CSE 515: Multimedia and Web Databases

Phase 3

Group 17
Aaryan Gupta
Fenil Madlani
Krisha Gala
Pranav Katariya
Radhika Ganapathy
Shivam Malviya

Abstract

In the final phase of this project, we perform image indexing using various clustering and classification algorithms such as Support Vector Machine, Personalized Page Rank and Decision Tree algorithms. Index structures were built using algorithms like Locality-Sensitive Hashing and VA-Files to carry out similar image search. In addition to this, we worked on relevance feedback systems to improve the nearest neighbour matches which helps account for the relevant and irrelevant results produced. Lastly, a query interface was built for the user to smoothly enter the query, retrieve results and give feedback which then produced revised results.

***Keywords*—** image features, image retrieval, indexing, vector models, classification, SVM, Decision Tree, Personalized Page Rank, clustering, LSH, VA Files, relevance feedback.

1. Introduction

In the first phase of this project, we dealt with concepts of feature vectors where we extracted features based on models like Color moments, Extended Local Binary Patterns, and Histogram of Oriented Gradients to generate feature descriptors for images which were then used for comparing purposes using various similarity distance functions like the Euclidean distance, Manhattan Distance, etc.

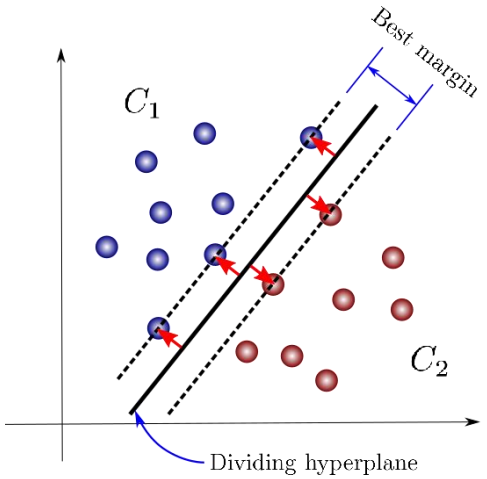
In the second phase, we delved deeper into the concepts of multimedia retrieval where we addressed the issue of the dimensionality curse. [10] Dimensionality curse is when multimedia database systems cannot manage more than a handful of facets of the multimedia data simultaneously. We were given a dataset of facial grayscale images. The images were taken during different times, varying lighting and facial expressions. We studied and implemented the dimensionality reduction techniques like PCA, SVD, LDA, and K-Means on the given image dataset and also applied page ranking algorithms to the same.

In the final phase of this project, we study and implement classification algorithms like Support Vector Machines, Decision Trees and Personalized Page Rank. We also work with indexing algorithms like Locality-Sensitive Hashing and Vector Approximation Files to build indexing tools for efficient similar image retrieval. In addition to this we account for user feedback by building relevant feedback systems for the classification algorithms. Lastly, we build a query interface for the user to smoothly run the above-mentioned tasks.

1.1. Terminology

1.1.1. Support Vector Machines

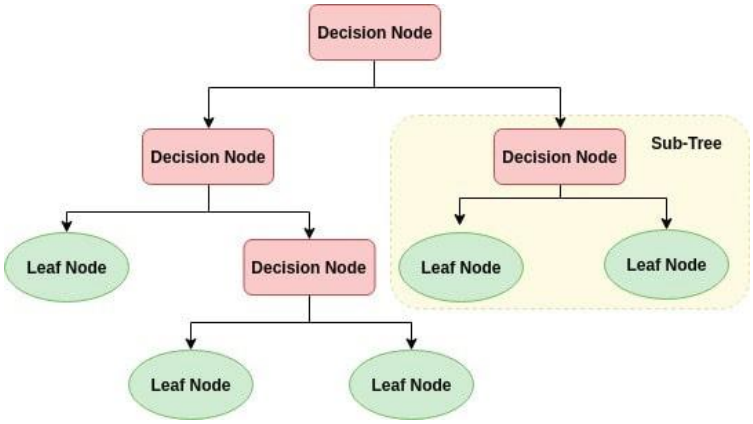
Support-vector machines are supervised learning models with associated learning algorithms that analyse data for classification and regression analysis. [1] It constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. A good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class.



1.1.2. Decision Tree

A Decision Tree is a is a hierarchal classification algorithm. It can be used for predicting categorical and continuous variables. Like SVM, it can be used for regression or ranking as well.[2] There are two types of trees: classification decision trees and regression decision trees. The biggest advantage of decision trees is that they make it very easy to interpret and visualize nonlinear data patterns. Another advantage of classification decision trees is the possibility to improve their accuracy by setting the logic for the branches split.

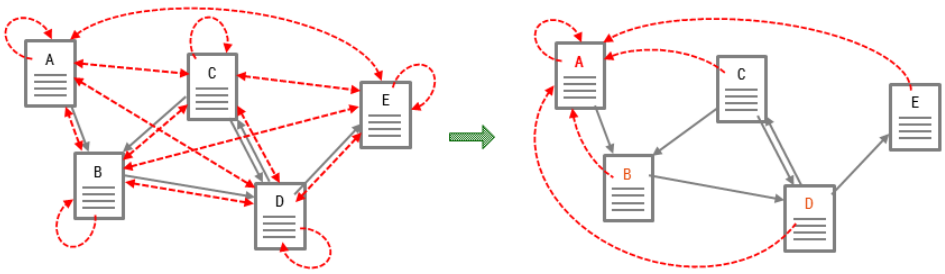
A decision tree consists of three types of nodes:
-Decision nodes – typically represented by squares
-Chance nodes – typically represented by circles
-End nodes – typically represented by triangles



1.1.3. Personalized Page Rank

Personalized PageRank (PPR) is a widely used node proximity measure in graph mining and network analysis. It is a standard tool for finding vertices in a graph that are most relevant to a query or user.[3] To personalize PageRank, one adjusts node weights or edge weights that determine teleport probabilities and transition probabilities in a random surfer model.

$$\boldsymbol{R}_{i+1} = (1-\alpha)(S^T + w \times d^T)\boldsymbol{R}_i + \alpha \begin{bmatrix} \frac{1}{N} \\ \vdots \\ \frac{1}{N} \end{bmatrix} \Rightarrow \boldsymbol{R}_{i+1} = (1-\alpha)(S^T + w \times d^T)\boldsymbol{R}_i + \alpha \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$



1.1.4. LSH

Locality-Sensitive Hashing (LSH) is an algorithm for solving approximate/exact “Near Neighbour Search" in high dimensional spaces. LSH can be referred to as a technique that hashes similar input items into the same “buckets” with a high probability using a hash function. It reduces the effect of dimensionality curse. [4] The key idea is to hash the points using several hash functions to ensure that for each function the probability of collision is much higher for objects that are close to each other than for those that are far apart. This will enable us to determine near neighbours by hashing the query point and retrieving elements stored in buckets containing that point.

1.1.5. Vector-Approximation Files

The Vector Approximation File approach based on uniform scalar quantization of feature vectors, and is a powerful technique that scales well with size and dimensionality of the data-set. VA-File partitions the space into hyper-rectangular cells, to obtained a quantized approximation for the data that reside inside the cells. Non-empty cell locations are encoded into bit strings and stored in a separate approximation file, on the hard-disk. During a nearest neighbour search, the vector approximation file is sequentially scanned and upper and lower bounds on the distance from the query vector to each cell are estimated. The bounds are used to prune irrelevant cells. The final set of candidate vectors are then read from the hard disk and the exact nearest neighbours are determined.[7]

1.1.6. Relevance Feedback

Relevance feedback is a feature of some information retrieval systems. The idea behind relevance feedback is to take the results that are initially returned from a given query, to gather user feedback, and to use information about whether or not those results are relevant to perform a new query.[8] Accordingly results are modified and returned after taking feedback into consideration.

1.1.7. Euclidean Distance

The Euclidean distance is commonly used for measuring distances between points in the 3D space we are living in. It is in fact the Minkowski distance of order 2. This distance metric is most commonly used for similarity measurement in image retrieval because of its efficiency and effectiveness. It measures the distance between two vectors of images by calculating the square root of the sum of the squared absolute differences.

$$\Delta_{Euc}(\vec{v}_q, \vec{v}_o) = \Delta_{Mink,2}(\vec{v}_q, \vec{v}_o) = \left(\sum_{i=1}^{n} |q_i - o_i|^2 \right)^{1/2},$$

1.2. Goal Description

Task 1:
In this task, we have to implement a program in which we are given a folder of images, one of the three feature models, and a user specified value of k. We have to compute k latent semantics (if not already computed and stored), and given a second folder of images, it should associate **X** labels to the images in the second folder using the classifier selected by the user. The classifiers to be implemented are - SVM classifier, Decision-Tree classifier, Personalized Page Rank. We also have to compute and print false positive and miss rates.

Task 2:
In this task, we have to implement a program in which we are given a folder of images, one of the three feature models, and a user specified value of k. We have to compute k latent semantics (if not already computed and stored), and given a second folder of images, it should associate **Y** labels to each image in the second folder using the classifier selected by the user. The classifiers to be implemented are - SVM classifier, Decision-Tree classifier, Personalized Page Rank. We also have to compute and print false positive and miss rates.

Task 3:
In this task, we have to implement a program in which we are given a folder of images, one of the three feature models, and a user specified value of k. We have to compute k latent semantics (if not already computed and stored), and given a second folder of images, it should associate **Z** labels to each image in the second folder using the classifier selected by the user. The classifiers to be implemented are - SVM classifier, Decision-Tree classifier, Personalized Page Rank. We also have to compute and print false positive and miss rates.

Task 4:
In this task, we have to implement a Locality Sensitive Hashing (LSH) tool, which takes as input the number of layers (L), the number of hashes per layer (κ) and a set of vectors (generated by other tasks) and creates an in-memory index structure containing the given set of vectors. In addition to this, we have to implement similar image search using this index structure. Given a folder of images and one of the three feature models, the images are stored in an LSH data structure (the program also outputs the size of the index structure in bytes). For any given image and "t", the index tool returns the "t" most similar images. The program should also return the number of buckets searched, the unique and overall number of images considered, false positive and miss rates.

Task 5:
In this task we have to implement a VA-file index tool and conduct nearest neighbour search operations. We will be given a parameter "b" denoting the number of bits per dimensions used for compressing the vector data and a set of vectors (generated by other tasks) as input. The program creates an in-memory index structure containing the indexes of the given set of vectors. The program also returns the size of the index structure in

bytes. We then have to implement similar image search using this index structure. Given a folder of images and one of the three feature models, the images will be stored in a VA-file data structure (the program also outputs the size of the index structure in bytes). For any given image and “t”, the tool returns the “t” most similar images. The program should also return the number of buckets searched, the unique and overall number of images considered, false positive and miss rates.

Task 6:
In this task we have to implement a Decision-Tree based relevance feedback system to improve nearest neighbour matches. This enables the user to label some of the results returned by the search task as relevant or irrelevant and then returns a new set of ranked results, either by revising the query or by re-ordering the existing results.

Task 7:
In this task we have to implement a SVM based relevance feedback system to improve nearest neighbour matches. This enables the user to label some of the results returned by the search task as relevant or irrelevant and then returns a new set of ranked results, either by revising the query or by re-ordering the existing results.

Task 8:
In this task we have implement a query interface. This allows the user to provide a query and relevant query parameters (including how many results to be returned). Query results are presented to the user in decreasing order of matching. The result interface should also allow the user to provide positive and/or negative feedback for the ranked results returned by the system. User feedback is then taken into account and a new set of ranked results are returned.

1.3.Assumptions

- Datasets provided are processed and can be used without further computation.
- Datasets given are consistent in terms of image ID, subject and type labels.
- All images to be used as input should be given in a folder labelled “images”.
- The query image should be provided in the format folder-name/image-name.
- The latent semantic files obtained in the previous phase for the various functions of PCA, SVD and LDA are correct.
- In the implementation of SVM, the number of clusters should be greater than equal to 2, to carry out classification task.
- Feedback given in Task 6 and Task 7 should be greater than 0 to take into account irrelevance/relevance feedback
- All user inputs are in accordance with the task input formats mention in the execution section.

2. Description of Proposed Solution and Implementation

Task 1, Task 2, Task 3

In this task, we are given an input folder of images for training, feature models and k from the user. We compute the data matrix along with accessing the labels. We then transform the data matrix if needed according to the latent space. We shuffle this so that we get randomized training data which we then pass to the classifier for fitting. The test folder is given by the user which is used to extract the true labels. We pass this test data to the model for prediction. We get the predictive labels as output and then compare this with true label from the training data to generate a confusion matrix. We then calculate false positives and miss rates from the generated confusion matrix.

Following are the implementations of the classifiers:

In **Support Vector Machines (SVM)**, we perform multiclass classification. We take one class as the positive class and the rest as a negative class. We then have two clusters. We have to find a separator between the 2 clusters based on the equation of a plane such that $wx + b > 0$ for the positive class and $wx + b < 0$ for the negative class. "w" and "b" is modified based on these conditions and learning rate to get the final separator values for the positive class. We do this for every positive class. For prediction we compare a test image with every separator. If it gives a positive value we append it to a predicted list. If there are multiple predicted classes, we calculate the distance of the test image with the separator of predicted classes. The minimum distance separator is the final predicted class.

In **Decision Tree classifier**, we first start with the Node class which stores the data about the feature, threshold, data going left and right, information gain, and the leaf node value. All of them are initially set to None. The root and decision nodes contain values for everything besides the leaf node value, and the leaf node contains the opposite. The constructor holds values for the hyperparameter i.e., the minimum number of samples required to split a node and the maximum depth of a tree. Both are used in recursive functions as exit conditions. We then have a function for entropy which calculates the impurity of an input vector. We calculate the information gain value of a split between a parent and two children. A function calculates the best splitting parameters for input features X and a target variable Y. It does so by iterating over every column in X and every threshold value in every column to find the optimal split using information gain. We then recursively build a decision tree till the stopping criteria is met. We store the built tree and then predict the classification on the query.

In **Personalized Page** Rank (PPR), we first make the similarity matrix. Along with this we insert the given query into the similarity matrix. The similarity graph is made using the method as described in *Phase 2 Task 9*. The class "Node" contains additional information about the PageRank score for each node and the distances between all the children and their parents. We then calculate the page rank scores for each node using the Personalised PageRank algorithm. The page rank score for each node is initially set to 1/ (no of subjects). We then update the scores of all the nodes one by one. This is iterated until the page rank score converges for each node. The page rank score for every node in each iteration is calculated as follows. The node's page rank score is calculated using its parents' (in-neighbours) page ranks. To make this personalized page rank, we restrict the teleportation vector to contain values for only the query. The node that corresponds to the highest page rank score is the result of the given query.

--Task 1--

```
PS C:\Users\Radhika\Desktop\ASU\CSE 515\PROJECT\asu-cse515-phase3\Code> python p3_task1.py 500 elbp 5 100 dtree
500_elbp.csv found!
Shape of data matrix:  (477, 576)
No. of labels:  477
Existing latent semantics and train matrix not found! Creating and saving them...
Latent Semantics shape:  (576, 50)
Transformed data matrix shape:  (477, 50)
Latent Semantics File:  500_elbp_50_LS.csv
Training model now...
dtree  model training completed!
100_elbp.csv found!
Shape of data matrix:  (100, 576)
No. of labels:  100
True Labels | Predicted Labels
cc       |      cc
cc       |      cc
cc       |      cc
cc       |      cc
cc       |      cc
cc       |      cc
con      |      con
con      |      con
con      |      con
con      |      con
con      |      con
con      |      con
con      |      con
emboss   |      emboss
emboss   |      emboss
emboss   |      emboss
emboss   |      emboss
emboss   |      emboss
emboss   |      emboss
emboss   |      original
emboss   |      emboss
jitter   |      noise02
```

```
The accuracy is :

0.13
Confusion Matrix:
 [[0 1 0 0 0 1 5 0 0 0]
 [0 0 0 0 0 8 4 0 0 0]
 [0 0 0 0 0 0 8 0 0 0]
 [0 0 0 1 0 2 9 0 0 0]
 [0 0 0 0 0 2 4 0 0 0]
 [0 0 0 0 1 7 7 0 0 0]
 [0 0 0 0 0 3 5 0 0 0]
 [0 0 0 0 0 2 5 0 0 0]
 [0 0 0 0 0 4 8 0 0 0]
 [0 0 0 0 0 4 9 0 0 0]]
False Positive Rate: 0.09666666666666666
Misses Rate: 0.87
Task Completed!
```

--Task 2--

```
Enter space-separated values of 'train folder', 'feature model', 'k':
500 elbp 5
Enter space-separated values of 'test folder', 'classifier':
100 dtree
500_elbp.csv found!
Shape of data matrix:  (477, 576)
No. of labels:  477
Existing latent semantics and train matrix found!
Training model now...
dtree  model training completed!
100_elbp.csv found!
Shape of data matrix:  (100, 576)
No. of labels:  100
True Labels | Predicted Labels
1           |      31
13          |      17
21          |      28
22          |      22
35          |      17
4           |      15
11          |      32
17          |      17
19          |      32
25          |      32
27          |      32
33          |      17
39          |      39
1           |      40
10          |      40
11          |      34
```

```
The accuracy is :

0.11
Confusion Matrix:
 [[1 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [1 0 1 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
False Positive Rate: 0.0234210526315789 47
Misses Rate: 0.89
Task Completed!
```

```
svm  model training completed!
100_elbp.csv found!
Shape of data matrix:  (100, 576)
No. of labels:  100
True Labels | Predicted Labels
1           |      9
13          |      9
21          |      9
22          |      9
35          |      9
4           |      9
11          |      9
17          |      9
19          |      9
25          |      9
27          |      9
33          |      9
39          |      9
1           |      9
10          |      9
11          |      9
13          |      9
21          |      9
23          |      9
31          |      9
38          |      9
10          |      32
11          |      32
15          |      32
16          |      32
19          |      32
```

```
The accuracy is :

0.01
Confusion Matrix:
 [[0 0 0 ... 0 0 4]
 [0 0 0 ... 0 0 2]
 [0 0 0 ... 0 0 5]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]
 [0 0 0 ... 0 0 1]]
False Positive Rate: 0.028285714285714286
Misses Rate: 0.99
Task Completed!
```

--Task 3—

```
The accuracy is :

0.13
Confusion Matrix:
 [[0 1 0 0 0 1 5 0 0 0]
 [0 0 0 0 0 8 4 0 0 0]
 [0 0 0 0 0 0 8 0 0 0]
 [0 0 0 1 0 2 9 0 0 0]
 [0 0 0 0 0 2 4 0 0 0]
 [0 0 0 0 1 7 7 0 0 0]
 [0 0 0 0 0 3 5 0 0 0]
 [0 0 0 0 0 2 5 0 0 0]
 [0 0 0 0 0 4 8 0 0 0]
 [0 0 0 0 0 4 9 0 0 0]]
False Positive Rate: 0.09666666666666666
Misses Rate: 0.87
Task Completed!
```

```
The accuracy is :

0.83
Confusion Matrix:
 [[ 6  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  7  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  7  0  0  0  0  1  0  0  0  0]
 [ 0  0  0  5  0  1  3  0  0  0  0  0]
 [ 0  0  1  0  6  0  0  0  0  0  0  0]
 [ 0  0  0  3  0 11  0  0  0  0  0  0]
 [ 0  0  0  1  0  1  5  0  0  0  0  0]
 [ 0  0  0  0  1  0  0  7  0  0  0  0]
 [ 0  1  0  0  0  0  0  0  9  0  0  0]
 [ 0  0  0  0  0  0  0  1  0  5  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  8  0]
 [ 0  2  0  0  0  0  1  0  0  0  0  7]]
False Positive Rate: 0.015454545454545455
Misses Rate: 0.17
Task Completed!
```

Task 4

In **Locality Sensitive Hashing,** we are given the number of bits, the image folder, feature model and a value for k. We then additionally ask for the number of layers (L) in the LSH index structure, the number of hash functions in each layer, query image and a value "t" (number of similar images to be retrieved) as input from the user. First, a random vector is generated which is scalar multiplied with the image. If the product is positive then "1" is assigned otherwise "0" is assigned. We build the LSH index structure using these randomly generated hash functions which generates a binary output of 1/0. This is repeated for "L" layers. Every image is passed through the hash functions which in turn generates a binary representation for the image i.e., k*L binary string. This is done for all the images in the folder. We create a dictionary where the key corresponds to the binary string and the value corresponds to the images that matches the binary string. This dictionary forms the index structure for the LSH algorithm. We then produce a binary string for the query image by passing it through the LSH structure. This binary string is searched for in the LSH index structure and the values obtained correspond to the nearest neighbours for that particular query image. If the number of neighbours produced is less that "t", we tune the algorithm by reducing the value of "k" by 1. This is repeated till we get at least "t" nearest neighbours in our search. We then compute the Euclidean distance between the query image and the nearest neighbours obtained to find the top "t" images. The number of buckets searched is the number of keys in the LSH index structure. The unique and overall images considered form the nearest neighbours. We then perform a sequential scan on the image folder to get the "t" similar images. This corresponds to the true positives. The correct images are those that appear both in sequential scan as well as LSH. We then calculate false positives and miss rates.

$$miss\ rate = \frac{t - number\ of\ correct\ images}{t}$$

$$false\ positives\ rate = \frac{number\ of\ nearest\ neighbours - number\ of\ correct\ images}{t}$$

--Task 4--

```
100_elbp.csv found!
Shape of data matrix:  (100, 576)
No. of labels:  100
Size of LSH structure:  1176  bytes
Ranked Nearest neighbours:  {1: 'image-cc-1-8.png', 2: 'image-poster-35-7.png', 3: 'image-original-27-8.png', 4: 'image-jitter-10-1.png'}

Buckets searched at this step: 40
Not enough nearest numbers found ! Optimizing! Decreasing K to :  7
Size of LSH structure:  640  bytes
Ranked Nearest neighbours:  {1: 'image-cc-1-8.png', 2: 'image-poster-35-7.png', 3: 'image-noise01-28-8.png', 4: 'image-original-27-8.png', 5: 'image-jitter-10-1.png'}

Buckets searched at this step: 35
Not enough nearest numbers found ! Optimizing! Decreasing K to :  6
Size of LSH structure:  640  bytes
Ranked Nearest neighbours:  {1: 'image-cc-1-8.png', 2: 'image-poster-35-7.png', 3: 'image-noise01-28-8.png', 4: 'image-original-27-8.png', 5: 'image-jitter-10-1.png'}

Buckets searched at this step: 30
Not enough nearest numbers found ! Optimizing! Decreasing K to :  5
Size of LSH structure:  360  bytes
Ranked Nearest neighbours:  {1: 'image-cc-1-8.png', 2: 'image-poster-32-5.png', 3: 'image-poster-34-8.png', 4: 'image-poster-31-9.png', 5: 'image-poster-37-3.png', 6: 'im
4-10.png', 9: 'image-noise01-28-8.png', 10: 'image-noise01-20-3.png', 11: 'image-original-27-8.png', 12: 'image-jitter-10-1.png', 13: 'image-con-11-5.png', 14: 'image-noi
9.png', 17: 'image-emboss-23-5.png', 18: 'image-stipple-38-1.png', 19: 'image-stipple-38-9.png', 20: 'image-con-39-2.png', 21: 'image-stipple-18-4.png', 22: 'image-stippl
}
```

```
Buckets searched at this step: 25
Enough nearest neighbors found! Here are top  20 :
Rank  1 :  image-cc-1-8.png
Rank  2 :  image-poster-32-5.png
Rank  3 :  image-poster-34-8.png
Rank  4 :  image-poster-31-9.png
Rank  5 :  image-poster-37-3.png
Rank  6 :  image-poster-35-7.png
Rank  7 :  image-poster-15-2.png
Rank  8 :  image-poster-4-10.png
Rank  9 :  image-noise01-28-8.png
Rank  10 :  image-noise01-20-3.png
Rank  11 :  image-original-27-8.png
Rank  12 :  image-jitter-10-1.png
Rank  13 :  image-con-11-5.png
Rank  14 :  image-noise02-31-8.png
Rank  15 :  image-stipple-21-8.png
Rank  16 :  image-stipple-8-9.png
Rank  17 :  image-emboss-23-5.png
Rank  18 :  image-stipple-38-1.png
Rank  19 :  image-stipple-38-9.png
Rank  20 :  image-con-39-2.png

----------Total number of buckets searched:  130 ----------
```

```
Total number of unique and overall images considered (it's the same):  24
------------------------------------------------------------------------
Performing sequential scan to identify true top neighbors...
Top 20 true neighbors from sequential scan:
Rank  1 :  image-cc-22-10.png
Rank  2 :  image-cc-1-8.png
Rank  3 :  image-cc-21-3.png
Rank  4 :  image-cc-13-4.png
Rank  5 :  image-cc-4-2.png
Rank  6 :  image-cc-35-1.png
Rank  7 :  image-poster-32-5.png
Rank  8 :  image-poster-34-8.png
Rank  9 :  image-poster-31-9.png
Rank  10 :  image-poster-37-3.png
Rank  11 :  image-poster-35-7.png
Rank  12 :  image-original-1-2.png
Rank  13 :  image-poster-15-2.png
Rank  14 :  image-poster-12-10.png
Rank  15 :  image-poster-11-7.png
Rank  16 :  image-neg-1-2.png
Rank  17 :  image-poster-4-10.png
Rank  18 :  image-poster-7-5.png
Rank  19 :  image-noise01-28-8.png
Rank  20 :  image-neg-31-3.png
Correct matches:  9
{'image-cc-1-8.png', 'image-poster-31-9.png', 'image-poster-32-5.png', 'image-poster-37-3.png',
------------------------------------------------------------------------
False Positive Rate:  0.75
Miss Rate:  0.55
------------------------------------------------------------------------
```

Task 5

In task 5, we are given the number of bits, image folder, feature model and k as input. In addition to this we also take query image and a value “t” (number of similar images to be retrieved) as input from the user. Based on the number of bits given by the user, we partition each dimension into 2^b+1 partition points. For every image in the given folder, we check whether the given dimension of the image lies in a particular region and accordingly assign a binary value to it. Similarly, each dimension of the image is represented in the form of binary bits which forms the vector approximation file for that image. We then compute the VA file for the given query image. To find the “t” nearest images we use the VA-SSA algorithm which finds the lower bound for each image. We then calculate L3 norm distance between the lower bound of the images in the folder and our query image and return top k images in the filtered region.

--Task 5--

```
PS D:\MS CS ASU\SEM 1\CSE 515 MWDB\Phase 3\Repo\asu-cse515-phase3\Code> python p3_task5.py
Enter space-separated values of 'query image', 'bits', 'input folder', 'feature model', 't':
all/image-cc-1-1.png 12 100 elbp 3
100_elbp.csv found!
Shape of data matrix:  (100, 576)
No. of labels:  100

The nearest images are :

Rank  1  : image-cc-1-8.png
Rank  2  : image-cc-13-4.png
Rank  3  : image-poster-7-5.png
--------------------------------------------------------------------------------------------------
Rank  1  : image-cc-1-8.png
Rank  2  : image-cc-13-4.png
Rank  3  : image-cc-21-3.png
Rank  4  : image-con-25-8.png
Rank  5  : image-original-1-2.png
Rank  6  : image-poster-7-5.png
Size of VA File structure:  4696  bytes
------------------------------------------------------------------------
Performing sequential scan to identify true top neighbors...
Top 3 true neighbors from sequential scan:
Rank  1 :  image-cc-1-8.png
Rank  2 :  image-cc-13-4.png
Rank  3 :  image-cc-21-3.png
Correct matches:  2
{'image-cc-13-4.png', 'image-cc-1-8.png'}
------------------------------------------------------------------------
False Positive Rate:  1.3333333333333333
Miss Rate:  0.3333333333333333
------------------------------------------------------------------------
```

Task 6, Task 7

We are given the nearest neighbours as input, along with some of them labelled as relevant and irrelevant. We now have two classes. We train the classifier Decision Tree for task 6 and Support Vector Machines for task 7. We have to train the classifier based on the given labelled nearest neighbour points. We then have to classify the remaining nearest neighbours as relevant or irrelevant. The classifier model will now have 2 classes – relevant and irrelevant. We have to predict the remaining neighbour points on the trained classifier model. We will then have all the nearest neighbours divided into 2 classes – relevant and irrelevant. In task 6, we rearrange and re-rank the relevant points at the top and the irrelevant points at the bottom to return the new re-ranked nearest neighbours. In task 7, we find the distance of all these neighbours with the separator in order to get the new ranks of the nearest neighbours which is fed back into Task 8.

--Task 6—

```
Enter space-separated index numbers for relevant images:
1 2 3 4 5
Enter space-separated index numbers for irrelevant images:
6 8 9
--------------------------------------------------------------------------------
Which classifier do you want to use for relevance feedback? (svm/dtree):
dtree
--------------------------------------------------------------------------------
Rank  1 : image-cc-1-8.png
Rank  2 : image-cc-13-4.png
Rank  3 : image-cc-35-1.png
Rank  4 : image-cc-21-3.png
Rank  5 : image-cc-4-2.png
Rank  6 : image-cc-22-10.png
Rank  7 : image-poster-4-10.png
Rank  8 : image-poster-12-10.png
Rank  9 : image-poster-15-2.png
Rank  10 : image-poster-37-3.png
Rank  11 : image-poster-34-8.png
Rank  12 : image-poster-35-7.png
Rank  13 : image-poster-31-9.png
Rank  14 : image-poster-32-5.png
Rank  15 : image-poster-11-7.png
Rank  16 : image-con-19-8.png
Rank  17 : image-poster-7-5.png
Rank  18 : image-neg-1-2.png
Rank  19 : image-con-27-5.png
Rank  20 : image-con-25-8.png
```

Task 8

Is essentially an interface for the user to specify a query image and relevant query parameters which include number of nearest neighbours required, feature model, the dataset to build the vector space on, dimensionality reduction if needed, the index structure (and its relevant parameters) to get the nearest neighbours? Based on the obtained nearest neighbours from the specified index structures, the user can then provide indexes of images which are relevant and irrelevant. This extra information is accounted for by feeding them into relevance feedback classifiers (SVM/DT) and the new set of reranked nearest neighbours is displayed to the user based on the given relevance feedback.

--Task 7 –Task 8—

```
Enter Index-tool (LSH/VA):
LSH
Enter (space-separated) index tool parameters:
#Layers(L) #Hash-per-layer(K)
5 8
Size of LSH structure:  360  bytes
Ranked Nearest neighbours:  {1: 'image-cc-1-8.png', 2: 'image-cc-13-4.png', 3: 'image-cc-35-1.png', 4: 'image-cc-21-3.png',
 5: 'image-cc-4-2.png', 6: 'image-cc-22-10.png', 7: 'image-neg-1-2.png'}

Buckets searched at this step: 40
Not enough nearest numbers found ! Optimizing! Decreasing K to :  7
Size of LSH structure:  360  bytes
Ranked Nearest neighbours:  {1: 'image-cc-1-8.png', 2: 'image-cc-13-4.png', 3: 'image-cc-35-1.png', 4: 'image-cc-21-3.png',
 5: 'image-cc-4-2.png', 6: 'image-cc-22-10.png', 7: 'image-neg-1-2.png'}

Buckets searched at this step: 35
Not enough nearest numbers found ! Optimizing! Decreasing K to :  6
Size of LSH structure:  360  bytes
Ranked Nearest neighbours:  {1: 'image-cc-1-8.png', 2: 'image-cc-13-4.png', 3: 'image-cc-35-1.png', 4: 'image-cc-21-3.png',
 5: 'image-cc-4-2.png', 6: 'image-cc-22-10.png', 7: 'image-neg-1-2.png'}

Buckets searched at this step: 30
Not enough nearest numbers found ! Optimizing! Decreasing K to :  5
Size of LSH structure:  360  bytes
Ranked Nearest neighbours:  {1: 'image-cc-1-8.png', 2: 'image-cc-13-4.png', 3: 'image-cc-35-1.png', 4: 'image-cc-21-3.png',
 5: 'image-cc-4-2.png', 6: 'image-cc-22-10.png', 7: 'image-neg-1-2.png'}
```

```
Buckets searched at this step: 25
Not enough nearest numbers found ! Optimizing! Decreasing K to :  4
Size of LSH structure:  232  bytes
Ranked Nearest neighbours:  {1: 'image-cc-1-8.png', 2: 'image-cc-13-4.png', 3: 'image-cc-35-1.png', 4: 'image-cc-21-3.png',
 5: 'image-cc-4-2.png', 6: 'image-con-25-8.png', 7: 'image-cc-22-10.png', 8: 'image-con-27-5.png', 9: 'image-neg-1-2.png',
10: 'image-poster-7-5.png', 11: 'image-con-19-8.png', 12: 'image-poster-11-7.png', 13: 'image-poster-32-5.png', 14: 'image-
poster-31-9.png', 15: 'image-poster-35-7.png', 16: 'image-poster-34-8.png', 17: 'image-poster-37-3.png', 18: 'image-poster-
15-2.png', 19: 'image-poster-12-10.png', 20: 'image-poster-4-10.png'}
```

```
 Buckets searched at this step: 20
 Enough nearest neighbors found! Here are top  20 :
 Rank  1 :  image-cc-1-8.png
 Rank  2 :  image-cc-13-4.png
 Rank  3 :  image-cc-35-1.png
 Rank  4 :  image-cc-21-3.png
 Rank  5 :  image-cc-4-2.png
 Rank  6 :  image-con-25-8.png
 Rank  7 :  image-cc-22-10.png
 Rank  8 :  image-con-27-5.png
 Rank  9 :  image-neg-1-2.png
 Rank  10 :  image-poster-7-5.png
 Rank  11 :  image-con-19-8.png
 Rank  12 :  image-poster-11-7.png
 Rank  13 :  image-poster-32-5.png
 Rank  14 :  image-poster-31-9.png
 Rank  15 :  image-poster-35-7.png
 Rank  16 :  image-poster-34-8.png
 Rank  17 :  image-poster-37-3.png
 Rank  18 :  image-poster-15-2.png
 Rank  19 :  image-poster-12-10.png
 Rank  20 :  image-poster-4-10.png

 ----------Total number of buckets searched:  150 ----------

 Total number of unique and overall images considered (it's the same):  20
 ------------------------------------------------------------------------------
```

```
Enter space-separated index numbers for relevant images:
1 2 3 4 5
Enter space-separated index numbers for irrelevant images:
6 8 9
--------------------------------------------------------------------------------
Which classifier do you want to use for relevance feedback? (svm/dtree):
svm
Finding separator for label:  irrelevant
--------------------------------------------------------------------------------
Rank  1 : image-cc-35-1.png
Rank  2 : image-cc-21-3.png
Rank  3 : image-cc-13-4.png
Rank  4 : image-cc-22-10.png
Rank  5 : image-cc-1-8.png
Rank  6 : image-cc-4-2.png
Rank  7 : image-neg-1-2.png
Rank  8 : image-poster-7-5.png
Rank  9 : image-con-25-8.png
Rank  10 : image-poster-31-9.png
Rank  11 : image-poster-35-7.png
Rank  12 : image-poster-32-5.png
Rank  13 : image-con-19-8.png
Rank  14 : image-con-27-5.png
Rank  15 : image-poster-11-7.png
Rank  16 : image-poster-12-10.png
Rank  17 : image-poster-34-8.png
Rank  18 : image-poster-15-2.png
Rank  19 : image-poster-4-10.png
Rank  20 : image-poster-37-3.png
--------------------------------------------------------------------------------
```

3. Interface Specification

This phase of the project uses the basic command line interface for taking the inputs and displaying outputs of every task. The inputs are passed as command line arguments to each task as specified in the next section under execution instructions. The outputs are displayed in stdout and screenshots of these outputs are displayed in the section above. All the task files have to be run using python 3.6 or above.

Data Matrix created during the execution of tasks is stored in the format:
image-folder-name_feature-model.csv

Transformed Data Matrix after dimensionality reduction is stored in the format:
image-folder-name_feature-model_k_WT.csv

Latent Semantic Files created during the execution of tasks is stored in the format:
image-folder-name_feature-model_k_LS.csv

4. System Requirements and Execution Instructions

In this phase of the project, we use Python as our programming language and MongoDB to store our database. In addition to this, we utilize several libraries such as NumPy, Sys, PIL, Pymongo, sklearn, SciPy, etc. to carry out the given tasks.

4.1. Environment Set up

We have used the PyCharm IDE. PyCharm is a programming integrated development environment that focuses on the Python programming language. Make sure Python 3.6 or higher version of Python is downloaded in your system. Based on the operating system of the computer, the corresponding version of PyCharm or a similar IDE can be easily downloaded from the website.

Ensure that you install all the necessary packages. Some of the packages installed are listed below:

NumPy - A package providing high-performance multidimensional array object

Sys - The sys module in Python provides various functions and variables that are used to manipulate different parts of the Python runtime environment.

OS - The OS module in Python provides functions for interacting with the operating system.

PyMongo - The PyMongo library allows interaction with the MongoDB database through Python.

PIL - Python Imaging Library is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different images of different formats.

Sklearn - An open-source library providing efficient tools for various classification, decomposition, regression, and clustering algorithms.

4.2.Execution

Run the commands in the following manner for each task, changing the command-line arguments as needed.

Task 1: python p3_task1.py

```
PS C:\Users\Radhika\Desktop\ASU\CSE 515\PROJECT\asu-cse515-phase3\Code> python .\p3_task1.py
500_elbp.csv found!
Shape of data matrix:  (477, 576)
No. of labels:  477
Existing latent semantics and train matrix found!
Latent Semantics File:  500_elbp_50_LS.csv
Training model now...
```

Task 2: python p3_task2.py

```
PS D:\MS CS ASU\SEM 1\CSE 515 MWDB\Phase 3\Repo\asu-cse515-phase3\Code> python p3_task2.py
Enter space-separated values of 'train folder', 'feature model', 'k':
2000 elbp 5
Enter space-separated values of 'test folder', 'classifier':
100 svm
2000_elbp.csv found!
Shape of data matrix:  (1929, 576)
No. of labels:  1929
Existing latent semantics and train matrix not found! Creating and saving them...
Latent Semantics shape:  (576, 5)
Transformed data matrix shape:  (1929, 5)
Training model now...
Finding separator for label:  16
Finding separator for label:  30
Finding separator for label:  11
Finding separator for label:  4
Finding separator for label:  20
Finding separator for label:  25
Finding separator for label:  29
Finding separator for label:  3
Finding separator for label:  5
Finding separator for label:  15
Finding separator for label:  18
Finding separator for label:  35
Finding separator for label:  21
Finding separator for label:  12
Finding separator for label:  39
Finding separator for label:  34
Finding separator for label:  22
Finding separator for label:  13
Finding separator for label:  2
```

Task 3: python p3_task3.py

```
PS D:\MS CS ASU\SEM 1\CSE 515 MWDB\Phase 3\Repo\asu-cse515-phase3\Code> python p3_task3.py
Enter space-separated values of 'train folder', 'feature model', 'k':
1000 elbp 10
Enter space-separated values of 'test folder', 'classifier':
100 svm
```

Task 4: python p3_task4.py

```
PS C:\Users\Radhika\Desktop\ASU\CSE 515\PROJECT\asu-cse515-phase3\Code> python p3_task4.py
100_elbp.csv found!
Shape of data matrix:  (100, 576)
No. of labels:  100
Size of LSH structure:  1176  bytes
Ranked Nearest neighbours:  {1: 'image-cc-1-8.png', 2: 'image-poster-35-7.png', 3: 'image-original-27-8.png', 4: 'image-jitter-10-1.png'}
```

Task 5: python p3_task5.py

```
PS C:\Users\Radhika\Desktop\ASU\CSE 515\PROJECT\asu-cse515-phase3\Code> python p3_task5.py
Enter space-separated values of 'query image', 'bits', 'input folder', 'feature model', 't':
100/image-cc-1-8.png 10 500 elbp 3
```

Task 6: python p3_task6.py

Task 7, Task 8: python p3_task8.py

```
PS D:\MS CS ASU\SEM 1\CSE 515 MWDB\Phase 3\Repo\asu-cse515-phase3\Code> python p3_task8.py
Enter the query image:
all/image-cc-1-1.png
Enter number of nearest neighbors required:
20
--------------------------------------------------------------------------------
Enter the details of vector set you want:
Enter the image folder for creating vector space:
100
Enter the feature-model required (cm/elbp/hog):
elbp
Enter the dimensionality reduction model (pca/svd/lda/kmeans/none)
pca
Enter the latent features needed (k):
10
Creating vector space...Please wait.
100_elbp.csv found!
Shape of data matrix:  (100, 576)
No. of labels:  100
Applying dimensionality reduction: pca
Latent Semantics shape:  (576, 10)
Transformed data matrix shape:  (100, 10)
```

5. Related Work

In [4], authors Alexandr Andoni and Piotr Indyk give an overview of efficient algorithms for the approximate exact near neighbour search problem. They survey a family of nearest neighbour algorithms that are based on the concept of Locality-Sensitive Hashing. They also describe a recently discovered hashing-based algorithm, for the case where the objects are points in the d-dimensional Euclidean space. In the end, they prove that the performance of the newly discovered algorithm is near-optimal in the class of the locality-sensitive hashing algorithms.

In [5], authors Stephen Blott and Roger Weber introduce the algorithm of Vector-Approximation file (VA-file) for similarity search in high dimensional vector spaces. They discuss how this method overcomes the dimensionality curse by adopting a filter-based approach of signature files instead of the traditional method of data partitioning. They go onto evaluating the performance of VA files on the basis of real and semi-synthetic vector data characterizing colour features of a moderate-sized image database. For large databases, the VA-File outperforms a well-tuned scan by a factor of up to four. They have also shown that performance does not degrade, and how it improves slightly with dimensionality. It is possible to integrate search in VA-File structures over multiple vector spaces, and also over signature files for non-metric spaces. Such searches are necessary to support conjunctive queries, which are frequent in multimedia databases. Further, they conclude that both the VA-File and signature methods are relatively straight-forward to parallelize and distribute.

In [6], the authors studied the impact of dimensionality on the nearest-neighbour similarity-search in high dimensional vector spaces from a theoretical and practical point of view. Under the assumption of uniformity and independence, they established lower bounds on the average performance of Near Neighbour-Search for space, data-partitioning, and clustering structures. They prove that these methods are out-performed by a simple sequential scan at moderate dimensionality (i.e. $d = 10$). Further, they have shown that any partitioning scheme and clustering technique must degenerate to a sequential scan through all their blocks if the number of dimensions is sufficiently large. Experiments with synthetic and real data were conducted to show that the performance of R*-trees and X-trees are outperformed by a sequential scan if dimensionality becomes large. They go on to conclude that at moderate and high dimensionality ($d > 6$), the VA-File method can out-perform any other method and that performance for this method even improves as dimensionality increases.

In [7], author Poonam proposed an overview of an efficient indexing method for high-dimensional databases using a filtering approach known as vector approximation approach which supports the nearest neighbour search efficiently. A cluster distance bound based on separating hyper planes, that complements the index in electively retrieving clusters that contain data entries closest to the query. In high dimensional, data-sets exhibit significant correlations and non-uniform distributions. Hence, indexing with the VA-File, by performing uniform, scalar quantization, is suboptimal. She then went on to propose an indexing method, based upon principles of vector quantization instead, where the data set is partitioned into Voronoi clusters and the clusters are accessed in order of the query-cluster distances. The cluster-distance bounds can be tightened by optimizing the clustering algorithm so as to optimize the cluster distance bounds.

6. Conclusions

The final phase of this project was an extended application of the prior phases and tasks, where we had worked on feature extraction, dimensionality reduction, vector spaces, and graphs. This phase introduced the concepts of classification of data into various classes, performing efficient nearest neighbour searches from database using index structures, and finally deploy relevance feedback to improve the search results.

We worked on SVM, DT, and PPR classifiers for classification tasks, where we trained these models on a given training dataset for the respective class labels and then performed predictions on a test dataset to generate the predicted labels. After comparison of several results, we cannot say that any particular classifier performs significantly better than the others. But the DT classifier predicts very well if tested on the same data it was trained on. Some other common trends we identified were that all the classifiers take longer time to train when we provide larger training sets and larger feature spaces to them, but they also improve in terms of their accuracy, false positive and miss rates. Another observation pertains to the fact that these classifiers perform significantly well for type labels but return poor results for subject and sample labels.

For nearest neighbour searches, the LSH and VA-File index structures were created, which work on the concept of dividing the database objects in several buckets, and only the buckets which are in the query range are searched to find the nearest neighbours. These index structures drastically reduce time of search as compared to the sequential scan, as we only have to perform search for objects in the buckets that lie in the query range.

For relevance feedback, we have implemented an interface which enables the user to provide a query and its parameters. We then perform nearest neighbour search using LSH or VA Files, and provide feedback labels as relevant or irrelevant to the returned set of results. With this extra information, the SVM or DT classifier is trained again on these nearest neighbours and they are re-ranked as per the feedback given by the user. From our observations, relevance feedback is generally working well, as the neighbours similar to the ‘relevant’ neighbours get assigned a higher rank, and the neighbours similar to the ‘irrelevant’ neighbours go to the lower ranks of the search results.

In conclusion, this phase concluded the abstract idea of real-world image searching and retrieval application through a proper channel of classifiers, index structures, and relevance feedback. As this phase marks the end of the project, we could combine all the phases of the project and get an idea of how the process of data retrieval actually works in a practical setting.

7. Bibliography

[1] https://en.wikipedia.org/wiki/Support-vector_machine#Definition

[2] https://www.logic2020.com/insight/tactical/decision-tree-classifier-overview

[3] https://changuk.github.io/algorithm/2013/08/28/pagerank.html

[4] Andoni, Alexandr & Indyk, Piotr. (2008). Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. Commun. ACM. 51. 117-122. 10.1145/1327452.1327494.

[5] Blott, Stephen & Weber, Roger. (1998). A Simple Vector-Approximation File for Similarity Search in High-Dimensional Vector Spaces.

[6] Weber, R., Schek, H., & Blott, S. (1998). A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. *VLDB.*

[7] Sahu, Mridu and Poonam Yerpude. “Vector Approximation File: Cluster Bounding in High-Dimension Data Set.” (2011).

[8] https://en.wikipedia.org/wiki/Relevance_feedback

[9] Elmore, Kimberly L., and Michael B. Richman. "Euclidean distance as a similarity metric for principal component analysis." Monthly weather review 129.3 (2001): 540-549.

[10] K. Seluk Candan and Maria Luisa Sapino. Data Management for Multimedia Retrieval. Cambridge University Press, New York, NY, USA, 2010

8. Appendix

Group Member	Contributions
Aaryan Gupta	Code – Task 1, 2, 3, 7, 8. SVM, PPR classifiers.
Fenil Madlani	Code – Task 5. Testing and error fixing in code.
Krisha Vijay Gala	Code – Task 6. Decision Tree classifier, Output collection.
Pranav Katariya	Code – Task 5. Testing and error fixing in code.
Radhika Ganapathy	Code – Task 4 Report-making. Output collection.
Shivam Malviya	Code – Task 1, 4, 7, 8. PPR classifier. Error Handling.