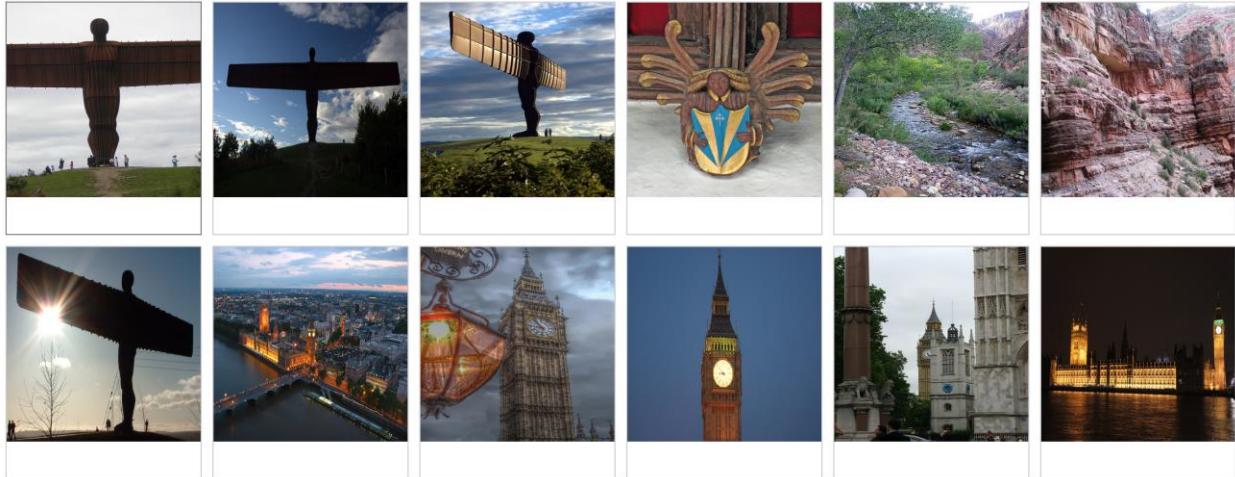
3192

392

113

1138



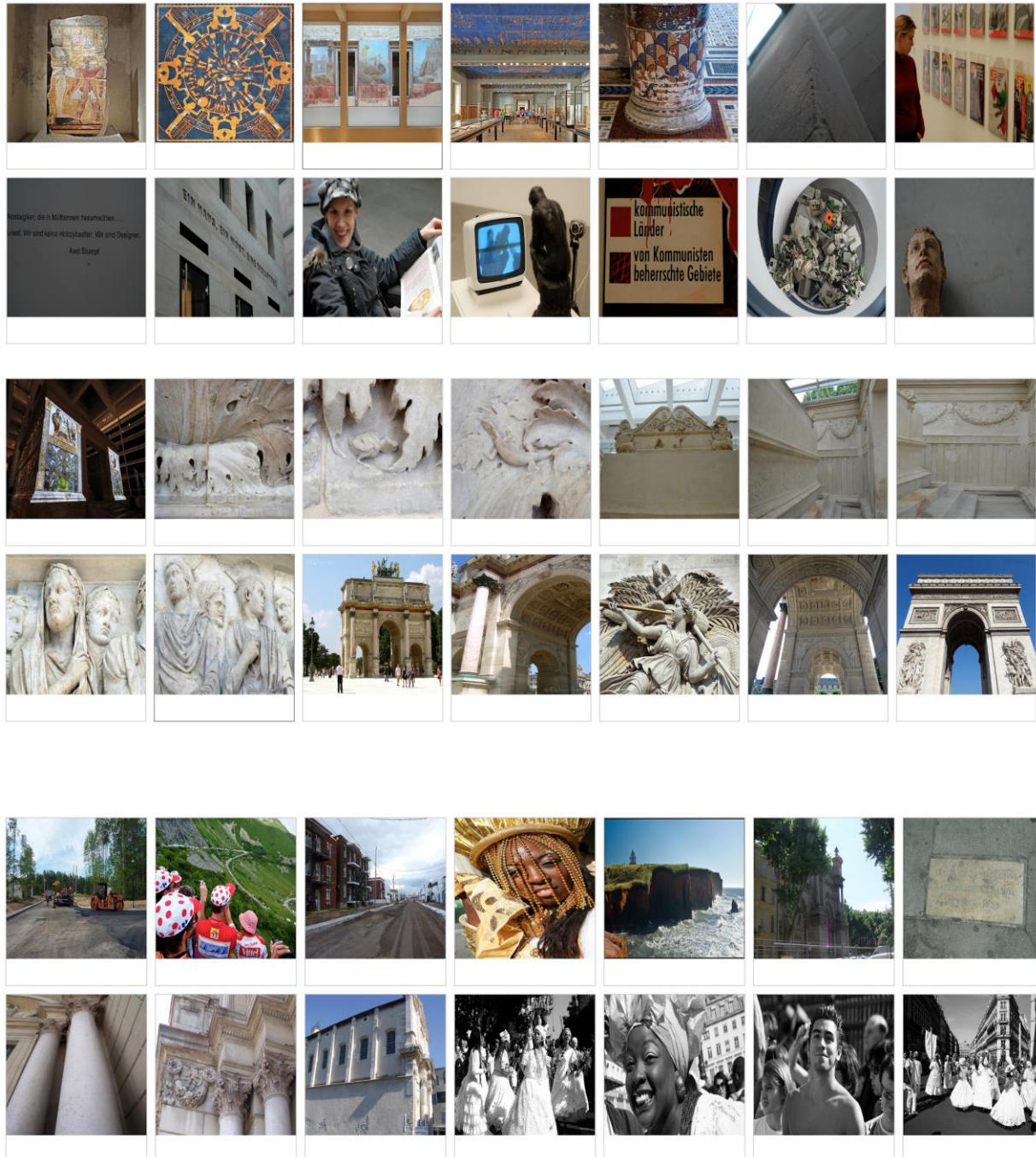


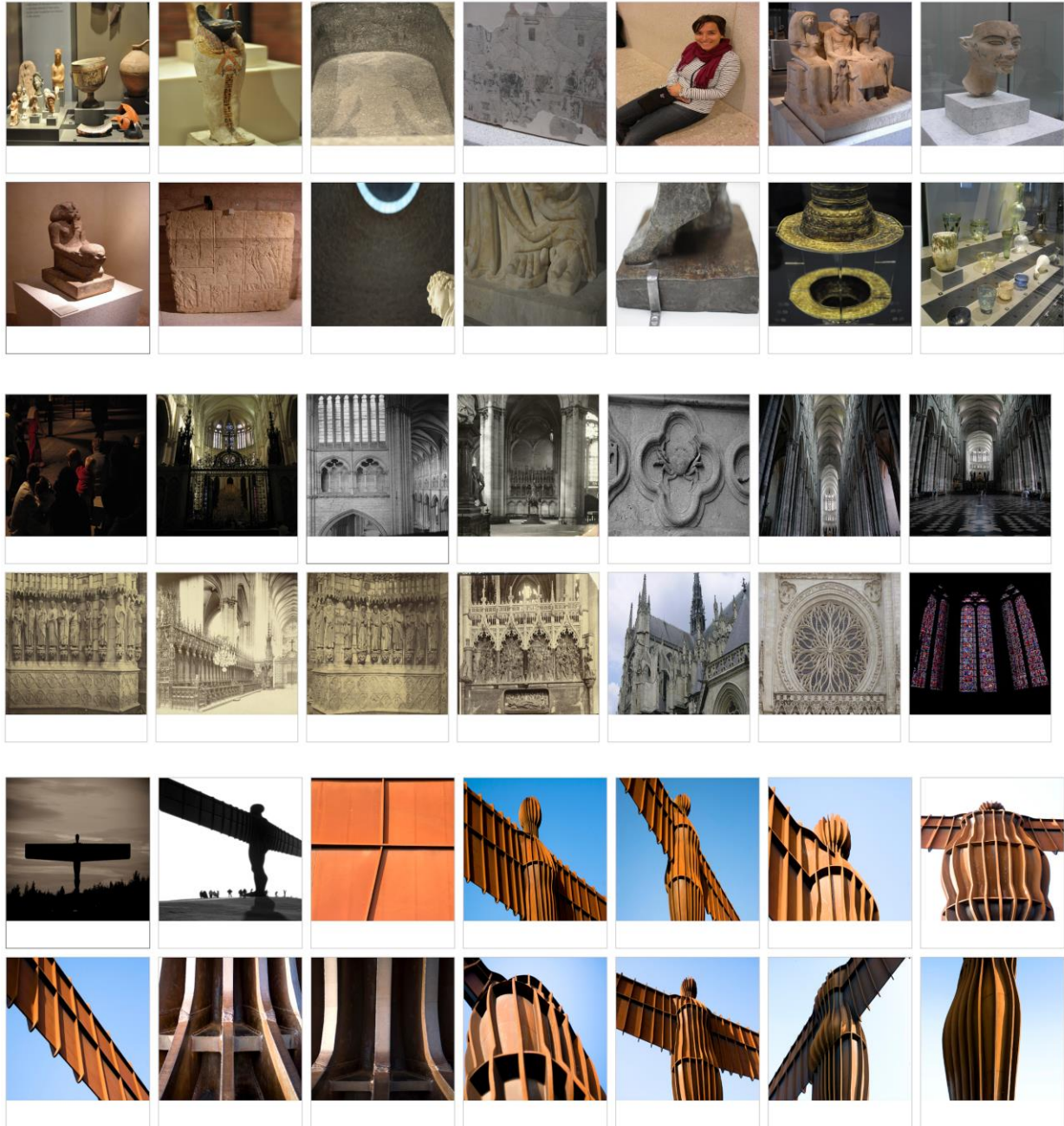
Task 2b:

Input: User Specified C value and the image-Image graph from task 1

Sample Input: 6

Output:





Task 3:

Input: User specified k value and Image-Image graph from task 1

Sample Input: K=7, graph from task 1

Output:



Task 4:

Input: User Specified k value and 3 image id's

Sample Input 1: K=6, Image ids = "2976144" , "3172496917" , "2614355710"

Output:



Sample Input 2: K=7, Image ids = "27483765" , "2492987710" , "487287905"

Output:



Task 5:

Input: Enter L(no of layers), K(no of hashes) "<L> <K>"

Sample Input: <L> <K> 3 30

Output: creates in memory index structure

Task 5b:

Input: Enter <image id> <t>

Sample input : Input: <Image Id> <t> 4268828872 5

Output:

With repetition Overall considered images= 1784

Total unique considered images= 1645









Task 6a:

Input: <filename> <k> task6.txt 6









Output :For Dome label:



For Stonemade label:

					
1674808621 stonemade	1674810741 stonemade	1674815731 stonemade	1674819681 stonemade	1675673802 stonemade	1675682410 stonemade

For Art label:

			
5834749503 art	5835305952 art	608304277 art	6492327911 art
			
12394260295 art	12394506145 art	12394604704 art	12394929383 art

Task 6aa:

Input Filename:

Sample Input: task6.txt

Sample Output:

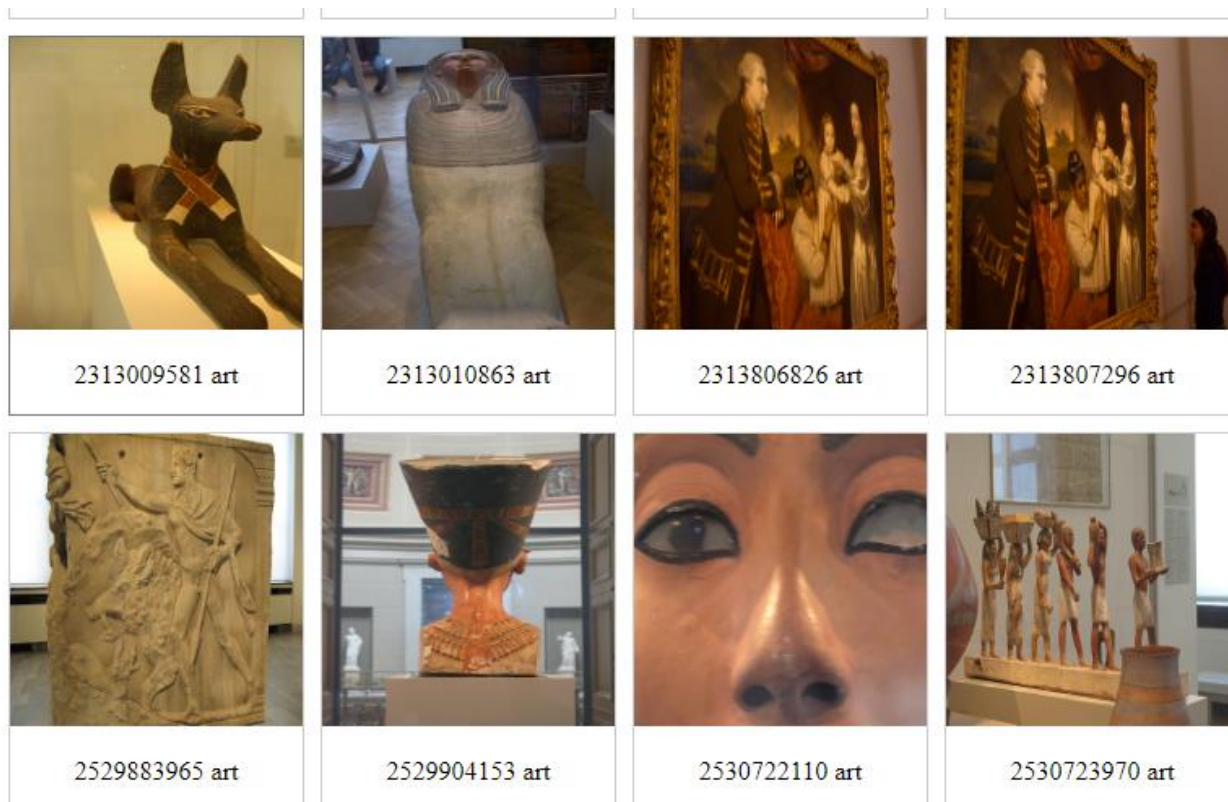
For dome label:



For stonemade label:



For art label:



Task 6b:

Input: filename

Sample Input: text6.txt

image	label
-------	-------

-------	--

3298433827	fort
------------	------

299114458	sculpture
-----------	-----------

948633075	sculpture
-----------	-----------

4815295122	sculpture
------------	-----------

5898734700	sculpture
------------	-----------

4027646409	fort
------------	------

1806444675	fort
------------	------

4501766904	fort
------------	------

6669397377	sculpture
------------	-----------

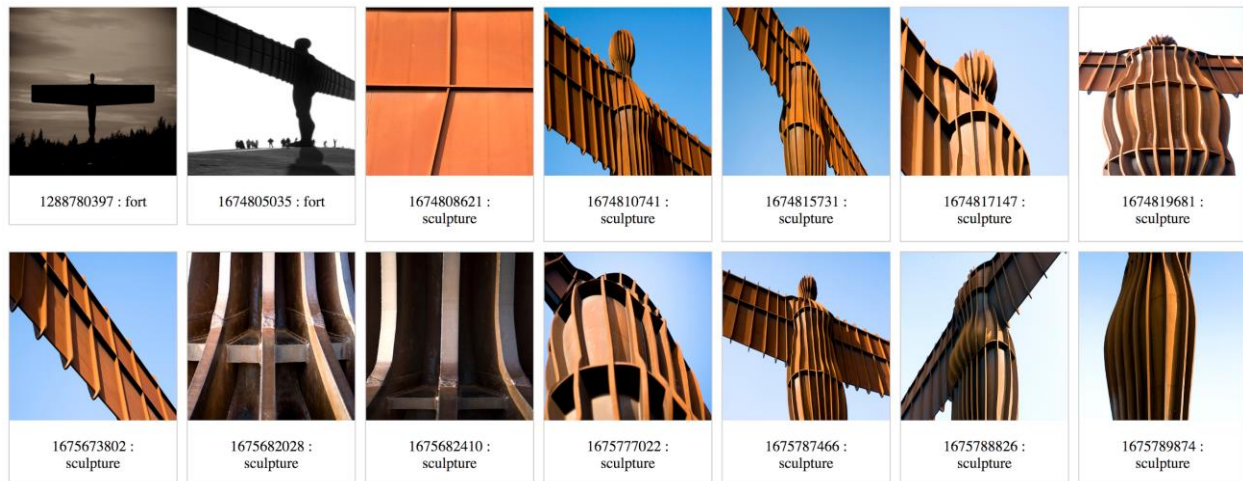
3630226176	sculpture
------------	-----------

3630226176	sculpture
------------	-----------

3779303606	sculpture
------------	-----------

4017014699	sculpture
------------	-----------

Output:



System requirements/installation and execution instructions

System Requirements

The hardware requirements are as follows:

- HDD: 20 GB
- RAM: > 8GB
- Processor: 2.0 GHz (intel i7 or better)

The software requirements are as follows:

- Python 3.7
- OS: Windows/Unix/Linux, preferably Ubuntu 14.04+
- Terminal, file editor

Execution Instructions:

- Install python
- Install MongoDB from <https://www.mongodb.com/download-center> and click next till its installed under localhost and default port 27017
- Go to code directory and type “pip install -r requirements.txt”
- Copy and unzip the desc.txt.zip, descvis.zip to the code directory from <http://skuld.cs.umass.edu/traces/mmsys/2015/paper-5>
- Copy the devset_topics.xml to the code directory from the above link as well
- Run python main.py

- Before running any tasks, make sure to run option 8 which loads all the data in the MongoDB.
- As all tasks are dependent on task 1, make sure to run task 1 after data loading is done before running any task.

Conclusions

We converted the dataset and created a image-image similarity graph and on this graph we applied partition, clustering, classification and indexing algorithms on the given dataset and visualized the results in web browser to view the k most dominant or nearest images in the dataset.

Related Work and Citations

1. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems 30: 107117, 1998
2. J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In KDD, pages 653658, 2004
3. https://en.wikipedia.org/wiki/Euclidean_distance
4. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
5. https://en.wikipedia.org/wiki/Principal_component_analysis
6. <https://stackoverflow.com/questions/19231268/correlation-coefficients-for-sparse-matrix-in-python>
7. https://en.wikipedia.org/wiki/K-means_clustering
8. https://en.wikipedia.org/wiki/Laplacian_matrix
9. Alexandr Andoni and Piotr Indyk. 2008. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Commun. ACM 51, 1 (January 2008), 117-122. DOI: <https://doi.org/10.1145/1327452.1327494>
10. Ravi B. Boppana. Eigenvalues and graph bisection: An average-case analysis. In IEEE Symposium on Foundations of Computer Science, pages 280–285, 1987.
11. Slaney, Malcolm & He Y. Lifshits, J. (2012). Optimal Parameters for Locality-Sensitive Hashing. Proceedings of the IEEE. 100. 10.1109/JPROC.2012.2193849.
12. <https://www.youtube.com/watch?v=DmJCM5FRWaM>

Appendix

Specific roles of the group members

Balachandar Sampath - Task 5 and 6a, Report

Ejaz Saifudeen - Task 2b, task 4, task 6b and Report

Shibani Hegde - Task 6b and Report

Amaresh Bingumalla - Task 1 and 4, Report

Srinivas Vallabhaneni - Task 2a and Report

Akhil Madamala - Task 2b and Report