# CSE 515: Project Phase 3

Aravamuthan Lakshminarayanan, Chirag Bhansali, Manthan Bharat Bhatt, Parvika Singhal, Yash Pande, Shubham Vipul Majmudar

***Abstract*—Identifying methods to perform query retrieval tasks in sub linear time has been a topic for research in Information Retrieval domain. This is one of the areas that we explore in this phase. To decrease search times, we attempt two methods - pruning irrelevant data during query time by making clusters and implementing a locality sensitive hashing based index structure. We further investigate Page Rank and Personalized Page Rank algorithms for most dominant images in our data set with (Personalized Page Rank) and without (Page Rank) user given context. In the last task, we look at image classification. We implement two classification techniques, k Nearest Neighbours and a Personalized Page Rank based classification algorithm.**

***Index Terms*—Image Search, Max-A-Min Clustering, K-Means Clustering, Page Rank, Personalized Page Rank, Locality Sensitive Hashing, K-Nearest Neighbour**

## I. Introduction

With the amount of digital information increasing, the need for sophisticated systems that can store and retrieve huge chunks of unstructured information is also increasing. According to Wikipedia, Information Retrieval is the science of searching for information in a document, searching for documents themselves, and also searching for meta data that describe data, and for databases of texts, images or sounds. One of the principal parameters to measure an IR system is query search speed. Various methods exist to improve query speeds, out of which we explore two - data pruning and indexing. In data pruning, a system neglects a segment of data (and does not perform a scan over it) that it believes is not similar to the query. This is usually done through clustering. Using the Flicker data-set [7], we create an image-image directed similarity graph and apply two clustering techniques on it - a slightly modified variant of Max-A-Min Clustering [13] and K-Means clustering on two different matrices. We visualize the results and confirm the grouping of similar images in same clusters. The other technique that we implement to optimize retrieval performance is indexing. We create a Locality Sensitive Hashing [10] based index structure [9] and assign each image to a list of hash-tables. We then use this index structure to perform k-NN search and observe the increase in speeds in comparison to previous phases. One other impediment that IR systems face is ranking documents after retrieval. In what order should the documents be displayed to the end-user? To answer this question, we consider two algorithms - Page Rank [11] and Personalized Page Rank [12]. Page Rank takes into account the importance of each document in the similarity graph. Personalized Page Rank takes into consideration a user's preferences to calculate importance. A document that is more important will be higher up the order. We implement both these algorithms and visualize our results on the Flicker data-set [7]. At last, we investigate supervised classification. For given set of images and labels associated with them, we classify other images in our data-set using K Nearest Neighbours and Personalized Page Rank.

This report is divided into nine sections. In section 2, we begin by presenting some terminologies that will help the reader comprehend further content to a greater degree. The third section describes problem statements of each task in terms of input parameters and the expected outputs. Section 4 lists all the assumptions that we have made while proposing our solution. This is followed by a detailed explanation of how we propose to solve each task in section 5. The system uses Command Line Interface to take inputs and display processed results. Section 6 will help you with operating this interface. In sections 7 and 8, we list steps to install the software and minimum system requirements, for both hardware and software, to run the software, respectively. We conclude the report in section 9 by mentioning our learnings and the inferences that we drew from the obtained results.

## II. Terminology

- **Random Walk With Restart**: To compute the importance/affinity of node "B" with respect to "A", we consider a random walker that starts from node "A", choosing randomly among the available edges every time. $u_a$ (B) denotes the steady state probability that our random walker will find himself at node "B" or in other words, the importance of "B" with respect to "A".
- **Community Detection**: In graph theory, partitioning a graph into group, clusters, or cohesive subgroups, such that nodes in a subgroup share common properties is called Community Detection.
- **Member Based Community Detection**: It is a class of Community Detection algorithms in which nodes are clustered based on specific characteristics. For example, in a party there can be two communities - male and female. Another example is that all nodes with degree 5 are grouped in one cluster (here node degree is the common characteristic)
- **Group Based Community Detection**: It is a class of Community Detection algorithms in which nodes are clustered based on their density of interaction. For example, in a data-set of universities, students of Arizona State University and students of University of Arizona may be grouped under different communities as the densities of interaction of students within their universities will be higher.
- **Spider Trap**: A group of Nodes in a graph is a spider trap if there are no links from within the group of nodes to outside the group of nodes

- **Dead Ends**: Nodes in a graph with no outgoing edges are dead ends for the random walk of nodes in the graph
- **p-Stable Distribution** Let $v \epsilon R^d$, and suppose $Z, X_1, ..., X_d$ are drawn i.i.d. from a distribution $D$. Then $D$ is p-stable if $(v, X) = ||v||_p Z$.
- **Fuzzy Classification**: Fuzzy classification is the process of grouping elements into a fuzzy set whose membership function is defined by the truth value of a fuzzy propositional function. A fuzzy propositional function is different from classics propositional functions in that the outcome of a fuzzy propositional function may not necessarily be true or false; it can be a value (determined probabilistically) in between.

## III. Goal Description

The goal of this phase is to experiment with graph analysis, clustering, indexing and classification. Following sub-sections describe in detail how these goals can be defined formally as problem statements.

### A. 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.

### B. 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.

### C. Task 3

Given an image-image graph, identify and visualize *k* most dominant images using Page Rank (PR) for a user supplied *k*

### D. 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*.

### E. Task 5

Implement a Locality Sensitive Hashing (LSH) tool, for a similarity/distance function of your choice, which takes as input the number of layers *L*, the number of hashes per layer *k*, and a set of vectors as input and creates an in-memory index structure containing the given set of vectors. Furthermore, 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*, visualize the *t* most similar images (also output the number of unique and overall number of images considered).

### F. Task 6

Implement a k-nearest neighbor based classification algorithm, and a PPR-based classification algorithm which takes 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.

## IV. Assumptions

- The reader of this report is familiar with fundamentals of linear algebra and information retrieval.
- The operator of the software is familiar with using CLI and basic python commands.
- In the raw input file, an image is considered only once. Repeated occurances are ignored.
- The choice of hyper-parameter values for any tasks is left to the developer, which can either be proof based or experimental.
- We are allowed to preprocess data and calculate certain graph matrices beforehand for a range of values of *k*.
- The choice of clustering algorithm - member based or group based is left upon the developer.
- For creating index structure with LSH, all the images in data-set are considered.
- If two class lables are given in the input for Task 6, they will be separated by whitespaces and belong equally to both the class labels.
- When one image gets same votes for K-NN or has the same probability for PPR from more than one labels, it can randomly be assigned any of these labels.

## V. Proposed Solution

### A. Task 1

The similarity of image ***i*** and image ***j*** is calculated as cosine similarity which can be expressed as follows:

$$Cosine\ Similarity(\mathbf{i}, \mathbf{j}) \ = \ \mathbf{i}.\mathbf{j} \ / \ |\mathbf{i}||\mathbf{j}|$$

For a given input *k*, the similarity graph is represented by creating adjacency matrix **A** such that $A_{ij} = 1$ if image ***j*** is in the top *k* most similar images of image ***i***. Note that by way of which the matrix **A** is created, it will be a directed graph.

### B. Task 2

The aim of this task, for a given input *c* and the directed similarity graph from task 1, is to partition the graph into *c* clusters. The problem of clustering directed graphs has been a topic of research in graph theory and has not been given as much importance as that of undirected graphs. [1] performs an excellent survey of various clustering frameworks discussed for directed graphs. According to [1], different algorithms for clustering undirected graphs can be divided into 3 major categories:

1) Naive graph transformation approaches: this class of algorithms ignore the direction of edges. One way of doing that is to add an undirected edge between node *i* and node *j* if there exist a directed edge from node *i*

to node *j* or vice-verse. The new adjacency matrix thus obtained will be:

$$A_{new} \ = \ A \ or \ A^T$$

Another more strict approach can be to include an undirected edge between node *i* and node *j* only if there exist a directed edge from node *i* to node *j* and from node *j* to node *i*. The new adjacency matrix thus obtained will be:

$$A_{new} \ = \ A \ and \ A^T$$

However, the new graph obtained loses the semantic information carried by the direction of edges. For example, if an edge in a directed graph represented papers that site a particular paper, transforming the graph by ignoring directions introduces reciprocal relationships that miss the point of endorsements.

2) Objective function optimization: some clustering algorithms optimize a cost function (for example modularity or normalized cut). The algorithm iteratively finds clusters until a local maximum or minimum is reached. [2], [3], [4] are some papers that optimize modularity and [5] optimized normalized weight cuts.
3) Transformations maintaining directions: these methods transform undirected graph into a directed graph such that the semantics of edge directions are not lost. [6] is one such approach that introduces edge direction information by adding weights to edges. Once the graph is undirected, any clustering algorithm for undirected graph can be applied.

For this solution we use [6] as a means to transform our undirected graph to a directed one. The reason for that is its accuracy; the algorithm has achieved 22% improvement in F-scores over previous state-of-the-art. Furthermore, it is 4-5 times faster than objective optimization techniques. We explain this algorithm next.

The definition of two matrices, Bibliographic coupling matrix (*B*) and Co-citation matrix (*C*), are essential for the understanding of the algorithm

$$B \ = \ A.A^T$$
$$C \ = \ A^T.A$$

Where *B[i,j]* represents the number of nodes node *i* and node *j* point to and *C[i,j]* is the number of nodes that point to both node *i* and node *j*. Using these matrices is intuitive because a summation of these will have higher when two nodes have similar in-degrees and out-degrees, which is exactly what we desire. However, hub nodes (nodes with extremely high degree) introduce parity in data. To normalize this effect, the paper uses in-degree ($D_{in}$) and out-degree ($D_{out}$) matrices and experimentally comes up with a formula.

$$U \ = \ D_o^{-1/2} A D_i^{-1/2} A^T D_o^{-1/2} \ + \ D_i^{-1/2} A^T D_o^{-1/2} A D_i^{-1/2}$$

Once found, we use this weighted undirected graph to apply KMeans clustering on its rows as feature vectors. To calculate distance between two vectors, we use cosine similarity. Note that the feature vectors of two nodes in *U* will be similar when they have similar in-coming and out-going edges. Therefore, it can be seen that this is an example of member based community detection wherein the common property that the nodes share is their degree.

We also explore one other type of member-based clustering. For that, we consider the matrix *W* where $W_{i,j}$ is the cosine similarity (weighted adjacency) between image *i* and image *j*. Applying KMeans clustering on the rows of this matrix signifies clustering together images that are similar to same set of images. This interpretation is much more intuitive and is an example of member-based algorithm where the common property the nodes share is similarity with same set of nodes.

For the other approach, we use our directed image-image similarity graph for clustering. Unlike our previous solution, this algorithm directly operates on the directed adjacency matrix. We have modified the standard Max-Min [13] algorithm to better fit our use-case. The algorithm can be described as follows:

**Algorithm 1** Modified Max-Min

```
1: procedure MIN-MAX-CLUSTER(graph,k)
2:     cluster-centers[0] $\leftarrow$ random node from graph
3:     $i \leftarrow 0$
4:     loop($i \leftarrow 1$ $to$ $k$):
5:       N = $argmax_n(\Sigma_i$(distance(n,cluster-centers[i]))
6:       n-max = $argmax$(in-degree(N))
7:       cluster-centers[i+1] = n-max
8:       $i \leftarrow i + 1$
9:     $i \leftarrow 0$
10:    loop($i \leftarrow 1$ $to$ $total - nodes$):
11:      c = $argmin_j$ (distance(i,cluster-centers[j]))
12:      append K[j] $\leftarrow i$
13:    return K
```

The algorithm begins by choosing a random node as the first cluster center. For each subsequent cluster centers, a node is chosen such that its sum of distances from previous cluster centers is maximum. However, if multiple such nodes exist then the algorithm picks the one with maximum in-degree because if a node has higher in-degree, it has higher probability of paths from other nodes reaching to it. Therefore, we are maximizing the probability of finding at-least one cluster center for each node. Once we have the cluster centers, the algorithm assigns each node to a cluster whose center is nearest to it. In real-life graphs, it so happens that not all nodes are connected; there exist disconnected components which may not have a path to any of the cluster centers. For such nodes, the algorithm iteratively assigns them to a cluster that has the least number of nodes in it. By doing that, it ensures that all clusters are tending towards being balanced.

### C. Task 3

For this task, we are given an image-image similarity graph (obtained after converting image-image similarity matrix into graph) and a scalar *K* which will be used to find *K* dominant images from the image set using PageRank Algorithm. PageRank (PR) is an algorithm used by Google Search to rank websites in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages. We describe this algorithm next.

The algorithms starts with matrix $A$ which is a column normalized adjacency matrix (the sum of each column is 1), so as to give equal opportunity to each outgoing node. This is achieved by the fact that we have a bound of constant degree of freedom of each node in our graph, say *M*, and therefore the value of single non zero element in the column of matrix $A$ would be $1/M$ and sum of elements in a particular column would turn out to be 1. We then calculate $\vec{u_q}$ which is given as:

$$\vec{u_q} = A * \vec{u_q}$$

Where $\vec{u_q}$ is eigenvector of the matrix A with eigenvalue equal to a scalar value 1. In the above equation the $\vec{u_q}$ at the RHS is value of $\vec{u_q}$ at previous instance and at LHS is the value at current instant. This relationship is used iteratively until the value $\vec{u_q}$ converges i.e. difference than a small value $\epsilon$. When the algorithm converges, the resultant vector $\vec{u_q}$ will contain the importance of each image.

The efficient way to find Page Rank is to devise an algorithm which is devoid of Spider Trap and Dead ends while doing random walk in the graph. This can be done using a vector $\vec{v_q}$, vector $\vec{v_q}$ is restart vector of the graph with uniform distribution of probabilities equal to value 1/(total no. of nodes in the graph). Thus the equation becomes:

$$\vec{u_q} = (1-c)A * \vec{u_q} + c\vec{v_q}$$

Where, $c$ is a constant which ranges from 0.1 to 0.2 (for Page Rank Calculations). Once we obtain the converged PageRank Vector, we get highest $K$ probability values and get the corresponding image identifiers and use the same to show them in visual manner using matlab plot.

## D. Task 4

The aim of this task, for a given input k and image ids, is to find the most relevant images using the Personalized Page Rank. For running the said algorithm, we use the directed graph / adjacency matrix from task 1 and a text file which contains the indexes of the images. When we receive the adjacency matrix from task1's output, we normalize the matrix by dividing each element by its corresponding column's sum value so that the sum of each column is 1. Before we describe our implementation of PPR, consider the following definition of Steady State Vector.

Steady State vector: Let $c$ be the probability of restarting a random walk from a node $q$. Then, the N-by-1 steady state probability vector, $\vec{u_q}$ (or simply, steady state vector) satisfies the equation:

$$\vec{u_q} = (1-c)A\vec{u_q} + c\vec{v_q}$$

Here, A is our column normalized adjacency matrix (the sum of each column is 1), so as to give equal opportunity to each outgoing node. vector $u_q$ is the steady state probability vector for the restart vector, vector v$_q$ is the restart vector of the query object. The value of $c$, which is the probability of restarting a random walk is set as 0.15 for better results [12].

The pseudo code of our implementation of personalized page rank is described in Algorithm 2. Here, *G* is the image-image similarity graph, $c$ is the probability of restarting a random walk, *tol* is the convergence lower threshold. The flag *is-per* determines whether PR is calculated or PPR. If the flag is set, it requires input image IDs for PPR and the total number of nodes in the graph which is given by *I* and *n* respectively.

**Algorithm 2** Personalized Page Rank

```
1: procedure PERSONALIZED-PAGE-RANK(G,c,tol,is-per,I,n)
2:     if(is-per)
3:         if($v_{q_i} \epsilon I$)
4:             $v_{q_i} \leftarrow 1/len(x)$
5:         else
6:             $v_{q_i} \leftarrow 0$
7:     else
8:         $v_{q_i} \leftarrow 1/n$
9:     A ← Adjacency matrix of G
10:    $\vec{u_q} \leftarrow 1/n$
11:    do
12:        $u_{qlast} \leftarrow \vec{u_q}$
13:        $\vec{u_q}$ = (1-c)A$\vec{u_q}$ + c$\vec{v_q}$
14:    while($|\vec{u_q} - \vec{u_{qlast}}| < n * tot$
```

The resultant vector $\vec{u_q}$ will contain the importance of each image, with respect to the image ids that the user entered. For displaying k most relevant ones, for choose the k highest values from the vector $\vec{u_q}$, which will be the image ids which are most closely related to the entered image ids.

## E. Task 5

Indexing is used to prune sections of search spaces during query time for fast retrieval. One such indexing technique is Locality Sensitive Hashing [10]. LSH uses a family of hash functions to index documents such that documents that are similar to each other have same hash values (collision) and documents that are dissimilar have different hash values. More formally, a family of hash functions *H* is called *(R, cR, P1, P2)*-sensitive if for any two documents **p** and **q**,

$$\text{if dist}(\mathbf{p}, \mathbf{q}) \leq R, \text{ then Prob}[\text{h}(\mathbf{p}) = \text{h}(\mathbf{q})] \geq P_1$$

$$\text{if dist}(\mathbf{p}, \mathbf{q}) \geq cR, \text{ then Prob}[\text{h}(\mathbf{p}) = \text{h}(\mathbf{q})] \leq P_2$$

where *R* is the range of distance with which collision is desired and *c* decides the approximation(error) allowed. For our use-case, we are using the euclidean distance to calculate distances between images. One most commonly used hash function family for euclidean distance is

$$h_{\mathbf{r},b}(\mathbf{x}) = (\mathbf{r}^T\mathbf{x} + b) \ / \ w$$

Here $h_{\mathbf{r},b}$ is the family of hash functions parameterized by **r** which is a random vector chosen from a 2-stable distribution. The distribution that we consider for our solution is the Gaussian Distribution. **x** is the input vector whose has is to be calculated. Therefore, we project our input vector on a random vector **r** and shift it by a random value *b*. We then divide **r** into $w$ intervals and the hash key is decided by the interval number in which the input image falls.

As can be seen, we need the values of two hyperparameters - $w$ and $b$. Before we can look at how these values can be chosen, consider the following definition of ratio

$$\rho = ln(1/P_1) \ / \ ln(1/P_2)$$

In order to reduce error in collisions, we need to have $P_1$ bigger and $P_2$ smaller as this would mean that if images **p** and **q** are near then the probability of them having the same hash value is high and if they are distant then the probability is low. Therefore, we need to find parameters that minimize the ratio $\rho$. The detailed description of $\rho$ can be referred from [9]. However, what is important is that $\rho$ is a function of $w$ and $c$. If we keep $c$ constant and plot a graph of $\rho$ vs $w$, we can gain some insights on choosing the right value of $w$. It turns out [9] has already done that and the graph can be seen in Fig. 1.

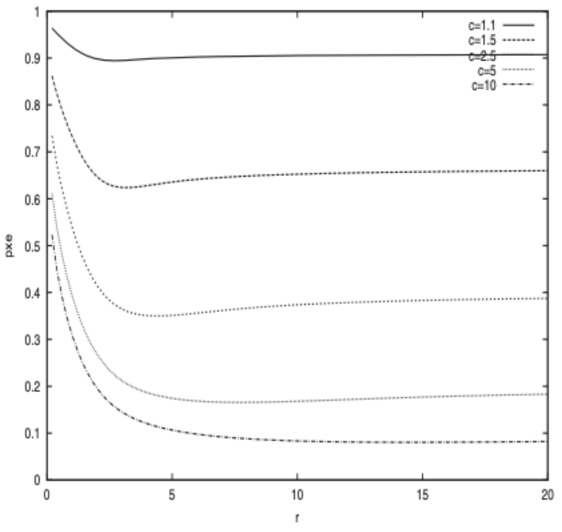

Fig. 1: Plot of $\rho$ vs $w$ for different value of $c$ [9]

From the graph, we can infer that for a sufficiently large value of $w$, $\rho$ stays constant. Therefore, any such value of $w$ can be used. For our solution, we experimented with different values in the interval [4,10] and obtained similar results. We finalized the value at 4. Furthermore, the hyperparameter $b$ can be a random value from $[0, w)$ so that we do not skip over an interval while projecting points.

In order to index images from our data-set, we choose three visual descriptor models - Color Moments (CM), Local Binary Pattern (LBP) and Color Name Histogram (CN,CN3x3). The reason for choosing these descriptors is that they are the salient-SIFT features that are visually better perceived by human eyes. They cover the color distribution (CM, CN, CN3x3) as well as the shape and texture information (LBP). Furthermore, our experiments revealed that these features generated the best results.

The process starts by taking two inputs, the number of hash tables $L$ and the number of hash conjugates in each table $k$. For each image in the data-set, $k$ hash values are found for each table and the hash key is constructed by conjugating each of the $k$ has values. Therefore, each image will have a $k$ length hash-key for each of the $L$ tables. Given a query image id, we find the hash bucket where it falls in each layer. We then retrieve all the images from those buckets and compare them with the query image to find distances. This process makes intuitive sense becks an image will have same hash key (collision) only with images that are similar to it. We then pick the top $t$ similar images based on their euclidean distances with query image, and visualize those similar images.

### *F. Task 6*

For this task we have used Personalized Page Rank algorithm (function) implemented in task 4 and all the dictionaries made in pre-processing stage. The input is first stored in a file having image-id - label pairs, with ids and multiple labels separated by whites-paces, and two different image ids separated by new lines. For k nearest neighbor, classifier we compare each image in the data-set to the list of all images which are in the input file i.e. which are labelled. Similarity between images are calculated by using cosine similarity measure on textual descriptors. The images are sorted and only the top $K$ images are stored following the principles of KNN. Furthermore, the votes are taken and stored in different dictionary having image wise fuzzy classifications for each label. The label with the highest fuzzy value is assigned to be the label of the images and thus all the images are stored label wise for visualization. In case of 2 labels having the highest fuzzy values, the final classified label can be any one of them. Thus, we finally created label wise clusters which had images pertaining to only one label with the highest votes from KNN. These label wise images are sorted on the basis of votes or fuzzy classification score so that the images with highest scores are displayed first.

For each label in the given sample input we call the personalized page rank function, wherein we pass the image ids having the same label to create $T_n$. $T_n$ matrix is the random walk matrix which has non-zero values only for the image ids specified. It is equal to $1/n$ where $n$ is the total number of images which are important for starting random walks. The PPRs which were called for all the labels, return a list of probabilities giving importance of each image for those labels. Probabilities of all the images are compared in all the lists one by one and labels are assigned to images according to the list having highest probabilistic value. What this semantically means is that each PPR returned list is the importance of all images when random walks started at images in that particular label or class. The label wise image lists are constructed and sorted according to their ppr probabilities so that we can visualize images in decreasing order of similarity.

## VI. Interface Specification

### *A. Task 1*

- Run *python3 Task1.py*
- When prompted to enter *k*, enter the number of similar images (degree of freedom) to be considered for each image

### B. Task 2

- For Max-A-Min clustering, Run *python3 Task2.py*
- When prompted to enter *k*, enter the number of similar images (degree of freedom) to be considered for each image
- When prompted to enter *c*, enter the number of desired clusters
- It will then print the cluster representatives and show 1 image for each cluster in a new window
- To continue to see further images, keep clicking on cross (window close) button
- To run member based clustering, Run *python3 Task2-m.py* with command line argument either *w* or *u*
- When prompted to enter *k*, enter the number of similar images (degree of freedom) to be considered for each image
- When prompted to enter *c*, enter the number of desired clusters
- It will then print the cluster representatives and show 1 image for each cluster in a new window
- To continue to see further images, keep clicking on cross (window close) button

### C. Task 3

- Run *python3 Task3.py*
- When prompted to enter *k*, enter the number of similar images (degree of freedom) to be considered for each image
- It will then print the dominant images and show 1 image for each cluster in a new window
- To continue to see further images, keep clicking on cross (window close) button

### D. Task 4

On running / executing the Test4.py file, the user will be asked to enter the following items:

- the value of k, which indicates the number of relevant images required.
- the image ids', 3 images needs to be entered by the user.

After the above items have been entered, the user will be displayed k most relevant images.

### E. Task 5

- Run *python Task5.py*
- It will create location maps and image dictionary and print the status
- When prompted to input *L*, enter the number of hash tables desired
- When prompted to input *k*, enter the number of hash key conjugates desired for each hash table
- It will then create index on image data and print status message
- When prompted to input image id, enter the query image id
- When prompted to input *t*, enter the desired number of similar images to be returned
- It will then display the number of collided images and total unique images out of them
- On a new window, top *t* images will be displayed along with their location ids

### F. Task 6

- When prompted to input *k*, enter the desired number of dominant images
- It will then print the dominant images and show 1 image for each cluster in a new window
- To continue to see further images, keep clicking on cross (window close) button
- Note that the labelled images are read from a file

## VII. System Requirements

- Python 3 or above
- OS: No Dependency (This code has been run and tested on Windows 10, Windows 8.1, MacOS Mojave, MacOS High Sierra, Ubuntu)
- RAM: 6GB
- Disk Space: 10GB

## VIII. Installation and Execution

Follow these steps to install and execute the software:

- Download Python 3 from here
- Download pip from here
- Install Python Dependencies using pip - numpy, configparser, pickle, xml, scipy, pillow, matplotlib, networkx, csv
- To install dependencies, Run: *pip3 install ¡dependency¿*
- Setup the project. Run: *python3 Preprocessing.py*
- To run any task, Run: *python3 ¡Task¿.py*

## IX. Conclusions

In this phase, we developed deep understanding of each task and the algorithms used in them. We summarise our learnings as follows:

- We looked at various clustering algorithms for undirected graphs and understood how they can be extended to directed graphs through semantic transformations. We learnt how Max-A-Min can be modified for our use-case. We saw the difference in member based clustering and group based clustering.
- We understood what Locality Sensitive Hashing is and how it can be used to index documents. We particularly explored a family of hash functions for euclidean distance measure (2-Norm) and identified the values and effects of each hyper-parameters.
- We acknowledged that PageRank is a way of giving probabilistic weights to all nodes of a graph. We also saw how Personalised Page Rank can take user preferences as input and calculate the same weights with a bias towards those nodes in the graph. We realized that this concept can be used for classification by starting random walks from images with labels and calculating probabilities for all other images in the graph.

## References

[1] Fragkiskos D. Malliarosa and Michalis Vazirgiannisa, *Clustering and Community Detection in Directed Network*, *Department of Informatics, Athens University of Economics and Business, Patision 76, 10434 Athens, Greeces*.

[2] E. A. Leicht and M. E. J. Newman. *Community structure in directed networks*, *Phys. Rev. Lett. 100 (2008) 118703.*

[3] Y. Kim, S. W. Son and H. Jeong. *Community structure in directed networks*, *Finding communities in directed networks, Phys. Rev. E 81 (2010) 016103*

[4] A. Arenas, J. Duch, A. Fernandez and S. Gomez. *Size reduction of complex networks preserving modularity*, *New Journal of Physics 9 (6) (2007) 176.*

[5] M. Meila and W. Pentney. *Clustering by weighted cuts in directed graphs*, in: *SDM 07: Proceedings of the 2007 SIAM International Conference on Data Mining, 2007, pp. 135144*

[6] V. Satuluri and S. Parthasarathy. *Symmetrizations for clustering directed graphs*, in: *EDBT 11: Proceedings of the 14th International Conference on Extending Database Technology, 2011, pp. 343354.*

[7] B. Ionescu, A. Popescu, M. Lupu, A.L. Ginsca and H. Muller. *Retrieving Diverse Social Images at MediaEval 2014: Challenge, Dataset and Evaluation, MediaEval Benchmarking Initiative for Multimedia Evaluation*, in: *vol. 1263, CEUR-WS.org, ISSN: 1613-0073, October 16-17, Barcelona, Spain, 2014*

[8] Ulrike von Luxburg. *A Tutorial on Spectral Clustering*, *Max Planck Institute for Biological Cybernetics, Carnegie Melon University, August 2006*

[9] Jingdong Wang, Heng Tao Shen, Jingkuan Song and Jianqiu Ji. *Hashing for Similarity Search: A Survey*, in: *CoRR, volume:abs/1408.2927, 2014*

[10] P. Indyk ansd R. Motwani. *Approximate Nearest Neighbor - Towards Removing the Curse of Dimensionality*. *In Proceedings of the 30 th Symposium on Theory of Computing, 1998, pp. 604-613*.

[11] S. Brin and L. Page. *The anatomy of a large-scale hypertextual Web search engine*. *Computer Networks and ISDN Systems 30: 107117, 1998*.

[12] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. *Automatic multimedia cross-modal correlation discovery*. in: *KDD, pages 653658, 2004*

[13] N. Karthikeyani Visalakshi and J. Suguna *K-means clustering using Max-min distance measure* *NAFIPS 2009 - 2009 Annual Meeting of the North American Fuzzy Information Processing Society*