Experimenting with Clustering, Indexing, Classification & Relevance feedback

Project Phase III Report

Submitted By

Group 19

DARSHAN D SHETTY (1211169970)

BHARATH KUMAR SURESH (1211182086)

UTHARA KEERTHI (1211106023)

DHANANJAYAN SANTHANAKRISHNAN (1211181423)

SHIVAM GAUR (1211181722)

Abstract

This phase of the project was based on the deliverables of the phase #2 in which we learnt how to operate video similarity measurements and subsequence searches in a vector space. We assumed that the SIFT vector files were already available and then we operated on them using the PCA technique on SIFT key points, extracting video features in the process. We also generated similarity graphs corresponding to frames in videos and using similarity values for edge pairs. After a certain degree of graph generation was achieved, the selection of m-most significant frames was done using ASCOS++, Page Rank & Personalized Page Rank algorithms. A modified version of ASCOS was also proposed and implemented for the same. The implementation of Locality Sensitive Hashing on multi-dimensional index structures using approaches like Nearest Neighbor Search gave us a way to visualize 2^k number of buckets in the corresponding layers of optimal hashing structure. Finally, the LSH technique was used for similar video object search in a video dataset; implementing SIFT key points in the process visualizing frames (not in the same video of the query itself) delivering the unique & overall number of SIFT vectors used and the number of bytes of data from the index accessed to process the query.

Keywords: PCA, SIFT, ASCOS, ASCOS++, LSH, similarity, buckets, page rank.

Introduction

Ø Terminology

SIFT (Scale-invariant feature transform) is an algorithm used to identify local features in the image. This is used to obtain points of interest in the image. This information can be used to locate objects of interest in the image [2].

Principal Component Analysis also called K-L transform is a statistical procedure to convert a set of correlated variables distributed and culminated in a set relation into a set of single/linear related variables. The procedure is heavily used to reduce dimensionality of a vector space under consideration. It uses a covariance matrix relation and Eigen decomposition relations to obtain a final set of vectors which are present in a reduced or possible linear dimension. Also, a principal component is one which provides no advantage if a further dimensionality reduction is done because it relates to all features symmetrically. This approach is very sensitive to original description of vector space [19] [9].

Locality Sensitive Hashing is an approach to reduce dimensionality of a High dimension spatial data. "LSH hashes input items so that similar items map to the same "buckets" with high probability (the number of buckets being much smaller than the universe of possible input items)". It uses a maximization procedure to increase the collisions between similar objects and is close to nearest neighbor or clustering of data [10].

Page Rank is an approach which states that in a network the most connected states are the most accessed unless it is personalized to a redundant behavior, and an online explanation is "PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites" [11].

Euclidean Distance is a metric which gives a distance metric between two spatial points in Euclidian Space. Thus Euclidian space is similar to metric space [12].

The **Euclidean distance** between points **p** and **q** is the length of the line segment connecting them ($\overline{\mathbf{pq}}$).

In Cartesian coordinates, if **p** = (p_1, p_2,..., p_n) and **q** = (q_1, q_2,..., q_n) are two points in Euclidean *n*-space, then the distance (d) from **p** to **q**, or from **q** to **p** is given by the Pythagorean formula:

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

Euclidean Distance, Ref. [12]

Intersection Similarity also called Jaccard Index/Similarity Coefficient is a measure of how similar two distributions are. It compares the maximum and minimum value of the distribution to determine how close two data sets really are [7].

If $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ are two vectors with all real $x_i, y_i \geq 0$, then their Jaccard similarity coefficient is defined as

$$J(\mathbf{x},\mathbf{y}) = \frac{\sum_i \min(x_i, y_i)}{\sum_i \max(x_i, y_i)},$$

Generalized jaccard similarity, Ref. [7]

Cosine Similarity is an angle similarity measure which is designated to measure the similarity between a set of dynamic data sets according to their inner product space. It is generally implemented in positive part of statistical space and the outcome in this situation is bounded between 0 and 1 and is heavily used for text based comparisons. Its major use is in data mining and information retrieval [8].

Given two vectors of attributes, *A* and *B*, the cosine similarity, *cos(θ)*, is represented using a dot product and magnitude as

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$, where A_i and B_i are components of vector A and B respectively.

Generalized jaccard similarity, Ref. [8]

Cosine distance is calculated as (1-Cosine similarity).

Mahalonobis Distance is a measure of distance between two vectors in a vector space which denotes the analysis of their Euclidian distance along with an emphasis on Statistical Distribution of points in space. Also, it can be concluded as a multi-dimensional approach towards standard deviations calculations [18]. The following formula is used to calculate Mahalonobis distance between 2 vectors $\vec{a}, \vec{b}$.

$$\Delta_{Mah}\left(\vec{a},\vec{b}\right) = \sqrt{(\vec{a}-\vec{b})^T S^{-1} (\vec{a}-\vec{b})} \; where\, S[x,y] = Cov(x,y) = E((x-\mu_x)(y-\mu_y))$$

Here, $\Delta_{Mah}\left(\vec{a},\vec{b}\right)$ is the Mahalonobis distance between $\vec{a}$ and $\vec{b}$ and S is the covariance matrix

Ø Goal Description

TASK #1

Given target dimensionality d and the SIFT vectors, the goal is to implement a program which computes the PCA over SIFT key points in a (d) dimension space and also report the (d) dimensions in terms of the input vector space.

TASK #2

Given an integer (k) and PCA output, the goal is to implement a program that creates a similarity graph for with edges representing similarity between video frames considering k, and store the output in a file where every line represents similarity between two video frames.

TASK #3

Given an integer (m) and a graph file having the similarity graphs generated in the previous task, the goal is to implement a program which will identify most significant (m) frames in the collection using Page Rank and ASCOD measures and visualize them for both approaches.

TASK #4

Given an integer (m), 3 input frames as pair of video/frame ids and a graph file having the similarity graphs generated in the task #2, the goal is to implement a program which will identify most relevant (m) frames (relative to input frames) in the collection using Personalized Page Rank and a suitably modified version of ASCOS measures to account for seed frames.

TASK #5

Given the SIFT key points, a certain number of layers (L) a number (2^k) of buckets per layer, the goal is to implement a program which maps each key point into a hash bucket for each layer.

TASK #6

Given the LSH index file from previous task, an integer (n) and an object described as <i;j;<x_1,y_1>;< x_2,y_2>>; where (i) and (j) are video and frame numbers and the rest denotes a rectangle containing object, the goal is to implement a program that visualizes (n) frames (not in the same video of query) that contain the most similar video objects; also providing the number of unique & overall SIFT vectors considered including the number of bytes of data from the index, accessed to process query.

Ø Assumptions

- The path to the input files is already known; i.e. out_file.sift for SIFT vectors and are valid.
- The Phase 3 file inputs are of the form mentioned in Phase 2. Also, memory is not a huge concern for the user as the data structure which stores the input data depends on the number of videos and hence, may be large.
- The programs process videos sorted by their names in alphabetical order. The video index starts from 0. i.e. Video 0 corresponds to the first video and video n-1 corresponds to the last one.
- The input video files are assumed to be MP4 files.
- Target dimensionality in the computation of PCA & K-means is assumed to be less than original dimensionality of the input data.
- For Tasks #1 and #2, d and k values are positive.
- For Task #5, the value of K, L the number of buckets and the number of layers is > 0.
- Tasks #2 and #5 are computationally expensive and thus require exceptional time and spatial complexity tolerance at user as well as application level.
- For Tasks #3 and #4, the parameters c and α are 0.6 and 0.85 respectively.

Proposed Solution

Task 1:

For Task 1, read the input data as n x k matrix where n is the number of data and k is the number of original dimensions and compute co-variance matrix for the input matrix (k x k). Do Eigen Decomposition for the co-variance matrix. Sort the eigen values and the corresponding eigen vectors in non-increasing order and select the first k eigen vectors. Find the dot product of the data matrix and the eigen vectors. This gives the transformed data matrix in the new vector space. After this select the top k eigen vectors and print it in the file based on the corresponding dimension index and score

Task 2:

For Task 2 to determine similarity between video file and generate the graph between similar edges, we start with using Manhattan distance to determine the nearest neighbor for vectors in the frame of the given video by comparing them with the frames of the rest of the videos, one at a time. After identifying the nearest vector, calculate the final vector distance as a weighted sum of the Euclidean distance between the vectors using location and the Manhattan distance calculated for rest of the dimensions of the vector. The frame distance is the sum of the vector distances. This distance is stored for all frames that were compared and the top k pruned and written to the file based on user input.

Task 5:

For task 5, we are supposed to develop a family of locality sensitive hash functions for the distance measure we use. The input given to us is the set of SIFT vectors (generated for the given set of demo video files) over which PCA was run to reduce the dimensionality from 130 (Scale, Orientation and 128 descriptors) to ‘d’ specified in task 1. We researched about different locality sensitive hash functions for different distance measures like Cosine Similarity, Euclidean and Mahalanobis Distance and evaluated the pros and cons of those. From our

research, we found that the Scale and Orientation values present in the SIFT vectors play a dominating role in the calculation of similarity i.e. two SIFT vectors with highly dis-similar 128 SIFT descriptors could give high similarity owing to the scale and orientation values and vice-versa. The reason behind this is that the 128 SIFT descriptors form a unit vector and their descriptors can take any value as shown in the figure below.

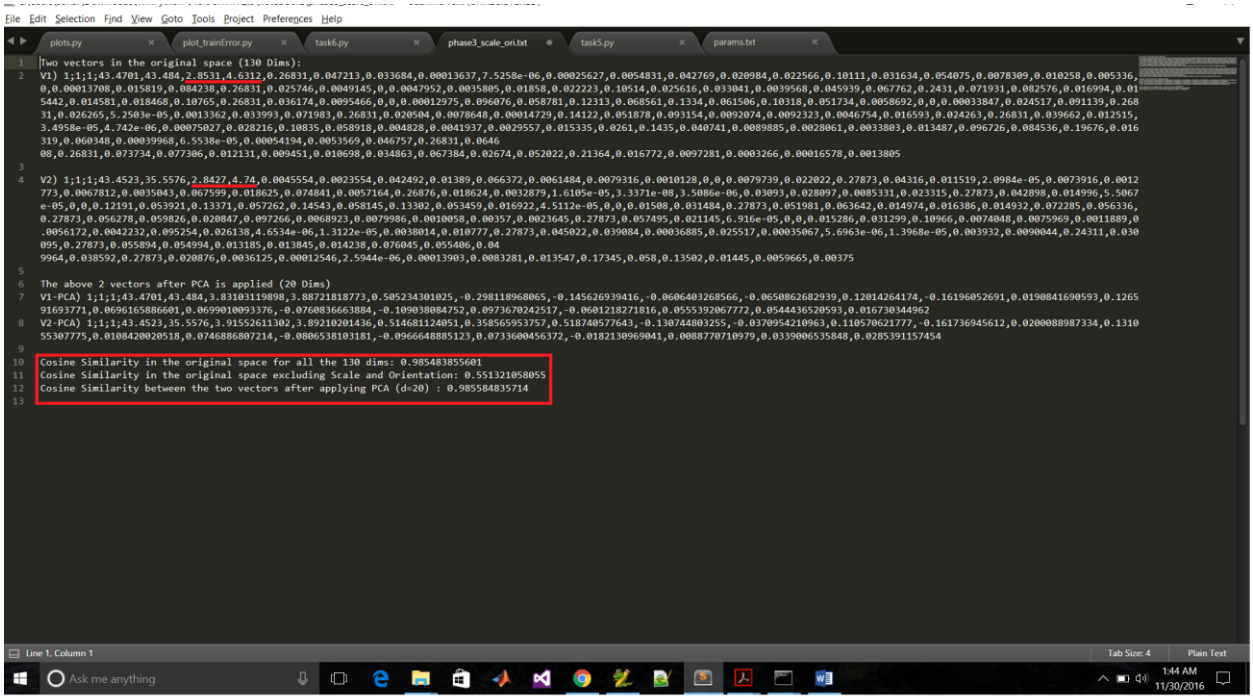

Fig.1 Demonstrates the inefficiency of Cosine Similarity

Because of this, the Locality Sensitive Hash family we built for Cosine Distance [3] (= 1 – Cosine Similarity) resulted in high collisions in the hash table (buckets in different layers) which ultimately lead to unmanageable number of False Positives. Also, we see that the application of PCA doesn't affect this as most of the distance and angles are preserved even after projection. Since we have highly different variances among different features, we tried incorporating Mahalanobis Distance Metric to develop a family of hash functions [14]. Our approach was based on the way LSH families are built for Euclidean Distance (explained later) except that we use the Inverse of the Covariance matrix to calculate the projection of a vector on a random unit vector. The resultant hash family was highly sensitive to even extremely small change in the data. Two vectors that were highly similar were mapped to different buckets. Since this method produced more of an exact hash function over LSH function, it gave to high False Negatives. Based on the above work, we decided to use Euclidean Distance measure as it provided a tradeoff between the false positives and Negatives. We built a LSH family of functions for Euclidean distance [15]. The core idea is to generate a random set of Unit vectors corresponding to the number of hash functions we need and find the projections of the given vectors on those unit vectors. The unit vectors over which the given points are projected are divided into grids of a size (D). The hash value returned by the function is the index of the grid corresponding to the projection of the given vector on the unit vector associated with that hash function. The grid size could be modified to produce different values of 'r' and 'C' for the LSH family. We also take the mod of the indexed value to satisfy the problem statement's expectations.

"The LSH algorithm relies on the existence of locality-sensitive hash functions. Let (H) be a family of hash functions mapping $\mathbb{R}^d$ to some universe U. For any two points p and q, consider a process in which we choose a function h from (H) uniformly at random, and analyze the probability that h(p) = h(q). The family H is called *locality sensitive* (with proper parameters) if it satisfies the following condition.

Definition 2.3 (Locality-sensitive hashing). A family (H) is called (*R*, *cR*, *P*1, *P*2)-sensitive if for any two points $p, q \in \mathbb{R}^d$.

• if $||p - q|| \leq R$ then $\Pr_H [h(q) = h(p)] \geq P1$,

• if $||p - q|| \geq cR$ then $\Pr_H [h(q) = h(p)] \leq P2$.

For a locality-sensitive hash (LSH) family to be useful, it has to satisfy $P1 > P2$." [16]

For Euclidean distance, the procedure we follow to build hash functions is as follows. The number of hash functions in the family (H),

- No. of hash functions required to be generated for the family (H), N=No. of hash functions per layer (K) * No. of Layers (L)
- For each of the hash functions $h_i \in \{h_1, h_2, h_3, h_4, \ldots, h_N\}$ in (H) we generate a random unit vector u_i with no. of dimensions = d, where 'd' is the input given to task 1. i.e. 'd' is the target dimensionality given to PCA for dimensionality reduction.
- For each of the input vectors $v_j \in$ {SIFT vectors present in the input file},
  - Projection of Vector v_j on vector $u_i = v_j \cdot u_i$, i.e. the dot product between the two vectors (since u_i is a unit vector)
  - A hash function h in the family returns the Index of the projection of v_j on u_i, on $u_i = (v_j \cdot u_i)/D$, where D is the size of each grid along the vector u_i. Then, For any two input vectors v_a and v_b,
    - If $\Delta(v_a,v_b) = ||v_a - v_b|| < D/2$, $\Pr[h(v_a)=h(v_b)] \geq 1/2 = P1$ and
    - If $\Delta(v_a,v_b) = ||v_a - v_b|| > 2D$, $\Pr[h(v_a)=h(v_b)] \leq 1/3 = P2$
    - As P1>P2 it is an efficient Locality Sensitive Hash function [15][16]
  - Final value returned by the hash function $h_i(v_j) = (v_j \cdot u_i)/D \bmod (2^b)$, where 'b' is the band or width of the output by the hash function h_i. As the value of 'b' increases the probability of collisions decreases [15].

As the number of buckets 'K' increases, the probability of collisions exponentially decreases. Hence, we see a lot of False positives in cases where K is low.

Task 6

Our approach for task 6 completely depends on the output from task 5. The input file of the form *.lsh is parsed to generate the LSH Data Structure. The LSH Data Structure(LSHDS) is a Python Dictionary of Dictionary of Lists [17]. The first level of the LSHDS contains the Bucket IDs of the form "Layer No._Bucket No." Now each of these entries contain a dictionary with the key values of the form "Video No._Frame No." Each of these is a list containing the index of the vectors (present in the original *.sift and *.spca file). The vectors corresponding to the query video are not added to the LSHDS. This is because the problem statement says retrieve the best vectors that are not the same video as the query. The vectors present in the Query Rectangle defined by the video no. frame no. and the diagonals are added to a dictionary again. The keys of this dictionary are the line indices of the query vectors and the values is a list containing the Bucket IDs of the same. Now using these 2, we sort the frames in descending order of the number of vectors they have in common with the query vectors. We display the 'n' best frames based on the user's choice. Sometimes when there are a lot of collisions in the hash table, there might be cases where there are a lot of frames that share the same no. of vectors with that of the query video (FP). In that we display only until 'n' frames specified based on the order in which the retrieved frames are stored in the list.

Input / Output File Formats

Task #1:

Input Format:

An integer value for target dimension (d) and location of the file out_file.sift containing SIFT vectors where each line is of the format: $<i;j;k;v_1,v_2,v_3 \ldots v_n>$ where *i*: video number, *j*: frame number, *k*: cell number, $v_{p(v1 \ldots vn)}$: describing the SIFT vector.

Output Format:

Task #1 stores output in filename_d.spc with each line with the format {<i,j,l,x,y>, [dim_i, …, dim_d]}, where i, j, l is the index of the video file (files sorted alphabetically), frame no, cell no, respectively; (x, y) tuple contain position of SIFT key point and (dim_1….dim_d) represent the values of scale, orientation and descriptors in d dimensions.

Task #2:

Input Format:

An integer value k indicating the number of similar frames to identify and the input file filename_d.spc containing PCA output from Task#1. Each line in the file is of the following format: $<i;j;k;v_1,v_2,v_3 \ldots v_n>$ where *i*: video number, *j*: frame number, *k*: cell number, $v_{p(v1 \ldots vn)}$: SIFT vector in reduced dimensionality.

Output Format:

Task #2 creates a similarity graph G(V, E), where V denotes frames of the videos ; E denotes edge pairs (v_a, v_b) such that, for each video frame it is one of the k most similar frames to v_a that is not already in the same video file. This is stored in filename_d_k.gspc where each line is of the format v_a, v_b, sim(a, b);

$v_a = <i_a, j_a>$ - i_a,j_a are the index of the video file(files sorted alphabetically) and frame no respectively in video v_a

$v_b = <i_b, j_b>$ i_b,j_b are the index of the video file(files sorted alphabetically) and frame no respectively in video v_b sim(a, b) is the degree of similarity between these two frames.

Task #3a:

Input Format:

Enter option 1 to run Page Rank program. Enter the path of the directory containing the video files. This is for visualizing the significant frames. Enter the path of the graph file *filename*_d_k.*gspc* and an integer *m*, the number of most significant frames.

Output Format:

The *m* most significant frames are printed in the following format: (*i,j)* - *p* , where *i*: video number, *j*: frame number, *p*: corresponding page rank of the frame. Each frame in the result is also displayed. See Fig. below:

```
C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\task34>python pagerank.py
What you want to run(Enter 1 or 2)? 1.PageRank or 2.RPR : 1
Please enter video DB Path with trailing slash: C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\DemoVideos\DemoVideos\
Please enter top-D ranking frames to show: 20
Enter the path of the graph file : C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\InputOutputFiles\filename_d_k_2.gspc
pagerank.py:103: RuntimeWarning: invalid value encountered in true_divide
  normalizedSim = similarityMatrix / np.sum(similarityMatrix,axis=0)
(14, 1) - 0.015400908091660592
(10, 1) - 0.014028309859773798
(27, 2) - 0.010917560681068831
(17, 1) - 0.010476308683726695
(7, 1) - 0.009054951309421593
(23, 1) - 0.006272391265906483
(4, 1) - 0.002511038288316732
(20, 1) - 0.002138793051958385
(56, 13) - 0.0021329262303734038
(24, 1) - 0.0018488519547101147
(37, 17) - 0.0007018698388373869
(20, 3) - 0.0007014668167877608
(14, 2) - 0.0006500015797177692
(20, 19) - 0.0005507675018946616
(15, 1) - 0.0005291005291005291
(11, 1) - 0.0005291005291005291
(37, 13) - 0.0005273430085223561
(37, 7) - 0.0005273228403308505
(37, 15) - 0.0005272851417011 99
(40, 13) - 0.0005263843609342618
20
```

Task #3b:

Input Format:

Enter the path of the directory containing the video files. This is for visualizing the significant frames. Enter the path of the graph file *filename*_d_k.*gspc* and an integer *m*, the number of most significant frames.

Output Format:

The *m* most significant frames are printed in the following format: (*i,j)* - *s* , where *i*: video number, *j*: frame number, *s*: corresponding ASCOS++ score of the frame. Each frame in the result is also displayed. See Fig. below:

```
C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\task34>python ascosplus.py
Please enter video DB Path with trailing slash: C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\DemoVideos\DemoVideos\
Please enter top-D ranking frames to show: 20
Enter the path of the graph file : C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\InputOutputFiles\filename_d_k_2.gspc
ascosplus.py:112: RuntimeWarning: invalid value encountered in true_divide
  normalizedSim = similarityMatrix / np.sum(similarityMatrix,axis=0)
(17, 1) - 0.019727672632502052
(14, 1) - 0.015434120965805464
(23, 1) - 0.012453116878464484
(27, 2) - 0.008962410762934699
(10, 1) - 0.008471638756341549
(7, 1) - 0.005418933597105773
(4, 1) - 0.003291570131763399
(20, 1) - 0.0026007104487088778
(24, 1) - 0.002226474234022911
(56, 13) - 0.0021999192299104664
(15, 1) - 0.0011766857142857144
(11, 1) - 0.000806095238095238
(34, 1) - 0.0007506962962962961
(8, 1) - 0.0005324867724867724
(14, 2) - 0.0005018213529907889
(62, 1) - 0.0004800825011965545
(20, 3) - 0.00046729944440480453
(33, 23) - 0.00042425497805567925
(37, 17) - 0.00041432674512679356
(30, 12) - 0.0003877734991121901
```

Task #4a:

Input Format:

Enter option 2 to run Personalized Page Rank program. Enter the path of the directory containing the video files. This is for visualizing the significant frames. Enter the path of the graph file *filename*_d_k.*gspc* and an integer *m*, the number of most significant frames. Enter the video number *v* and the frame number *f* for each of the 3 seed frames, one-by-one.

Output Format:

The *m* most significant frames are printed in the following format: (*i,j)* - *p* , where *i*: video number, *j*: frame number, *p*: corresponding RPR-2 score of the frame. Each frame in the result is also displayed. See Fig. below:

```
C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\task34>python pagerank.py
What you want to run(Enter 1 or 2)? 1.PageRank or 2.RPR : 2
Please enter video DB Path with trailing slash: C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\DemoVideos\DemoVideos\
Please enter top-D ranking frames to show: 20
Enter the path of the graph file : C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\InputOutputFiles\filename_d_k_2.gspc
pagerank.py:136: RuntimeWarning: invalid value encountered in true_divide
  normalizedSim = similarityMatrix / np.sum(similarityMatrix,axis=0)
Enter the 3 seed frames: Video number followed by Frame number
Enter the video number: 3
Enter the frame number: 1
Enter the video number: 1
Enter the frame number: 5
Enter the video number: 7
Enter the frame number: 8
[(1, 5), (3, 1), (7, 8)]
(7, 8) - 0.05000000000000001
(1, 5) - 0.05000000000000001
(3, 1) - 0.05000000000000001
(45, 1) - 0.04120289396274766
(27, 2) - 0.03285975105270356
(33, 8) - 0.02240548169017854
(33, 9) - 0.020292772775275074
(56, 13) - 0.01746580738024153
(14, 1) - 0.01314767382600919
(10, 1) - 0.012799636260883472
(17, 1) - 0.010201755077951473
(7, 1) - 0.00688974075869635
(23, 1) - 0.006159296027811245
(37, 17) - 0.0005277411169086895
(20, 3) - 0.0004766201562838772
(14, 2) - 0.000470487503234716
(20, 19) - 0.00039921062445438257
(15, 1) - 0.0003822751322751323
(11, 1) - 0.0003822751322751323
(37, 13) - 0.0003795589641088649
20
```

Task #4b:

Input Format:

Enter the path of the directory containing the video files. This is for visualizing the significant frames. Enter the path of the graph file *filename*_d_k.*gspc* and an integer *m*, the number of most significant frames. Enter the video number *v* and the frame number *f* for each of the 3 seed frames, one-by-one.

Output Format:

The *m* most significant frames are printed in the following format: (*i,j)* - *s* , where *i*: video number, *j*: frame number, *s*: corresponding modified ASCOS++ score of the frame. Each frame in the result is also displayed. See Fig. below:

```
C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\task34>python ascosmodified.py
Please enter video DB Path: C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\DemoVideos\DemoVideos\
Please enter top-D ranking frames to show: 20
Enter the path of the graph file : C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\InputOutputFiles\filename_d_k_2.gspc
ascosmodified.py:113: RuntimeWarning: invalid value encountered in true_divide
  normalizedSim = similarityMatrix / np.sum(similarityMatrix,axis=0)
Enter the 3 seed frames: Video number followed by Frame number
Enter the video number: 3
Enter the frame number: 1
Enter the video number: 1
Enter the frame number: 5
Enter the video number: 7
Enter the frame number: 8
(45, 1) - 0.017508172855607712
(7, 8) - 0.017491219447277915
(1, 5) - 0.017491219447277915
(3, 1) - 0.017491219447277915
(27, 2) - 0.0163796511862517
(14, 1) - 0.011504130870052478
(17, 1) - 0.010177349214594495
(10, 1) - 0.007134859358792448
(56, 13) - 0.006392202260546126
(23, 1) - 0.0060787469553669786
(7, 1) - 0.005807928840936914
(33, 8) - 0.005146493460632994
(33, 9) - 0.0045672194704418825
(4, 1) - 0.004273471494717906
(20, 1) - 0.0033613238115218127
(24, 1) - 0.0028841726377972043
(15, 1) - 0.0009231152539894181
(11, 1) - 0.0007743187322751322
(62, 1) - 0.0006293348027513756
(14, 2) - 0.0006225383457799387
```

Task #5:

Input Format:

- Task 5 is executed by task5.py python script.
- <Usage> python task5.py <Options>.
- Use -h option to display the options.

The different options the script takes is shown below:

```
C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\Phase3_2.0>python task5.py -h
Usage: task5.py [options]

Options:
  -h, --help      show this help message and exit
  -I INFILEPATH   Input file name of the form filename_d.spca, default name
                  out_file_d.spca
  -L NOOFLAYERSL  Number of layers L, default value=3
  -K NOOFBITS     Number of buckets per layer, 2^K, default value=15
  -O OUTFILEPATH  Output File of the form filename_d.lsh, default name of the
                  form InputFileName_d.lsh
  -D GRIDSIZE     Size of each grid D, default value=0.12
  -M              Define whether to use Mahalanobis Distance over Euclidean,
                  The option is Disabled
  -b BAND         No. of Bits each hash function contributes, Default value=2
```

Fig. Options taken by task5.py

```
C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\Phase3_2.0>python task5.py -I ..\InputOutputFiles\out_file_d_60D.spca -O ..\InputOutputFiles\out_file_60D_K15_L6.lsh -K 15 -L 6 -D 0.12
Time Taken for execution:1365.19312571 seconds
```

- The '-M' option has been disabled due to accuracy/efficiency.

Output Format:

- The script computes the buckets to which each vector is hashed into for all the layers.
- If there are L layers given in the input, the first 'L' lines would represent the first vector given in the input hashed into different buckets.
- Layer Numbers and Bucket Numbers start from 0.

- The output is of the form
  - Layer number, bucket number, <Video No.;Frame No.;Cell No.;Keypoint position(x);Keypoint position(y)>
  - The output is generated for every vector in the input file
  - No. of lines in the output file = No. of lines in the input file * no. of layers (L)
  - The program also prints the time taken for execution in seconds

The screenshot below displays the output when run for K=5 and L=3:

```
0,21,<1;1;1;23.4658;23.4698>
1,26,<1;1;1;23.4658;23.4698>
2,14,<1;1;1;23.4658;23.4698>
0,4,<1;1;1;23.4658;23.4698>
1,3,<1;1;1;23.4658;23.4698>
2,2,<1;1;1;23.4658;23.4698>
0,24,<1;1;1;23.4614;35.5547>
1,28,<1;1;1;23.4614;35.5547>
2,11,<1;1;1;23.4614;35.5547>
0,9,<1;1;1;23.4614;35.5547>
1,3,<1;1;1;23.4614;35.5547>
2,30,<1;1;1;23.4614;35.5547>
0,21,<1;1;1;23.4568;63.4486>
1,26,<1;1;1;23.4568;63.4486>
2,14,<1;1;1;23.4568;63.4486>
```

Fig. Sample output for task 5

The screenshot below displays the output when run for K=10 and L=3:

```
0,981,<1;1;1;23.4658;23.4698>
1,770,<1;1;1;23.4658;23.4698>
2,850,<1;1;1;23.4658;23.4698>
0,402,<1;1;1;23.4658;23.4698>
1,518,<1;1;1;23.4658;23.4698>
2,1000,<1;1;1;23.4658;23.4698>
0,516,<1;1;1;23.4614;35.5547>
1,575,<1;1;1;23.4614;35.5547>
2,51,<1;1;1;23.4614;35.5547>
0,1,<1;1;1;23.4614;35.5547>
1,307,<1;1;1;23.4614;35.5547>
2,121,<1;1;1;23.4614;35.5547>
0,981,<1;1;1;23.4568;63.4486>
1,770,<1;1;1;23.4568;63.4486>
2,850,<1;1;1;23.4568;63.4486>
0,402,<1;1;1;23.4568;63.4486>
1,518,<1;1;1;23.4568;63.4486>
2,1000,<1;1;1;23.4568;63.4486>
```

Fig. Sample output for task 5

Task #6:

Input Format:

- Task 6 is executed by task6.py python script.
- <Usage> python task6.py Command_Line_Parameters.
- Run python task6.py to get information about the different Command Line arguments.
- The script takes 9 command line parameters.
- They are <Path to the input file, No. of frames to be retrieved, Video No., Frame No, x1,y1,x2,y2,Path containing the demo Videos>.
- (x1,y1) and (x2,y2) are the co-ordinates of the diagonals containing the rectangle.

Example: python task6.py NewOutputFiles\out_file_60d_K15_L6.lsh 10 28 2 4 300 50 350 C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\DemoVideos\DemoVideos\

Refer to the fig. below for more details:

```
C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\Phase3_2.0>python task6.py
Usage:   python task5.py InputFile n i j x1 y1 x2 y2 Demo_Videos_Directory

Where
        - InputFile is the path to one of the output files generated by Task 5
        - 'n' represents the no. of frames containing similar objects
        - i and j represent the query video and frame no. (Videos in the directory will be of the form 'i.mp4')
        - (x1,y1) form the coordinates of one of the vertices of the rectangle containing the object to be queried in the given video and frame
        - (x2,y2) form thecoordinates of the vertex that is diagonally opposite to (x1,y1)
        - Demo_Videos_Directory is the path to the directory (without spaces) containing the given Demo Videos
Time Taken for execution:0.0019690683469383227 seconds

C:\D-Drive\studies\MCS\MWDB\Project\Phase3_bharath\Phase3_2.0>
```

Fig. Showing task6.py's usage

Output Format:

The script outputs the following:

- Unique No. of SIFT Vectors considered
- Overall No. of SIFT Vectors considered
- No. of bytes of data from the index to process the query
- Display of each of the 'n' (no. of frames to be retrieved) frames retrieved, one by one
- 'n' entries of the form (Video No., Frame No.)
- Time Taken for execution in seconds

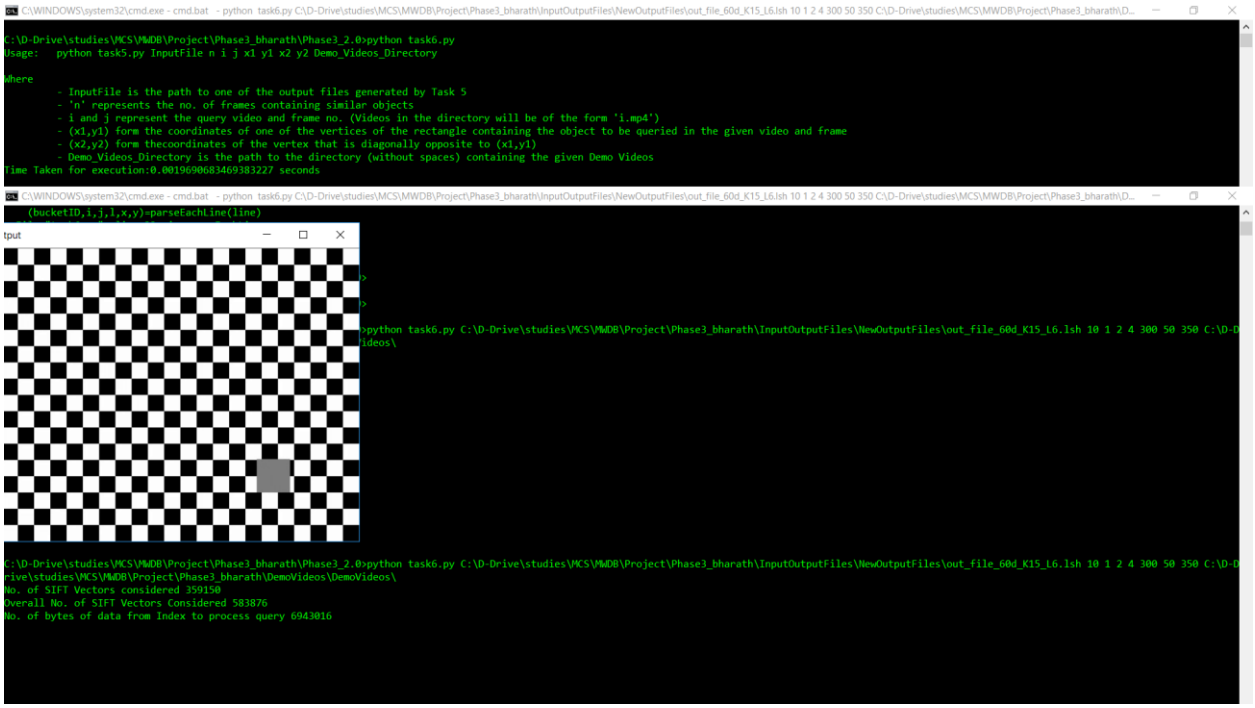

Fig. Showing one of the frames displayed when the code was run

Fig. Output printed to console when task6.py is run

Implementation

Task #1:

1. Read the input data as n x k matrix where n is the number of data and k is the number of original dimensions.
2. Compute the co-variance matrix for the input matrix (k x k).
3. Do Eigen Decomposition for the co-variance matrix.
4. Sort the eigen values and the corresponding eigen vectors in non-increasing order.
5. Select the first k eigen vectors.
6. Find the dot product of the data matrix and the eigen vectors.
7. This gives the transformed data matrix in the new vector space.
8. After this select the top k eigen vectors and print it in the file based on the corresponding dimension index and score.

Task #2:

1. Parse the input file and create a list of lists for all videos containing reduced dimension vectors for each frame in the video.
2. Do the following for each frame of the video by comparing with frames from all other videos:
   a. Determine the nearest neighbor for each vector using Manhattan distance and calculate vector distance.
   b. Calculate frame distance as the sum of vector distances.
   c. Store the frame distance in a list.
   d. Prune the frame distance list based on user input k.
   e. Store this data in filename_d_k.gspc for the frame.

Task #3a:

1. Enter the path of the video files as input.
2. Enter the number(m) of the most significant frames as input.
3. Enter the path to the graph file, read it and create the similarity matrix.
4. Create a structure mapping the (video number, frame number) of a frame with the index k such that this frame is the k^{th} node of the graph.
5. Normalize the similarity matrix. This becomes the transition matrix T_G corresponding to the graph.
6. Initialize the page rank column vector with the value 1/n for every frame. Here, n denotes the total number of frames in all the videos.
7. Compute the page rank equation $\vec{p} = \alpha T_G \vec{p} + (1-\alpha)\vec{v}$ [6] where $\vec{p}$ is the page rank vector, T_G is the transition matrix and $\vec{v}$ is the teleportation vector with all entries as 1/n. The α parameter is set as 0.85.
8. The vector returned by the convergence of the above equation is the page rank vector corresponding to the graph.
9. Sort the page rank vector and print the video number, frame number and the page rank of the top m significant frames.

10. Display the m significant frames.

Task #3b:

1. Enter the path of the video files as input.
2. Enter the number(m) of the most significant frames as input.
3. Enter the path to the graph file, read it and create the adjacency matrix $A = [a_{ij}]$.
4. Create a structure mapping the (video number, frame number) of a frame with the index k such that this frame is the k[th] node of the graph.
5. Normalize the adjacency matrix; the resultant matrix is $P = [p_{ij}] = \sum_{\forall k} a_{ij}/a_{kj}$. [5]
6. Compute the matrix $Q = [q_{ij}] = [p_{ij}(1 - \exp(a_{ij}))]$. [5]
7. Next, compute the ASCOS++ score matrix $S = (I - cQ^T)^{-1}(1 - c)I$ [5] where I is the identity matrix and the parameter c is set to 0.6.
8. Sort the ASCOS++ matrix S.
9. For the m most significant frames, print the video number, frame number and the corresponding ASCOS score.
10. Display the m most significant frames.

Task #4a:

1. Enter the path of the video files as input.
2. Enter the number(m) of the most significant frames as input.
3. Enter the path to the graph file, read it and create the similarity matrix.
4. Create a structure mapping the (video number, frame number) of a frame with the index k such that this frame is the k[th] node of the graph.
5. Normalize the similarity matrix. This becomes the transition matrix T_G corresponding to the graph.
6. Enter the video number and frame number of the 3 seed frames as input.
7. Initialize the page rank column vector with the value 1/n for every frame. Here, n denotes the total number of frames in all the videos.
8. Set the entries of the seed vector to be 1/|S| for all seed nodes and 0 for all non-seed nodes. Here, S denotes the set of seeds.
9. Now, we implement the RPR-2 algorithm.
10. First, we compute the personalized page rank equation $\overrightarrow{\pi_i} = \alpha T_G \overrightarrow{\pi_i} + (1 - \alpha)\overrightarrow{s_i}$ for every seed node $v_i \in S$ where S is the set of seed nodes. Here, $\overrightarrow{s_i}$ is a restart vector where $\overrightarrow{s_i}[i] = 1$ and $\forall_{j \neq i}\ \overrightarrow{s_i}[j] = 0$. [4]
11. Next, we compute the sum of $\overrightarrow{\pi_i}[j]$ over every seed node $v_j \in S$. This is computed for every seed node $v_i \in S$, i.e. $\prod(v_i) = \sum_{v_j \in S} \overrightarrow{\pi_i}[j]$. [4]
12. We then find the maximum of these sums $\prod(v_i)$, using which we find the subset S_{crit} subset of S, where $S_{crit} = argmax_{v_i} \prod(v_i)$. [4]
13. Next, compute the average of the $\overrightarrow{\pi_i}$ values for every $v_i \in S_{crit}$. This gives the RPR-2 scores corresponding to the graph.
14. Sort the RPR-2 scores and print the video number, frame number and the corresponding score for the m most significant frames.
15. Display the m most significant frames.

Task #4b:

1. Enter the path of the video files as input.
2. Enter the number(m) of the most significant frames as input.
3. Enter the path to the graph file, read it and create the adjacency matrix $A = [a_{ij}]$.
4. Create a structure mapping the (video number, frame number) of a frame with the index k such that this frame is the k[th] node of the graph.
5. Normalize the adjacency matrix; the resultant matrix is $P = [p_{ij}] = \sum_{\forall k} a_{ij}/a_{kj}$. [5]
6. Compute the matrix $Q = [q_{ij}] = [p_{ij}(1 - \exp(a_{ij}))]$. [5]
7. Enter the video number and frame number of the 3 seed frames as input.
8. Compute the weight matrix $W = [w_{ij}]$ where $w_{ij} = 1/|S|\ \forall i$, if the j^{th} frame is a seed node, and $w_{ij} = 0\ \forall i$, if the j^{th} frame is a non-seed node. [5]

9. Next, compute the modified ASCOS++ score matrix $S = (I - c(\alpha Q^T + (1-\alpha)W))^{-1}(1-c)I$ where I is the identity matrix and the parameters c and α are set to 0.6 and 0.85 respectively.
10. Sort the ASCOS++ matrix S.
11. For the m most significant frames, print the video number, frame number and the corresponding ASCOS score.
12. Display the m most significant frames.

Task #5:

As discussed earlier, task 5 builds a family of hash functions for Euclidean Distance. The procedure is explained in the steps given below.

1. Parse the Input arguments such as Input File Path, No. of layers, No. of Buckets per layer, Output file, Size of each grid and No. of bits each hash function contributes.
2. Calculate the number of hash functions that needs to be constructed per layer,
   No. of Hash functions per layer = No. of Buckets per layer/ No. of bits each hash function contributes.
3. Calculate the total no. of hash functions that needs to be constructed for the family, which is the previous step's value multiplied by the total number of layers.
4. Generate random unit vectors with the no. of dimensions equal to the no. of dimensions of each vector to be hashed present in the input file.
5. No. of such random vectors is equal to the no. of hash functions that needs to be generated for the LSH family.
6. Parse the given input file line by line and slice only the portion corresponding to the SIFT vectors excluding the key points position, Video, Frame and Cell Numbers.
7. For each vector in the input, calculate the projection of the vector on each of the random unit vectors generated.
8. Divide each projection by the grid size and take mod of 2^no. of bits each hash function is supposed to contribute. Convert this value to binary form.
9. Append the values corresponding to each hash function and create a string of 0's and 1's.
10. Divide this string into L equal partitions where L is equal to the no. of layers.
11. For each of the partitions, if, Length of the partition mod No. of bits each hash function contributes $\neq 1$
    - Remove the last 't' characters of the string, where t = (Length of the partition) % No. of bits each hash function contributes.
12. The index of the partitioned string gives the layer number and partition string converted to integer gives us the bucket no. for that layer.
13. Write the layer number, bucket number and key points position (x, y), Video, Frame and Cell Numbers to the file.
14. Repeat 11 to 13 'L' times.
15. Repeat 7 to 14 until we encounter the end of input file.

Task #6:

1. Parse the input arguments specified in the question and the path containing the demo videos
2. Find the number of layers used by reading the first few lines of the file
3. Find the list of vectors in the present in the query rectangle
4. For each of those vectors save the Bucket No. to which the vector is mapped for each of the layers
5. Parse the Input file line by line except for the lines corresponding to the input videos
6. Create a dictionary of dictionary of lists as the LSH data structure
7. Create a dictionary of lists to preserve the vectors contained in the query region, keys of the dictionary is vector index and values are the buckets to which the vector is mapped to in different layers
8. For each line, retrieve the Bucket ID which is of the form "Layer No._Bucket No" and FrameID which is of the form "Video No._Frame No."
9. The first level of the dictionary will have the Bucket IDs as keys and the next layer will have the Frame IDs as keys.
10. If the vector corresponding to the file happens to be in the query video, skip

11. Store the line index corresponding the original *.sift or *.spca file in the Data Structure
12. If the vector is in the region bounded by the rectangle, update the dictionary containing such vectors (step 6)
13. If EOF (End of File) is not reached go to 8
14. Calculate the number of vectors each frame in the query demo video set has in common with the query rectangle
15. Sort them in descending order of matches
16. Display them
17. Calculate the number of unique vectors, total number of vectors and index size in bytes and display them

System Requirements

Operating System: Windows
RAM: 8 GB
Software tools: Python, PyScripter, WinPython tool, Spyder
Libraries: Numpy 1.8, scikit-learn, Scipy, OpenCV[2] python extension
Additional tools: MATLAB

Related Work

Feature extraction is a varied approach and is used to bring in light the significant data out of a clustered structured or semi-structured data. Similarities between them; i.e. frames should be unique and different approaches can be used to determine similarity based on the intended use of the application and the consideration for user subjectivity. PCA (Principal Component Analysis) provides us a way to increase the variance among data sets; thus increasing the efficiency with which features are extracted. SIFT (the type of features we operated upon) are distinct when it comes to positioning an object in a whole set. Furthermore, by generating similarity graphs we can visualize the degree with which different frames are associated with each other and then efficiently judge the factors which influence similarities among them. When it comes to Page rank this algorithm relates to the structural significance of well-connected and densely clustered graphs, generating efficient results for extraction of main objects (webpages, significant frames). Its different forms specifically personalized version influences the importance of relevance among objects as they are considered the most when a random walk algorithm is portrayed. ASCOS & ASCOS++ also provide us with a deeper understanding of importance of taking weighted graphs into consideration because they could judge the amount of asymmetric access to a particular object when compared to some seed particulates. LSH (Locality Sensitive Hashing) implements the idea of Nearest neighbor search in Multi-dimensional index structure which better clusters the similar data into groups (buckets) and corresponding layers further reducing the amount of false positives and misses in generalized approaches. This approach is also very helpful in the search of similar objects inside a certain frame, inside a certain video query with a constant; which was efficiently performed during the project work [1][4][6][7].

Conclusion

The proliferation of video content on the Web makes similarity detection an indispensable tool in Web data management, searching, and navigation. Video similarity determination is a complex procedure and if content-based retrieval is added to it then it adds the additional overhead of user subjectivity to determine similarity for videos. Identifying and determining appropriate distance measures was a challenge as the same measures need to yield correct result even after dimensionality reduction using PCA. Graph visualization of similarity relations is an effective tool and the computations that it can entail are vast, it encompasses Page rank and Asymmetric similarity measures which provided us a way to better see the relevant & significant dynamic frames (which was the ultimate goal) further paving the way for LSH (Locality Sensitive Hashing) which effectively clusters data into chunks and upon implementing it the results were up to the satisfaction of the specific application area inside which we were operating.

Bibliography

[1] Alexandr Andoni and Piotr Indyk. "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions". Communications of the ACM, vol. 51, no. 1, 2008, pp. 117-122.

[2] David G. Lowe 'Distinctive Image Features from Scale-Invariant Keypoints', January 5, 2004, Page 2

[3] Github (Nearpy- Pipeline – Usage – Experiments – Storage) Accessed 30 Nov. 2016. URL: http://pixelogik.github.io/NearPy/

[4] Huang, S., Li, X., Candan, K. S., Sapino, M. L. (2016). Reducing seed noise in personalized PageRank. Social Network Analysis and Mining, 6(1), 1-25.

[5] Hung-Hsuan Chen and C. Lee Giles. 2015. ASCOS++: An Asymmetric Similarity Measure for Weighted Networks to Address the Problem of SimRank. ACM Trans. Knowl. Discov. Data 10, 2, Article 15 (October 2015)

[6] Sergey Brin , Lawrence Page, The anatomy of a large-scale hypertextual Web search engine, Computer Networks and ISDN Systems, v.30 n.1-7, p.107-117, April 1, 1998

[7] Wikipedia (Jaccard Index) (2016), Accessed 22 Oct 2016. URL: https://en.wikipedia.org/wiki/Jaccard_index

[8] Wikipedia (Cosine Similarity) (2016), Accessed 22 Oct 2016. URL: https://en.wikipedia.org/wiki/Cosine_similarity

[9] Wikipedia (Principal Component Analysis) (2016), Accessed 22 Oct. 2016. URL: https://en.wikipedia.org/wiki/Principal_component_analysis

[10] Wikipedia (Locality-sensitive hashing) (2016), Accessed 26 Nov. 2016. URL: https://en.wikipedia.org/wiki/Locality-sensitive_hashing

[11] Wikipedia (Page Rank) (2016), Accessed 26 Nov. 2016. URL: https://en.wikipedia.org/wiki/PageRank

[12] Wikipedia (Euclidian Distance) (2016), Accessed 01 Dec. 2016. URL: https://en.wikipedia.org/wiki/Euclidean_distance

[14] Kulis, Brian, Prateek Jain, and Kristen Grauman. "Fast similarity search for learned metrics." IEEE Transactions on Pattern Analysis and Machine Intelligence 31.12 (2009): 2143-2157.

[15] Jeff M. Phillips (University of Utah), Accessed 30 Nov. 2016. URL: https://www.cs.utah.edu/~jeffp/teaching/cs5955/L6-LSH.pdf

[16] Andoni, Alexandr, and Piotr Indyk. "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions." 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06). IEEE, 2006.

[17] Whole source (The Python Tutorial), Accessed 30 Nov. 2016. URL: https://docs.python.org/3/tutorial/datastructures.html+

[18] Wikipedia (Mahalanobis distance) (2016), Accessed 26 Nov. 2016. URL: https://en.wikipedia.org/wiki/Mahalanobis_distance

[19] Whole Numpy ref.(Eigen decomp., Covariance matrix, Dot prod.) (2016). Accessed 16 Oct 2016. URL: https://docs.scipy.org/doc/numpy/reference/generated/

Appendix

UTHARA KEERTHI
Implemented Task-3 Part (a) Page Rank algorithm, Task-4 Part(a) Personalized PageRank (RPR-2) algorithm and Part(b) modified ASCOS++ algorithm.

DARSHAN SHETTY
Implemented Task 2. Constructed OpenCV video output processing functions.

BHARATH KUMAR SURESH
Official Mascot.

DHANANJAYAN SANTHANAKRISHNAN
Implemented Task-3 Part (a) Initial naive version of PageRank Algorithm Task 3 (b) ASCOS Algorithm. Studied about the various Personalized versions of PageRank algorithm. Proposed the modified version of ASCOS++.

SHIVAM GAUR
Comprehensive research on Page rank (with personalized approach) & ASCOD/ASCOS measures and report creation.