CSE 515 Group Project Phase 3

Preston Mott

Brandon Bayles

Ian Bolton

Keenan Rahman

Kunhao Zhang

Tanya Aggarwal

Abstract

This phase of the project involved the team continuing to utilize the Olivetti Faces dataset of images to extract Color Moments (CM), Extended Binary Local Patterns (ELBP), and Histograms of Oriented Gradients (HOG) feature sets.[1] These are represented as vector models, and then further utilized to perform several tasks related to classification using Support Vector Machine (SVM), Decision Tree (DT), and Personalized Page Rank (PPR) [1]. Data was also utilized for clustering purpose and was performed using Vector Approximation Files (VA-Files), and Locality Sensitive Hashing (LSH).[1] For dimensionality reduction we used Principal Component Analysis (PCA). With these goals in mind, the various tasks were allocated to the team and combined to produce our software.

Keywords

Feature Extraction, Dimensionality Reduction, Vector Representation, Latent Semantics, Principal Component Analysis, PCA, Singular Value Decomposition, Personalized PageRank, Decision Tree, Support Vector Machine, SVM, Locality Sensitive Hashing, VA-Files, Clustering, Classification

Introduction

A huge amount of data is being produced every day, these data can be documents, images, videos, audio and lot other things. One important feature of these data items is that they have high dimensions. Dealing with textual data is easier when compared with images.

When working with high dimensional data in large quantity there needs to be an easy and accessible way to query out specific data from the large database. Performing sequential scan on large data with high dimension can be time consuming and my lead to exponential time when we try to query a given data in the database. For instance, we were given images of size 64*64[1] and given a specific image trying to find similar image can be computationally very expensive.

Using multimedia big data approach, we have used various ways to classify different types of images and find the similar images to the query image. This phase of project deals with performing classification on different types of data in order to correctly classify given new query image. Create clusters and use them to find nearest images, so when a new query image comes, we can correctly identify its correct cluster and perform analysis to find k nearest images. After this we used the approach of user feedback where we consider user feedback about relevant and irrelevant images and use them to correctly classify nearest images. In this phase we performed classification by implementing Support Vector Machine (SVM), Decision Tree (DT), Personalized PageRank (PPR). For clustering and nearest neighbor search we used Vector Approximation Files (VA-Files), and Locality Sensitive Hashing (LSH). For getting user relevance feedback we used the classification and nearest neighbor search algorithm in parallel to get nearest images considering user feedback.

Goal Description

Task 1

Implement a program which,

– Given a folder of images, one of the three feature models, and a user specified value of k, computes k latent semantics (if not already computed and stored), and

– Given a second folder of images, associates X labels to the images in the second folder using the classifier selected by the user:

∗ an SVM classifer,

∗ a decision-tree classifier, or

∗ a PPR based clasifier,

in this latent space. Also compute and print false positive and miss rates. [1]

Task 2

Implement a program which,

– Given a folder of images, one of the three feature models, and a user specified value of k, computes k latent semantics (if not already computed and stored), and

– Given a second folder of images, associates Y labels to the images in the second folder using the classifier selected by the user:

∗ an SVM classifer,

∗ a decision-tree classifier, or

∗ a PPR based clasifier,

in this latent space. Also compute and print false positive and miss rates. [1]

Task 3

Implement a program which,

– Given a folder of images, one of the three feature models, and a user specified value of k, computes k latent semantics (if not already computed and stored), and

– Given a second folder of images, associates Z labels to the images in the second folder using the classifier selected by the user:

∗ an SVM classifer,

∗ a decision-tree classifier, or

∗ a PPR based clasifier,

in this latent space. Also compute and print false positive and miss rates. [1]

Task 4

– Implement a Locality Sensitive Hashing (LSH) tool, which takes as input (a) the number of layers, L, (b) the number of hashes per layer, κ, and (c) a set of vectors (generated by other tasks) as input and creates an in-memory index structure containing the given set of vectors. [1]

– 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), and

∗ given image and t, the tool outputs the t most similar images; it also outputs · the numbers of buckets searched as well as the unique and overall number of images considered · false positive and miss rates.

Task 5

– Implement a VA-file index tool and associated nearest neighbor search operations. Given (a) a parameter b denoting the number of bits per dimensions used for compressing the vector data and (b) a set of vectors (generated by other tasks) as input, the program and creates an in-memory index structure containing the indexes of the given set of vectors. The program also outputs the size of the index structure in bytes. [1]

– 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 a VA-file data structure (the program also outputs the size of the index structure in bytes), and
>
> ∗ given image and t, the tool outputs the t most similar images; it also outputs · the numbers of buckets searched as well as the unique and overall number of images considered · false positive and miss rates.

Task 6

Decision-tree-based relevance feedback: Implement a decision tree based relevance feedback system to improve nearest neighbor match, which 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. [1]

Task 7

SVM-clasifier-based relevance feedback: Implement an SVM based relevance feedback system to improve nearest neighbor matches, which 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. [1]

Task 8

Query and feedback interface: Implement a query interface, which allows the user to provide a query, 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 to user to provide positive and/or negative feedback for the ranked results returned by the system. User feedback is than taken into account (either by revising the query or by re-ordering the results as appropriate) and a new set of ranked results are returned. [1]

Description of the Proposed Solution/Implementation

Helper/Interface Files

Helper

This file provides a variety of functions to tasks 1-3 that are reused frequently and thus were more convenient to add to a single file to be referenced by the three task files. These functions include one to extract the selected features from a set of images, one to call the train function of the appropriate classifier, one to call the test function of the appropriate classifier, one to get the object feature matrix, one to apply the appropriate dimensionality reduction to the data, one to retrieve the features of the images from the database, one to get the image labels, one to print the classifier test results including miss and false positive rates, one to print the actual versus the predicted labels, one to determine if a value is a float, and finally one to take the euclidean distance between two values.

Database

For the database we use the latest version of MongoDb. Database implementation is done under the database.py file. We have a new database named "mwdb_database_phase_3". This database has multiple tables that are used to contain data for different usage.

- The 1st table "latent_semantics" contains all the image matrices that are either in reduced form or having all the features. It has 2 columns "_id" that has the image file name as id, and "latent_semantic" which has the image feature matrix stored.

Feature Descriptors

Color Moments

Color moments are measures that can be used to differentiate images based on their features of color.[2] In order to calculate the color moments, three ways were used (mean, standard deviation, and skewness). The image matrix was divided into an 8x8 box to perform calculations. All these calculations are combined to calculate the color moments of a given image.

Mean: The first color moment can be interpreted as being the average color in the given image.[2] This is calculated by formula.[2]

$$E_i = \sum_{j=1}^{N} \frac{1}{N} p_{ij}$$

Here N is the total number of points in the given box and p_{ij} is the value of the ith row and jth column of the box.

Standard Deviation: The second color moment is obtained by taking the square root of the variance of the color distribution. This is calculated by formula.[2]

$$\sigma_i = \sqrt{\left(\frac{1}{N} \sum_{j=1}^{N} (p_{ij} - E_i)^2\right)}$$

Here E_i is the mean value of the image.

Skewness: The third color moment measures how asymmetric the color distribution is and thus it gives information about the shape of the color distribution. This is calculated by formula.[2]

Extended Local Binary Pattern

$$s_i = \sqrt[3]{\left(\frac{1}{N} \sum_{j=1}^{N} (p_{ij} - E_i)^3\right)}$$

$$\mathrm{LBP} = \sum_{i=0}^{P-1} s(n_i - G_c)\Big) 2^i$$

$$s(x) = \begin{cases} 1, & if\ x > 0 \\ 0, otherwise \end{cases}$$

The extended LBP is a joint distribution of grayscale and rotational invariant LBP with the rotational invariant Variance measure.[4] In order to calculate the ELBP we set the local neighborhood to 8.00 and the radius is set to 1.00 on a given image vector. The formula for ELBP is [3]:

where P is the number of neighborhood pixels, n_i represents the ith neighboring pixel, and c represents the center pixel. The histogram features of size 2P are extracted from the obtained LBP code. [3]

Histogram of Oriented Gradients

The histogram of oriented gradients is a feature descriptor used in image processing. The technique counts occurrences of gradient orientation in localized portions of an image. In order to calculate the HOG, we set the number of orientation bins to 9, cell size to 8 and block size to 2. The block normalization is set to the L2-norm clipping threshold set to 0.2.

$$\text{L2-norm: } f = \frac{v}{\sqrt{\|v\|_2^2 + e^2}}$$

L2-hys: L2-norm followed by clipping (limiting the maximum values of v to 0.2) and renormalizing.

For each of the models a separate function is created that takes image id and image matrix as input and provides the result for the corresponding model selected.

Dimensionality Reduction

Principal Component Analysis

The PCA file takes a k value and a NumPy array dataset of size n x p where n represents the number of data vectors, p represents the number of elements per data vector, and k represents the number of latent semantics to reduce the dimensions to. The program starts by normalizing the data by taking the mean of every column and then subtracting these means from the original data. The covariance matrix of the normalized data is then taken and used to extract the eigenvalues and eigenvectors. The covariance matrix is calculated using the following formula:

$$cov = \frac{(D^{-1} \cdot D)}{n-1}$$

Where *cov* is the covariance matrix, D represents the normalized data matrix and n represents the number of data vectors. The eigenvalues and eigenvectors are determined from this result by utilizing the scipy.linalg.eigh method. The eigenvalues are sorted in descending order and then the k highest associated eigenvectors are selected. The final dimensionally reduced data is found by projecting the selected eigenvectors onto the normalized data with a dot product calculation. The method then returns the dimensionally reduced data, the final selected eigenvectors, the sorted eigenvalues, and the sorted eigenvectors in a list. This code is referenced whenever a task needs to calculate a PCA decomposition.

Image Classifiers

Support Vector Machine

The SVM takes two arguments in order to train the classification model. It takes an array of images and list of labels for each of the image as input. For multiclass SVM we implemented Crammer-Singer formulation. [5] At every iteration, dual decomposition

$$\hat{y} = \operatorname*{argmax}_{m \in [k]} \boldsymbol{w}_m^{\mathrm{T}} \boldsymbol{x}.$$

,

methods update a small subset of dual variables by solving a restricted optimization problem. [5]

In order to classify an image of a given type we used the following equation. Each vector $w_m \in Rd$ can be thought as a prototype representing the mth class and the inner product $w^T m_x$ as the score of the mth class with respect to x. [5]

This equation [5] means that, for each training instance, we suffer no loss if the score of the correct class is larger than the score of the "closest" class by at least 1. The accuracy of the model for the images in 3 classes is close to 70% and it gradually decreases if we keep on adding more and more classes. The major reason as per our analysis can be the smaller number of training data provided.

$$\underset{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_k}{\text{minimize}} \ \frac{1}{2}\sum_{m=1}^{k}\|\boldsymbol{w}_m\|^2 + C\sum_{i=1}^{n}\left[1+\max_{m\neq y_i}\boldsymbol{w}_m^{\mathrm{T}}\boldsymbol{x}_i-\boldsymbol{w}_{y_i}^{\mathrm{T}}\boldsymbol{x}_i\right]$$

Assumptions

- It also assumes that the training set has a set of labels with indices to match the provided latent semantics of the associated image provided alongside it as well.
- The value of the regularization parameter is set to be used as 0.001. Its a very small value of C, this causes optimizer to look for a larger margin separating hyperplane, even if that hyperplane misclassifies more points. The reason for doing is we have images of lots of different label and data size is small. In order to cater it and get better result we had to use smaller value of C.
- The max iteration value for SVM is set to 1000 as sometimes with more images of different types, SVM might not be able to converge at better rate and considering the amount of time each iteration takes we set the value to 1000.
- For gradient descent we use learning rate of 0.001, we used this low value to make our SVM classifier as much accurate as possible.

Decision Tree

To begin construction of our decision tree, a research phase was commenced. As discussion of this topic had not yet been broached, this step was critical in order to see our project completed by the deadline. It was here that we learned the essentials of decision tree formation and classification using such a data structure. Decision trees are made up of nodes which contain a query (which posits a binary question based upon the training data), a true branch, and a false branch. When making a tree, we pass in a training dataset, and build the tree recursively. Using this data, nodes are created which ask queries that separate the data; such questions may be numerical or categorical in nature. The data is subdivided at each level in this way, eventually reaching a leaf node where a classification lies. Leaf nodes contain the classification of the given data, which is outputted. A given leaf node may have still have several possibilities as to what the data that falls there may be, to prevent overfitting. In these instances, it is an equal likelihood amongst all possible labels at that leaf node, thus the first of the labels is chosen as the corresponding classification. To determine the best possible separation of data, we calculate what is known as the information gain. To do so, we must also calculate the Gini index, or Gini impurity. This is a calculation which considers the probabilities of each label in the resulting split. Information gain uses this Gini impurity to quantify how much the split truly divides the data by measuring the proportion of rows on either side of the split. The best split is found iteratively by using the values of the current feature being used to separate the training data. The separation with the largest split is chosen as the query to divide the data with.

Assumptions

- Python 3 and all listed Python libraries are installed.
- Training data is passed in with the parameters X_train, Y_train, where X_train is the data as a list of lists and Y_train is a vector with the corresponding labels for each row.
- Testing data is passed in similar to how training data is, but there is no Y_train variable.
- It is assumed only one classification result is desired, and thus the first classification present at a leaf node is chosen to be returned by the program.
- The output when classifying a set of test data is a list of labels, corresponding to the order of data points.

Personalized Page Rank

The PPR classifier starts by being initialized with the latent semantic values and labels of the training set. Then the user can call the predict function with the latent semantic values of the test set to begin the classification process. The first step is to generate a similarity matrix of the image's latent semantic values. This is done by creating a 2D dataframe of zeroes with a cell for each pair of images in the combined training and test sets.

Next, the dataframe is partitioned into smaller dataframes for use in multiprocessing similarity calculations. Each partition is passed into a subprocess that calculates the similarity of the latent semantic values of each pair of images in the partition using Euclidean distance. This similarity value is stored into the appropriate cell. Once each of the partitions' subprocesses are complete, the partitions are put back together to form the entire similarity matrix.

Next, this matrix is used to create a similarity graph where each node has 5 edges to the top 5 most similar images. This similarity graph is then converted into a transition matrix where each cell corresponds to a pair of nodes and if there is an edge between the nodes then the value of the cell is 1, otherwise it is 0. Next, for each image in the test set, a personalized page rank transformation is applied to the transition matrix wherein the seed set is composed of the current test set image. This transformation is based on the basic formula below:

$$p = \left(I - beta(T)\right)^{-1}(1 - beta)c$$

Where p is the vector of scores, I is the identity matrix, beta is the probability of the random surfer to not jump, T is the transition matrix, and c is the vector associated with the likelihood of the random surfer to jump to a particular node when the random surfer makes a jump. This c vector is created with a much higher chance to jump to the node given in the seed set than those out of the seed set, thus differentiating this algorithm from regular PageRank.

Once a PPR score has been generated, the top 5 highest scoring images from the training set are analyzed and have their labels retrieved. Of this set, the most commonly occurring label is selected as the most likely label for the test set image given as the seed set to PPR. This process is repeated for every image in the test set until a most likely label is generated for each one. This set of labels is returned by the classifier to finish the algorithm.

Assumptions

- This program assumes that Python is installed as well as the numpy, pandas, math, and multiprocessing libraries. Additionally, it assumes that the latent semantics for each image in both the training and test sets have already been generated and are provided to represent the images.
- It also assumes that the training set has a set of labels with indices to match the provided latent semantics of the associated image provided alongside it as well.
- The algorithm assumes that the provided dataset of images for both training and testing sets is not large, as even with multiprocessing implemented the similarity matrix calculation is particularly expensive and takes significant processing time with datasets greater than 500 images.
- For the PPR scoring, the value of 0.84 is given for beta since it is the industry standard and gives a higher probability of the surfer visiting a node's neighbors over jumping to a different node.
- The value of 0.99 is given to the node associated with the current test node provided in the seed set for the c vector and all other indexes are given a value of 0.01 / length of the similarity graph to give an even but significantly smaller chance of jumping to the non-seed set nodes.

Image Clustering

Vector Approximation Files

We created the Vector Approximation File (VA-File) in a similar way to how it was described in the paper by Stephen Blott and Roger Weber [7]. The purpose of VA-Files is to create a structure in memory that allows you to receive a good approximation of similar objects while limiting the operations that are needed to be performed. The first step is to calculate the bj value for each dimension as described in the reference paper[7]. This bj value is used to calculate how many partition ranges are created for each dimension; the specific number of partition ranges created is equal to 2^{bj} for the jth dimension. The second step of creating the structure in memory is to create partitions in the dimensional space that the objects exist in. To calculate these partitions we take the largest and smallest value at each dimension in the entire dataset. These values become the lower and upper bounds for the partition on the respective dimension. We then create 2^{bj} evenly distributed partitions for each of the j-dimensions. We can now begin to put the data inside of the VA-File structure.

To do this we iterate through each datapoint and find its appropriate partition for each of its dimensions values. Each partition set on each dimension can be represented by a bit string. The number of bits are each to the bj value for that dimension. The first partition is represented by 0 in binary and the last partition is represented by 2^{bj} in binary. Each time a datapoint is added to the VA-File the partition that it was added in is stored in binary. Once all of the datapoints have been placed inside of the VA-File the concatenation of all of the partition bit values can be used to represent the VA-File filled with the input datapoints. To generate the VA-File we take Now that all of the datapoints are stored inside of the VA-File

Assumptions

- When using the vector file as input to generate the VA-File the query image must be an object inside of the vector file and directory of images.
- The smallest number of partitions that can be generated is 2.

Locality Sensitive Hashing

Locality sensitive hashing (LSH) is a widely popular technique used in approximate nearest neighbor search. The nearest neighbor problem is defined as follows: given a collection of n points, build a data structure which, given any query point, reports the data point that is closest to the query.

We obtain our result by carefully designing a family of locality-sensitive hash functions for each layer.[6] There, a point p was mapped into 1 using a random projection. Then, the line 1 was partitioned into intervals of length w, where w is a parameter. The hash function for p returned the index of the interval containing the projection of p. [6]

To implement the algorithm, we take following 3 parameters as argument. First, we take number of layers, and number of hashed, and along with these 2 values we take vectors as input. For each layer L we generate H hash functions, each having B number of buckets. Now in order to create search index structure we use the given set of vectors and generate set of buckets for the given set of inputs. Now we take an image folder as an input and using all these images and the LSH model generated previously we populated the whole data structure. Once the data structure is populated, we use a query image and using each layer and each hash we try to get the bucket number under which this image falls.

For a given layer if union all the unique images that each hash function gives for a specific layer. Once we get all unique images for specific layer under all the hash functions, we iterate it to another layer and then intersect the results of all the images from previous layer to the results of current layer. Like this we iterate on all layers and find k-nearest images.

In case, if we don’t get the k images we expand our search by considering neighboring buckets until we get the k images that we desired.

Assumptions

- We try to use the specified vector file to initialize the search data structure. In case the vectors are not provided we randomly initialize the vector space to initialize LSH.

- The total number of buckets are minimum of 2 power number of hashes or dataset size.
- We stop our search when we get more than T images we are looking for
- In case if we don't get T images, we keep on expanding considering neighboring images until the expansion rate has considered about 40 buckets. The reason to set this value is that we don't want LSH to keep on running for long time as we have limited time for demo.
- For relevance feedback LSH runs until we get 30 nearest images or expansion rate reaches 20. Expansion rate means number of neighboring buckets considered.

Task Details

Task 1, 2, & 3

Tasks 1, 2, and 3 are all implemented in a similar fashion. We utilized all the feature models (color moments, histogram of oriented gradients, and extended local binary patterns) and dimensionality reduction algorithms (PCA, ALL). For applying classification, we used (SVM, DT, PPR).

We first apply a feature extraction algorithm on a while iterating through all the images and then applying linear transformations to the given image. As a result, an image-feature matrix is generated with images as rows and feature values as columns. Then if the reduction algorithm is given ALL, we don't apply any reduction on the images, in case of PCA, after getting the object-feature matrix, this matrix is passed to dimensionality reduction algorithms; the description of each dimensionality reduction algorithm is given in the above headings. These algorithms give us k-latent semantics for the given objects.

After setting up our image date we create an array and call it X_train which contains all the training images. We create another list of name Y_train that contains the label of each of the image. For task-1 the labels are the type names, for task-2 the labels are the subject id, and for task-3 the labels are the sample id. Once X_train and Y_train are initialized we send these parameters to the classifier defined by user. The definition and detail of each classifier is given above.

Once we get the trained model, we use another folder defined by user to extract its feature in a similar way as we had done for training data previously. We create an array of images and name it X_test. This array is passed to the predict function of classifier that computes the prediction of each test image and returns the predicted labels for each image. We then pass the predicted result and actual result to the function that computes the counts of misses and false positive for each of the label.

Task 4

First given set of vectors, value of layer and value of hashes, we pass these values to LSH in order to generate the data structure. The vector values can be small sometime for which we appended the random vector values for each vector in order to get the exact vector length we want.

After getting the data structure, we iterate and fetch all the images in the given folder. We utilized all the feature models (color moments, histogram of oriented gradients, and extended local binary patterns) in order to calculate the features of the images. These images are then passed to the LSH to populate the images. Each image is assigned to a bucket of a layer of specific hash.

Once the data structure is populated, we get the query image and in similar fashion we first calculate the feature of this image. After this, the image is passed to the LSH model in order to calculate the T nearest images. Once we get the T nearest images. We run a KNN nearest neighbor search to get the true T nearest images of the query image. After getting the results we use the nearest images from KNN as actual T images and images from LSH as predicted nearest images. Then we compare both the list find the misses and false positive for the query image.

Task 5

In this task we utilize the VA-File structure that is described in the Vector Approximation File section. Once the appropriate user input is provided and the VA-File has been constructed we can perform an efficient t-nearest-neighbor search. This is done by first inputting a query datapoint and finding the partition that it belongs to. It will then check all of the datapoints inside of its bucket and output the t nearest objects using Euclidean distance. If its bucket does not have t objects inside of it then it will expand its search to include the next nearest partition. It will continue to do this until it has found t objects. We then calculate the true k-nearest-neighbors by ignoring the buckets and simply performing Euclidean distance on all of the objects, sort the values, and output the top k results. We can then compare the results from the t-nearest-neighbors and the k-nearest-neighbors search to calculate the false positive and miss rate. The false positive rate is calculated by the number of nodes found in the t-nearest search but not in the k-nearest search divided by the total number considered. The miss rate is calculated by the nodes in the k-nearest search that are not considered by the t-nearest search divided by the total number considered.

Task 6

Task 6 utilized the decision tree classifier constructed during the creation of Tasks 1, 2, and 3. Here, we execute a nearest neighbor search either through Task 4 or Task 5. From this initial search, the user is then prompted to label each of these results as relevant or not (represented in our program as a 1 or 0, respectively). Using these parameters, it is the aim of this Task to reorder the results of the nearest neighbor query. This reordering is done through the use of the decision tree. Trained on the nearest neighbors and the given relevancy vector, it is then tested on the entire image folder passed into the query task. The neighbors are retrieved and displayed to the user in a new ordering depending on their relevance score.

Task 7

In order to implement SVM based classifier based on user feedback. We used an approach that included the use of LSH model we previously implement and SVM classifier.

First given a folder of images we first create a LSH index structure by using random vectors for generation of bucked and populating the index structure with all the images of the given folder. After the hashing algorithm is constructed, we passed the query image to get the first 25 nearest images of the given query image and display all 25 images to the user.

User is then asked to provide feedback about which images seems to be relevant and which images doesn’t seem to be relevant to the query image. After getting user feedback we classify the marked images under 2 labels (relevant and irrelevant). We then train the SVM model based on these images. Once the classifier gets trained, we pass all the uniquely identified images from the LSH to the SVM and analyze what images that were identified by the LSH are predicted as relevant by the SVM classifier. Once we get the relevant marked images from SVM we display this result to the output with the score values that were predicted by the LSH

Task 8

The purpose of task 8 is to create an interface for tasks 4, 5, 6, and 7. The interface is used by performing either task 4 or task 5 then feeding the output of one of those tasks to either task 6 or task 7. The output of task 4 and task 5 that is used is the output file that is described in the task description of task 4 and task 5. This file contains all of the distance and image names that were considered using the partition structure created. The top t results that are returned from task 4 or task 5 are then visualized to the user. The user can then input whether the images displayed are relevant to them or not. This input will be fed into task 6 or task 7 along with the output file from task 4 or task 5. Then the new top results using the relevancy scores and either task 6 or task 7 will be visually displayed to the user.

Interface Specifications

Task 1, 2, & 3

This task is located under task/task_1.py or task/task_2.py or task/task_3.py and take following arguments as input: an image path for the training images, an image path for the testing images, a feature model name, a dimensionality reduction option (either PCA or ALL to indicate no dimensionality reduction), an image classifier, and a k value. These values are used to extract the indicated features, perform dimensionality reduction if desired, and then pass these values into the training and testing functions for the selected classifier. The results of the classifier are then printed with the predicted and actual labels as well as the per class and overall correct, miss, and false positive rates.

Task 4

This task is located under task/task_4.py and take following arguments as input: the number of layers to use, the number of hashes per layer to use, the vector path, the training image path, the feature model to use, a t value to represent the t nearest neighbors to retrieve, and the query image path. These are all provided through the task 8 interface, and the results are returned to the interface as well. Through the interface, the user is given the option of utilizing a vector created from a previous phase or run or to generate a new vector file to use from a folder of images. The features of the query image and image folder provided are extracted and stored in data frames. An LSH system is then trained on the given path of images and the indexing system is set up using the given number of layers and hashes per layer. The top k closest images to the query image are retrieved using the LSH model and then sent back to the interface for display to the user.

Task 5

This task takes several parameters as input. It takes in a decision choice, image path, model name, query image path, t value, b value, and optional vector path from the user. The decision choice is either a value of 1 or 2 that lets the user decide whether they want to generate the VA-File with either a vector file, this can be either a feature vector file or a latent semantic vector file, or from a dataset of images. The image path that is passed in is a folder that contains the images whose features are going to be used to populate the VA-File and optionally generate the VA-File structure if that decision is chosen. The model name is used to generate the feature vector for the image dataset that was passed in. The query image path references a file that belongs to the image directory passed in and is used for the query image inside of the search function. The query image path that is specified has to be present inside of the vector file and image directory that is passed in by the user. The t value is going to be used inside of the t-nearest-neighbor search to determine how many values should be returned. The b value will be used to calculate the bj value for each of the dimensions present in the dataset. The vector path file will be used to generate the VA-Files structure if that decision is chosen. The output of this task will be on the terminal and stored into a file. The terminal will output the nearest t objects and the bucket and node count considered to find those nearest t objects. Then the nearest k objects will be output along with the false positive and miss rate. Then a file will be output which contains the rankings of each image considered, this will be used as input for both tasks 6 and 7.

Task 6

This task uses the interface of Task 8 to pass input in. To start this task, the user must execute the command “python task_8.py” into the command line. From here, the user must answer a series of prompts in order to obtain the needed parameters for Task 6 execution. The first of these parameters is the nearest neighbors to be reordered. This is done through either Task 4 or Task 5. Following this, we specify our input decision (through vectors or image directory), image path, feature model, query image path, t value, and task parameters. The task parameters are specific to Tasks 4 and 5 and may be viewed in their respective Task descriptions. Following execution of the chosen task, the nearest neighbors are visualized to the user in ascending order of similarity. The user is then prompted to classify each of these images either as relevant or not (1 or 0). Then, to ensure Task 6 is performed, the user must enter “task_6” when asked which of the specified tasks to use for query revision. The decision tree classifies the image folder and returns the top t images initially visualized and labelled by the user, except reordered depending on their determined relevancy.

Task 7

This task uses the interface of Task 8 to pass input in. To start this task, the user must execute the command “python task_8.py” into the command line. From here, the user must answer a series of prompts in order to obtain the needed parameters for Task 7 execution. The first of these parameters is the nearest neighbors to be reordered. This is done through either Task 4 or Task 5. Following this, we specify our input decision through vectors, image directory, feature model, query image path, t value, and task parameters. The task parameters are specific to Tasks 4 and 5 and may be viewed in their respective Task descriptions. Following execution of the chosen task, the nearest neighbors are visualized to the user in ascending order of similarity. The user is then prompted to classify each of these images either as relevant or not (1 or 0). Then, to ensure Task 7 is performed, the user must enter “task_7” when asked which of the specified tasks to use for query revision. The support vector machine classifies the image folder and returns the top t images initially visualized and labelled by the user, except reordered depending on their determined relevancy.

Task 8

For this task many of the same inputs are described inside of both task 4 and task 5 so we will repeat which inputs are passed in but not their functionality since those have already been previously described. Task 8 takes in as input a task decision, input decision, image path, model name, query image path, and t value. The task decision is to decide whether to use the partition structure from either task 4 or task 5. The rest of the listed inputs are already described inside of the task description for task 4 and task 5. Then the results of task 4 or task 5 will be output to the terminal while the top t results from the chosen task will be visualized for the user. The user will then have to input the relevancy score of each of the images that was returned. The user will then choose either use task 6 or task 7 for the relevancy structure. This and the top t results will then be passed into the chosen relevancy algorithm. The output of either task 6 or task 7 will then be displayed onto the terminal while the top t results from these tasks and visualized for the user.

System Requirements / Installation and Execution Instructions

Prerequisites

- Python 3.6
- MongoDB Server 5.0
- The following Python libraries:
    - os
    - sys
    - matplotlib
    - numpy
    - PIL
    - pathlib
    - glob
    - json
    - csv
    - pandas
    - collections
    - scipy
    - sklearn
    - math
    - multiprocessing

Install Necessary Python Libraries

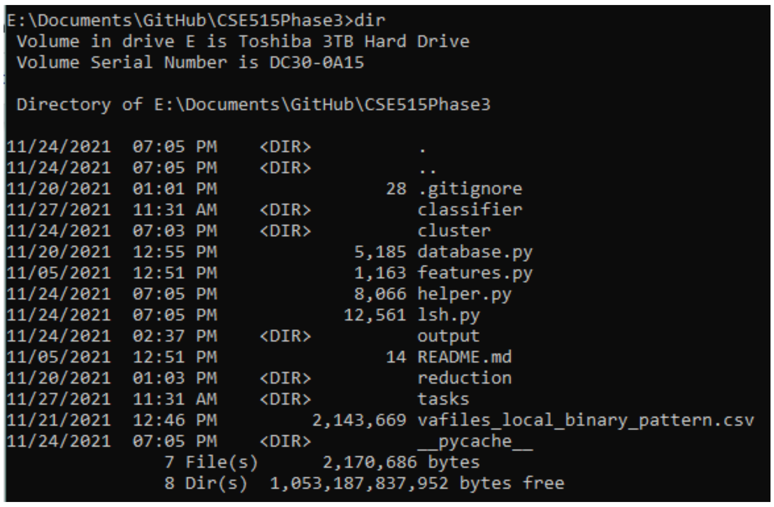
The libraries listed above, if not already installed, can be installed by opening a terminal and entering the command ‘pip install $library_name’ where $library_name represents the name of the library to install.

Start Database

Before running any of the tasks, the user should have launched a MongoDB server instance on their localhost network. This can be done by running the ‘mongod.exe’ file located in the \bin folder of the directory they installed MongoDB into. If this is the user’s first time installing and running a MongoDB server, they should also be sure to create the directory MongoDB will save its data into, so there should be a path ‘C:\data\db’. If this path does not exist, use the ‘md’ command to create it. Before running the tasks, the downloaded images for this phase should be moved into the database as well. To do this, open the ‘database.py’ file of our submission and add the line ‘insert_image_dataset($path)’ where $path represents the string of the path where the images have been downloaded to.

Navigate to Correct Directory

Be sure to navigate to the correct directory in the console before attempting to run any of the tasks. This should be the folder “CSE515Phase3”, under the path that you saved the project to. You can use the “cd” command to reach this folder. Once theore, use the “dir” command to ensure that your current directory looks something like this:


```
E:\Documents\GitHub\CSE515Phase3>dir
 Volume in drive E is Toshiba 3TB Hard Drive
 Volume Serial Number is DC30-0A15

 Directory of E:\Documents\GitHub\CSE515Phase3

11/24/2021  07:05 PM    <DIR>          .
11/24/2021  07:05 PM    <DIR>          ..
11/20/2021  01:01 PM                28 .gitignore
11/27/2021  11:31 AM    <DIR>          classifier
11/24/2021  07:03 PM    <DIR>          cluster
11/20/2021  12:55 PM             5,185 database.py
11/05/2021  12:51 PM             1,163 features.py
11/24/2021  07:05 PM             8,066 helper.py
11/24/2021  07:05 PM            12,561 lsh.py
11/24/2021  02:37 PM    <DIR>          output
11/05/2021  12:51 PM                14 README.md
11/20/2021  01:03 PM    <DIR>          reduction
11/27/2021  11:31 AM    <DIR>          tasks
11/21/2021  12:46 PM         2,143,669 vafiles_local_binary_pattern.csv
11/24/2021  07:05 PM    <DIR>          __pycache__
               7 File(s)      2,170,686 bytes
               8 Dir(s)  1,053,187,837,952 bytes free
```

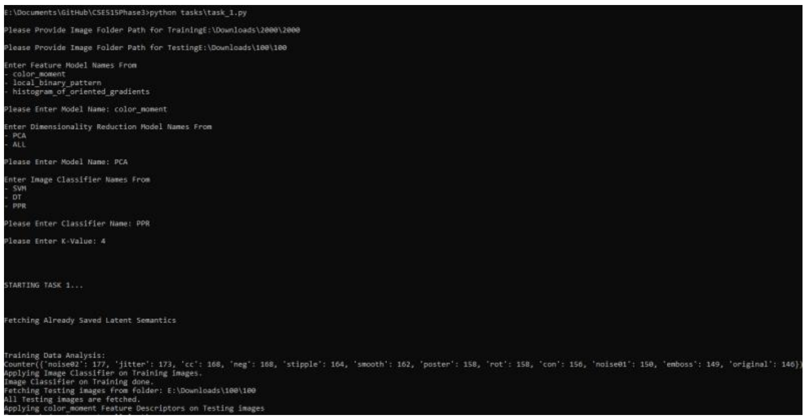
A few files may be different, but your folder should contain the following to run correctly:

- classifier
    - decision_tree.py
    - ppr.py
    - svm.py
- database.py
- features.py
- helper.py
- lsh.py
- output
    - task_4

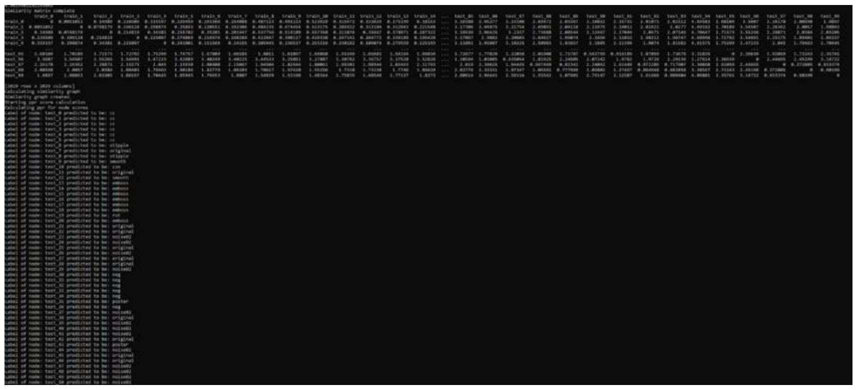
    - task_5
- reduction
    - pca.py
    - svd.py
- tasks
    - task_1.py
    - task_2.py
    - task_3.py
    - task_4.py
    - task_5.py
    - task_6.py
    - task_7.py
    - task_8.py

Task 1

For this task we take multiple parameters as input. First, we take the image folder path for training classifier and next parameter of image folder for testing the classifier. Next, we take feature descriptor and reduction algorithm as input. In order to have all dimensions included we use "ALL" in dimensionality reduction algorithm name. After this we specify classifier from SVM, DT, PPR and provide k-value (number of dimensions).

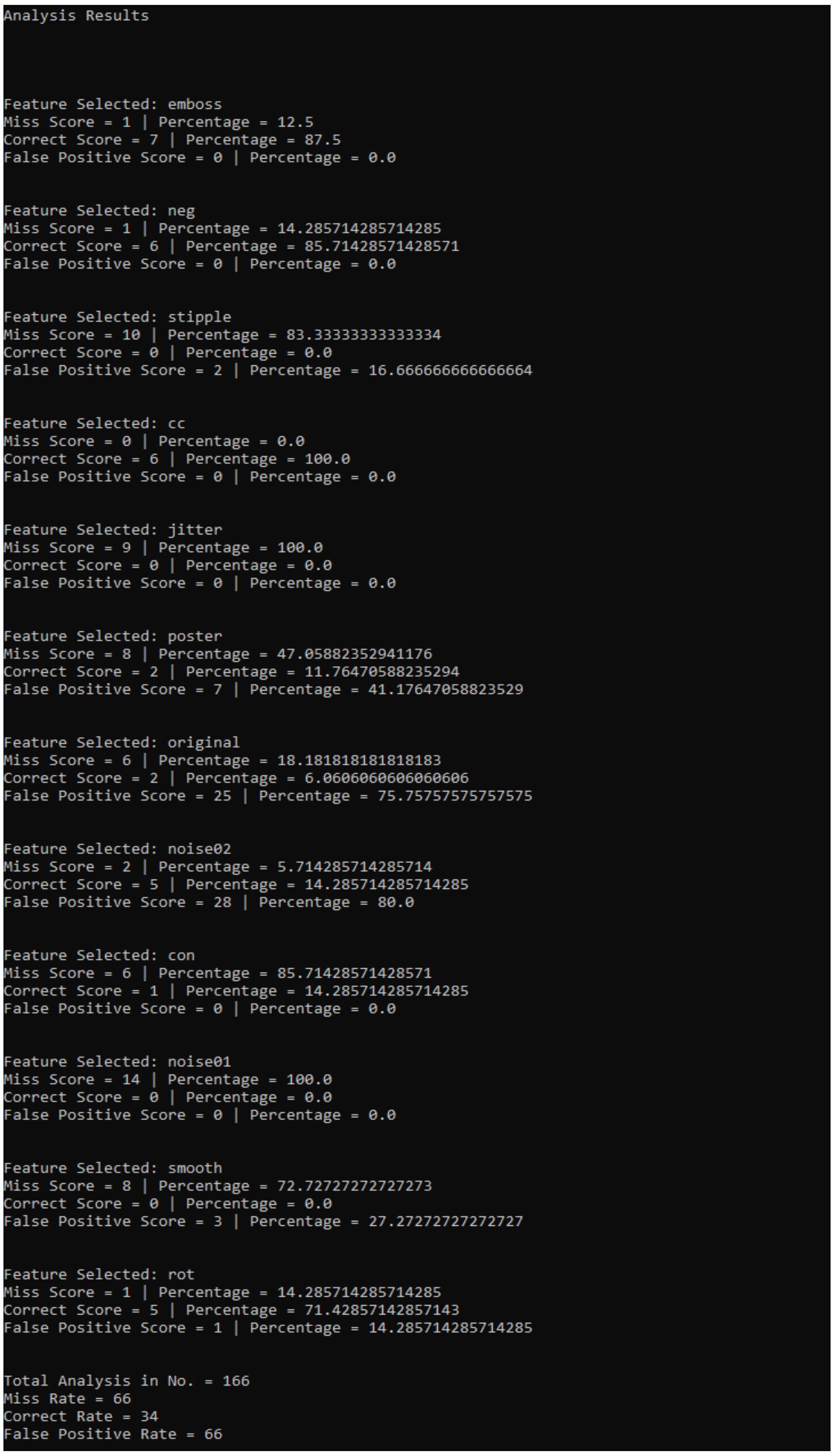


```
E:\Documents\GitHub\CSE515Phase3>python tasks\task_1.py

Please Provide Image Folder Path for TrainingE:\Downloads\2000\2000

Please Provide Image Folder Path for TestingE:\Downloads\100\100

Enter Feature Model Names From
- color_moment
- local_binary_pattern
- histogram_of_oriented_gradients

Please Enter Model Name: color_moment

Enter Dimensionality Reduction Model Names From
- PCA
- ALL

Please Enter Model Name: PCA

Enter Image Classifier Names From
- SVM
- DT
- PPR

Please Enter Classifier Name: PPR

Please Enter K-Value: 4



STARTING TASK 1...



Fetching Already Saved Latent Semantics



Training Data Analysis:
Counter({'noise02': 177, 'jitter': 173, 'cc': 168, 'neg': 168, 'stipple': 164, 'smooth': 162, 'poster': 158, 'rot': 158, 'con': 156, 'noise01': 150, 'emboss': 149, 'original': 146})
Applying Image Classifier on Training images.
Image Classifier on Training done.
Fetching Testing images from folder: E:\Downloads\100\100
All Testing images are fetched.
Applying color_moment Feature Descriptors on Testing images
```

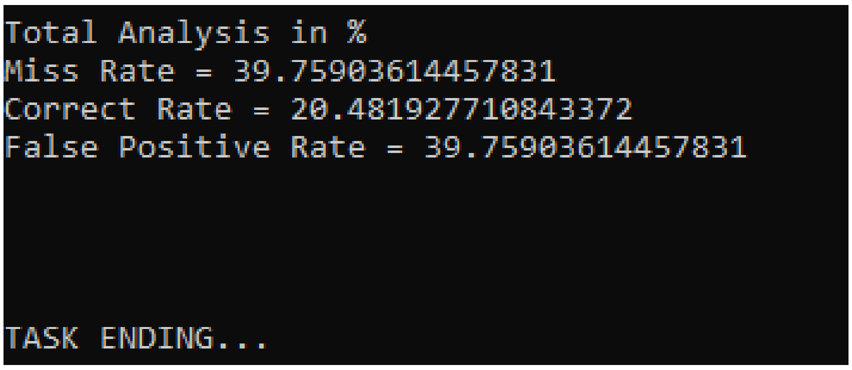


```
[illegible]
Similarity matrix complete
[illegible]

[illegible]
Calculating similarity graph
Similarity graph created
Starting ppr score calculation
Calculating ppr for node scores
Label of node: test_0 predicted to be: cc
Label of node: test_1 predicted to be: cc
Label of node: test_2 predicted to be: cc
Label of node: test_3 predicted to be: cc
Label of node: test_4 predicted to be: cc
Label of node: test_5 predicted to be: cc
Label of node: test_6 predicted to be: stipple
Label of node: test_7 predicted to be: original
Label of node: test_8 predicted to be: stipple
Label of node: test_9 predicted to be: smooth
Label of node: test_10 predicted to be: con
Label of node: test_11 predicted to be: original
Label of node: test_12 predicted to be: smooth
Label of node: test_13 predicted to be: emboss
Label of node: test_14 predicted to be: emboss
Label of node: test_15 predicted to be: emboss
Label of node: test_16 predicted to be: emboss
Label of node: test_17 predicted to be: emboss
Label of node: test_18 predicted to be: emboss
Label of node: test_19 predicted to be: rot
Label of node: test_20 predicted to be: emboss
Label of node: test_21 predicted to be: original
Label of node: test_22 predicted to be: original
Label of node: test_23 predicted to be: noise02
Label of node: test_24 predicted to be: noise02
Label of node: test_25 predicted to be: original
Label of node: test_26 predicted to be: noise02
Label of node: test_27 predicted to be: original
Label of node: test_28 predicted to be: original
Label of node: test_29 predicted to be: noise02
Label of node: test_30 predicted to be: neg
Label of node: test_31 predicted to be: neg
Label of node: test_32 predicted to be: neg
Label of node: test_33 predicted to be: neg
Label of node: test_34 predicted to be: neg
Label of node: test_35 predicted to be: poster
[illegible]
```

```
[illegible]
```



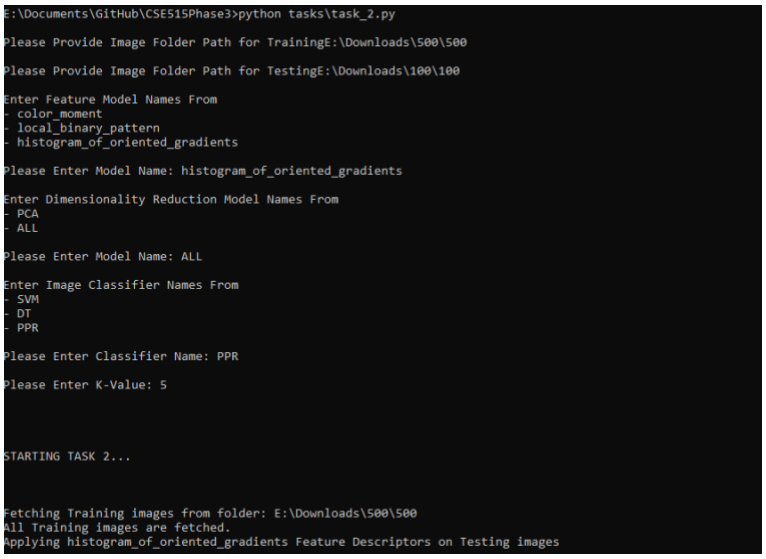
```
Analysis Results



Feature Selected: emboss
Miss Score = 1 | Percentage = 12.5
Correct Score = 7 | Percentage = 87.5
False Positive Score = 0 | Percentage = 0.0


Feature Selected: neg
Miss Score = 1 | Percentage = 14.285714285714285
Correct Score = 6 | Percentage = 85.71428571428571
False Positive Score = 0 | Percentage = 0.0


Feature Selected: stipple
Miss Score = 10 | Percentage = 83.33333333333334
Correct Score = 0 | Percentage = 0.0
False Positive Score = 2 | Percentage = 16.666666666666664


Feature Selected: cc
Miss Score = 0 | Percentage = 0.0
Correct Score = 6 | Percentage = 100.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: jitter
Miss Score = 9 | Percentage = 100.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: poster
Miss Score = 8 | Percentage = 47.05882352941176
Correct Score = 2 | Percentage = 11.76470588235294
False Positive Score = 7 | Percentage = 41.17647058823529


Feature Selected: original
Miss Score = 6 | Percentage = 18.181818181818183
Correct Score = 2 | Percentage = 6.0606060606060606
False Positive Score = 25 | Percentage = 75.75757575757575


Feature Selected: noise02
Miss Score = 2 | Percentage = 5.714285714285714
Correct Score = 5 | Percentage = 14.285714285714285
False Positive Score = 28 | Percentage = 80.0


Feature Selected: con
Miss Score = 6 | Percentage = 85.71428571428571
Correct Score = 1 | Percentage = 14.285714285714285
False Positive Score = 0 | Percentage = 0.0


Feature Selected: noise01
Miss Score = 14 | Percentage = 100.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: smooth
Miss Score = 8 | Percentage = 72.72727272727273
Correct Score = 0 | Percentage = 0.0
False Positive Score = 3 | Percentage = 27.27272727272727


Feature Selected: rot
Miss Score = 1 | Percentage = 14.285714285714285
Correct Score = 5 | Percentage = 71.42857142857143
False Positive Score = 1 | Percentage = 14.285714285714285


Total Analysis in No. = 166
Miss Rate = 66
Correct Rate = 34
False Positive Rate = 66
```



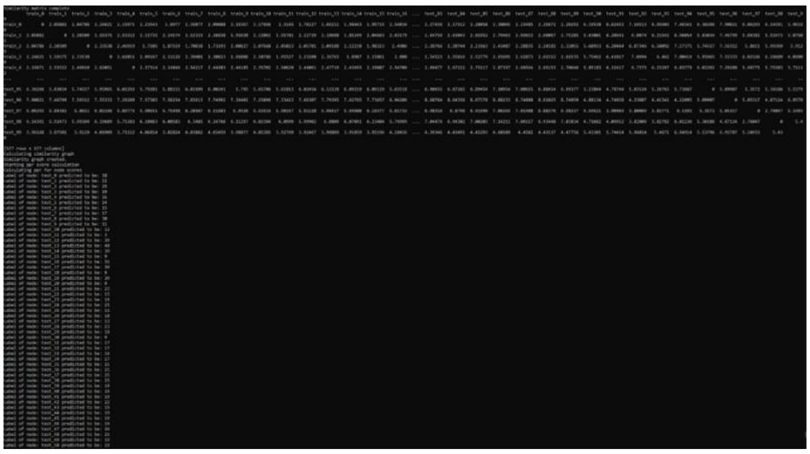
```
Total Analysis in %
Miss Rate = 39.75903614457831
Correct Rate = 20.481927710843372
False Positive Rate = 39.75903614457831




TASK ENDING...
```

Task 2

For this task we take multiple parameters as input. First, we take the image folder path for training classifier and next parameter of image folder for testing the classifier. Next, we take feature descriptor and reduction algorithm as input. In order to have all dimensions included we use “ALL” in dimensionality reduction algorithm name. After this we specify classifier from SVM, DT, PPR and provide k-value (number of dimensions).

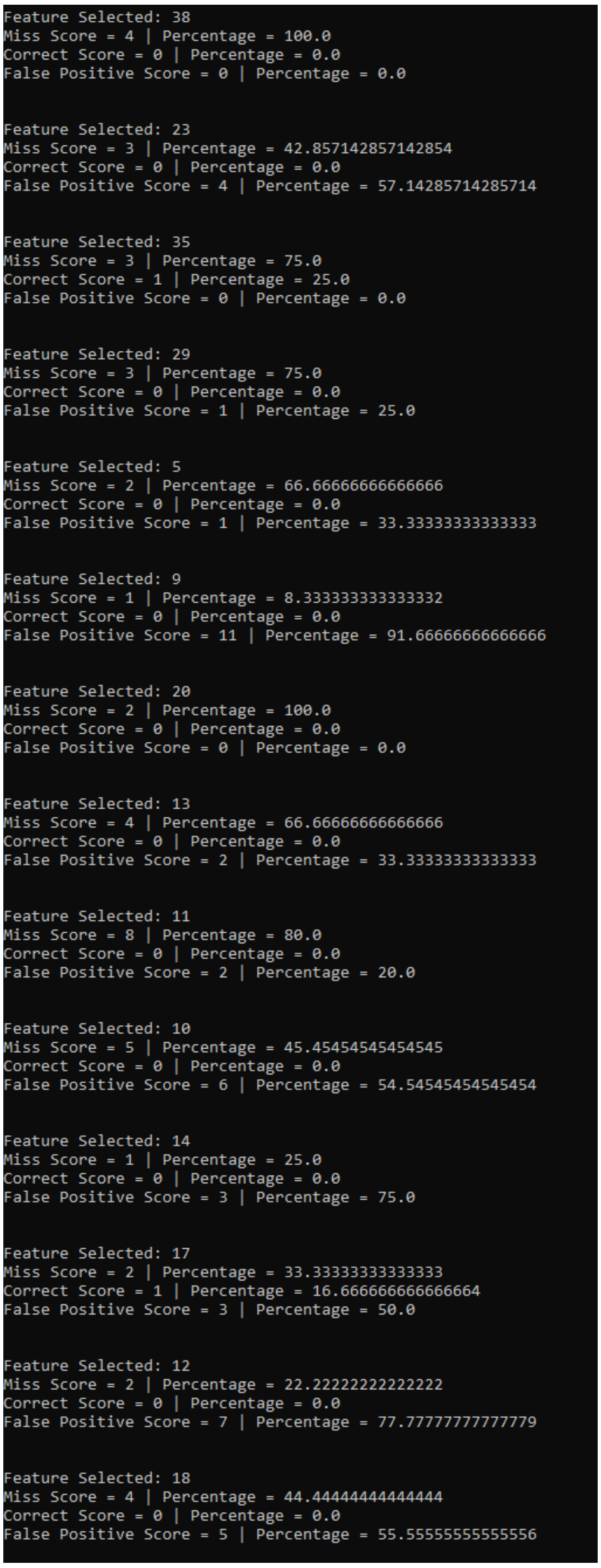


```
E:\Documents\GitHub\CSE515Phase3>python tasks\task_2.py

Please Provide Image Folder Path for TrainingE:\Downloads\500\500

Please Provide Image Folder Path for TestingE:\Downloads\100\100

Enter Feature Model Names From
- color_moment
- local_binary_pattern
- histogram_of_oriented_gradients

Please Enter Model Name: histogram_of_oriented_gradients

Enter Dimensionality Reduction Model Names From
- PCA
- ALL

Please Enter Model Name: ALL

Enter Image Classifier Names From
- SVM
- DT
- PPR

Please Enter Classifier Name: PPR

Please Enter K-Value: 5



STARTING TASK 2...



Fetching Training images from folder: E:\Downloads\500\500
All Training images are fetched.
Applying histogram_of_oriented_gradients Feature Descriptors on Testing images
```

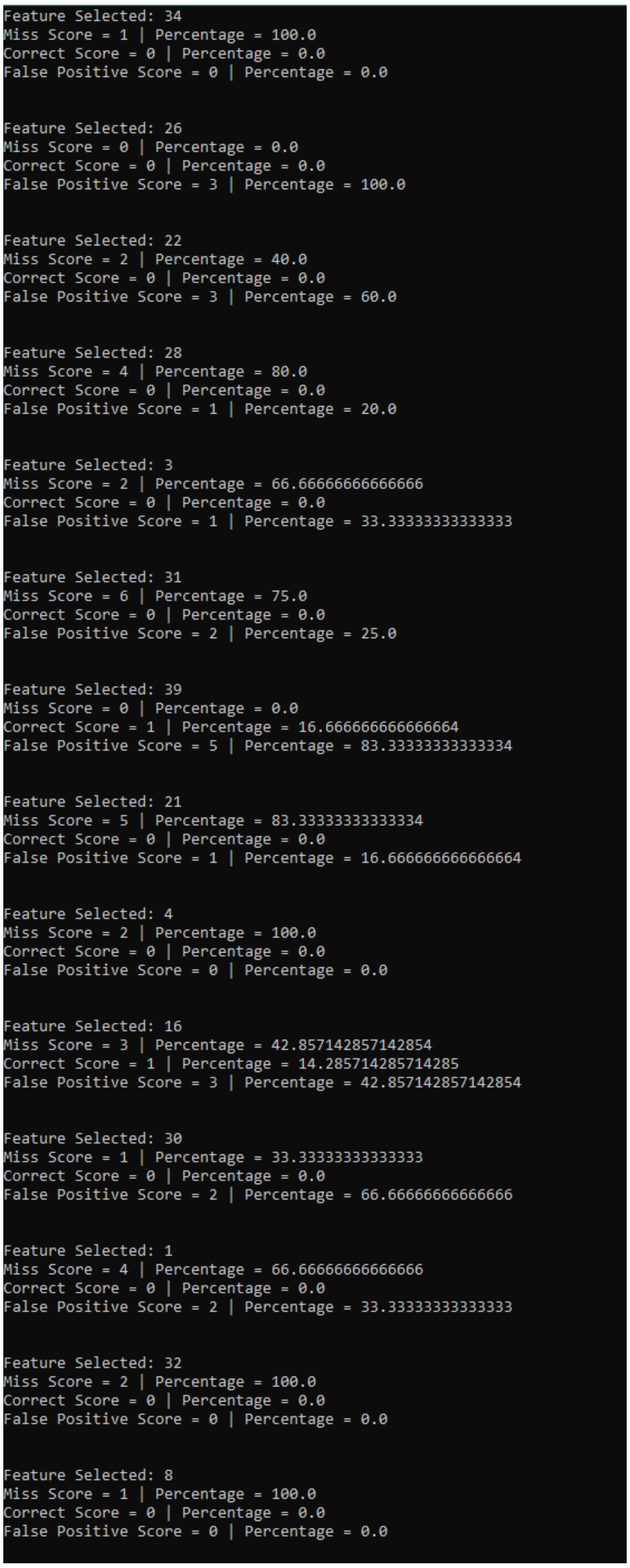


```
Similarity matrix complete
[illegible]

[577 rows x 577 columns]
Calculating similarity graph
Similarity graph created.
Starting ppr score calculation
Calculating ppr for node scores
Label of node: test_0 predicted to be: [illegible]
[illegible]
```

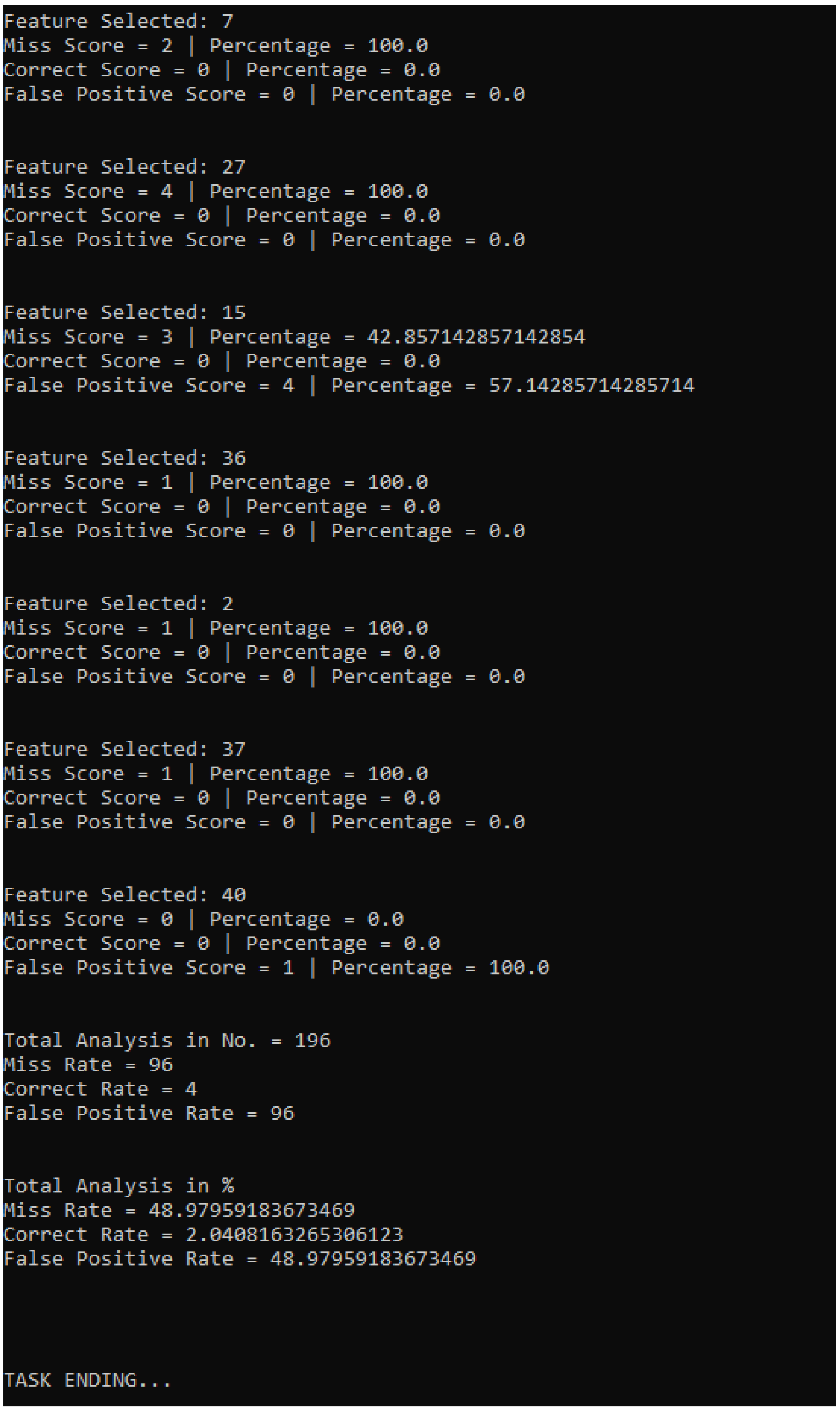
```
[illegible]
PPR Classifications generated, returning...
Actual = [illegible]
Predicted = [illegible]



Analysis Results



Feature Selected: [illegible]
Miss Score = [illegible] | Percentage = 100.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: [illegible]
Miss Score = [illegible] | Percentage = [illegible]
Correct Score = 0 | Percentage = 0.0
False Positive Score = [illegible] | Percentage = [illegible]


Feature Selected: [illegible]
Miss Score = [illegible] | Percentage = [illegible]
Correct Score = 0 | Percentage = 0.0
False Positive Score = [illegible] | Percentage = [illegible]
```



```
Feature Selected: 38
Miss Score = 4 | Percentage = 100.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: 23
Miss Score = 3 | Percentage = 42.857142857142854
Correct Score = 0 | Percentage = 0.0
False Positive Score = 4 | Percentage = 57.14285714285714


Feature Selected: 35
Miss Score = 3 | Percentage = 75.0
Correct Score = 1 | Percentage = 25.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: 29
Miss Score = 3 | Percentage = 75.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 1 | Percentage = 25.0


Feature Selected: 5
Miss Score = 2 | Percentage = 66.66666666666666
Correct Score = 0 | Percentage = 0.0
False Positive Score = 1 | Percentage = 33.33333333333333


Feature Selected: 9
Miss Score = 1 | Percentage = 8.333333333333332
Correct Score = 0 | Percentage = 0.0
False Positive Score = 11 | Percentage = 91.66666666666666


Feature Selected: 20
Miss Score = 2 | Percentage = 100.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: 13
Miss Score = 4 | Percentage = 66.66666666666666
Correct Score = 0 | Percentage = 0.0
False Positive Score = 2 | Percentage = 33.33333333333333


Feature Selected: 11
Miss Score = 8 | Percentage = 80.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 2 | Percentage = 20.0


Feature Selected: 10
Miss Score = 5 | Percentage = 45.45454545454545
Correct Score = 0 | Percentage = 0.0
False Positive Score = 6 | Percentage = 54.54545454545454


Feature Selected: 14
Miss Score = 1 | Percentage = 25.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 3 | Percentage = 75.0


Feature Selected: 17
Miss Score = 2 | Percentage = 33.33333333333333
Correct Score = 1 | Percentage = 16.666666666666664
False Positive Score = 3 | Percentage = 50.0


Feature Selected: 12
Miss Score = 2 | Percentage = 22.22222222222222
Correct Score = 0 | Percentage = 0.0
False Positive Score = 7 | Percentage = 77.77777777777779


Feature Selected: 18
Miss Score = 4 | Percentage = 44.44444444444444
Correct Score = 0 | Percentage = 0.0
False Positive Score = 5 | Percentage = 55.55555555555556
```



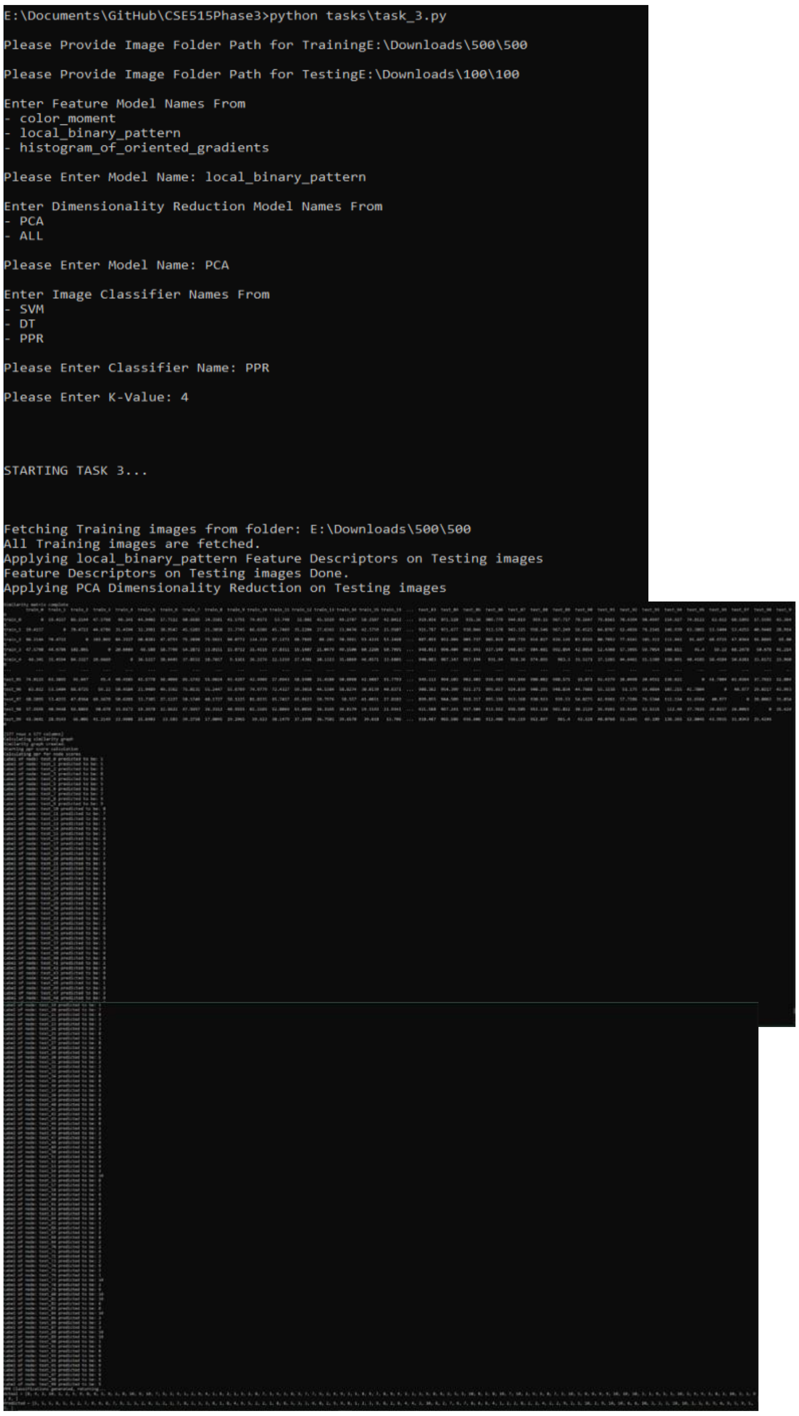
```
Feature Selected: 34
Miss Score = 1 | Percentage = 100.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: 26
Miss Score = 0 | Percentage = 0.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 3 | Percentage = 100.0


Feature Selected: 22
Miss Score = 2 | Percentage = 40.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 3 | Percentage = 60.0


Feature Selected: 28
Miss Score = 4 | Percentage = 80.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 1 | Percentage = 20.0


Feature Selected: 3
Miss Score = 2 | Percentage = 66.66666666666666
Correct Score = 0 | Percentage = 0.0
False Positive Score = 1 | Percentage = 33.33333333333333


Feature Selected: 31
Miss Score = 6 | Percentage = 75.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 2 | Percentage = 25.0


Feature Selected: 39
Miss Score = 0 | Percentage = 0.0
Correct Score = 1 | Percentage = 16.666666666666664
False Positive Score = 5 | Percentage = 83.33333333333334


Feature Selected: 21
Miss Score = 5 | Percentage = 83.33333333333334
Correct Score = 0 | Percentage = 0.0
False Positive Score = 1 | Percentage = 16.666666666666664


Feature Selected: 4
Miss Score = 2 | Percentage = 100.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: 16
Miss Score = 3 | Percentage = 42.857142857142854
Correct Score = 1 | Percentage = 14.285714285714285
False Positive Score = 3 | Percentage = 42.857142857142854


Feature Selected: 30
Miss Score = 1 | Percentage = 33.33333333333333
Correct Score = 0 | Percentage = 0.0
False Positive Score = 2 | Percentage = 66.66666666666666


Feature Selected: 1
Miss Score = 4 | Percentage = 66.66666666666666
Correct Score = 0 | Percentage = 0.0
False Positive Score = 2 | Percentage = 33.33333333333333


Feature Selected: 32
Miss Score = 2 | Percentage = 100.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: 8
Miss Score = 1 | Percentage = 100.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 0 | Percentage = 0.0
```



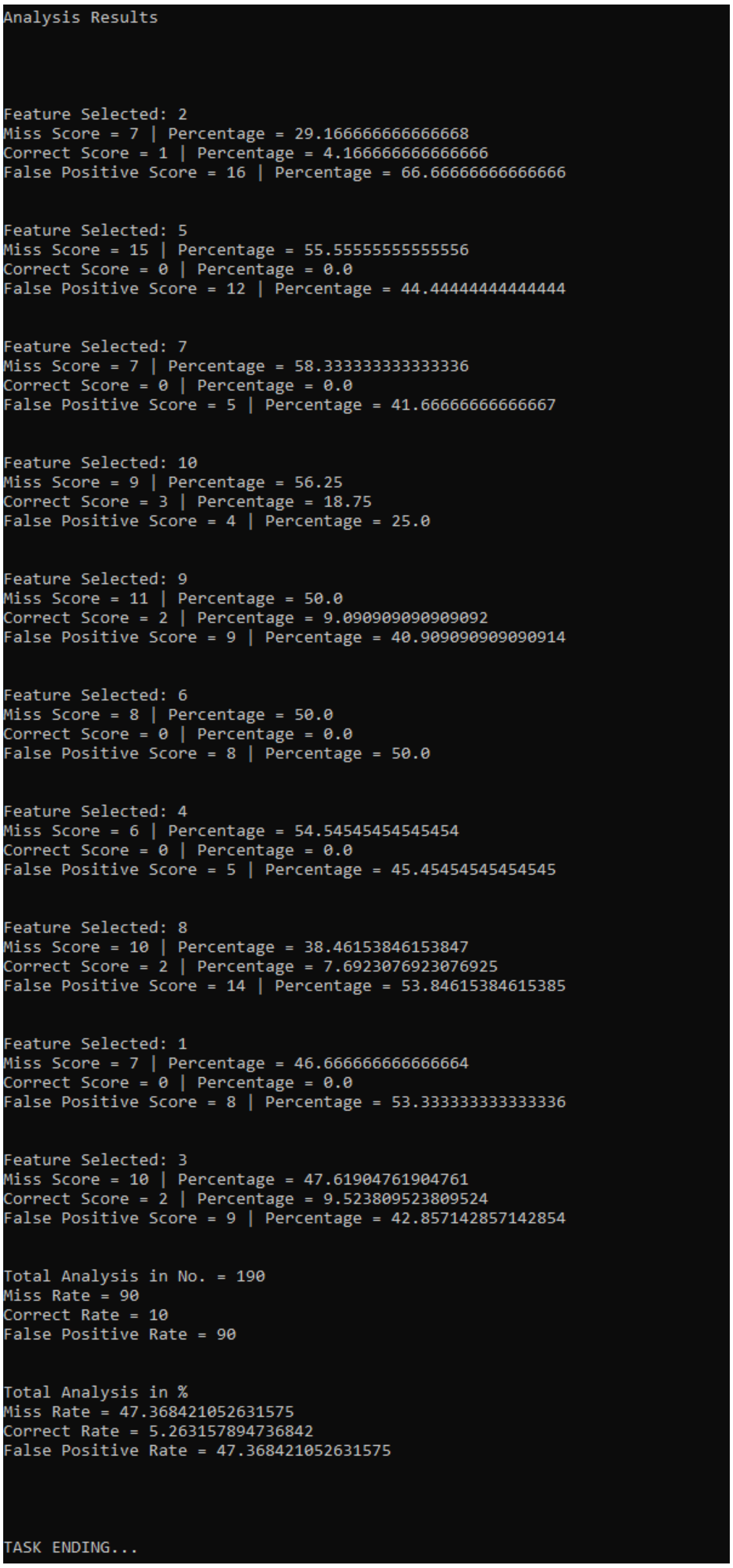
```
Feature Selected: 7
Miss Score = 2 | Percentage = 100.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: 27
Miss Score = 4 | Percentage = 100.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: 15
Miss Score = 3 | Percentage = 42.857142857142854
Correct Score = 0 | Percentage = 0.0
False Positive Score = 4 | Percentage = 57.14285714285714


Feature Selected: 36
Miss Score = 1 | Percentage = 100.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: 2
Miss Score = 1 | Percentage = 100.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: 37
Miss Score = 1 | Percentage = 100.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 0 | Percentage = 0.0


Feature Selected: 40
Miss Score = 0 | Percentage = 0.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 1 | Percentage = 100.0


Total Analysis in No. = 196
Miss Rate = 96
Correct Rate = 4
False Positive Rate = 96


Total Analysis in %
Miss Rate = 48.97959183673469
Correct Rate = 2.0408163265306123
False Positive Rate = 48.97959183673469




TASK ENDING...
```

Task 3

For this task we take multiple parameters as input. First, we take the image folder path for training classifier and next parameter of image folder for testing the classifier. Next, we take feature descriptor and reduction algorithm as input. In order to have all dimensions included we use “ALL” in dimensionality reduction algorithm name. After this we specify classifier from SVM, DT, PPR and provide k-value (number of dimensions).



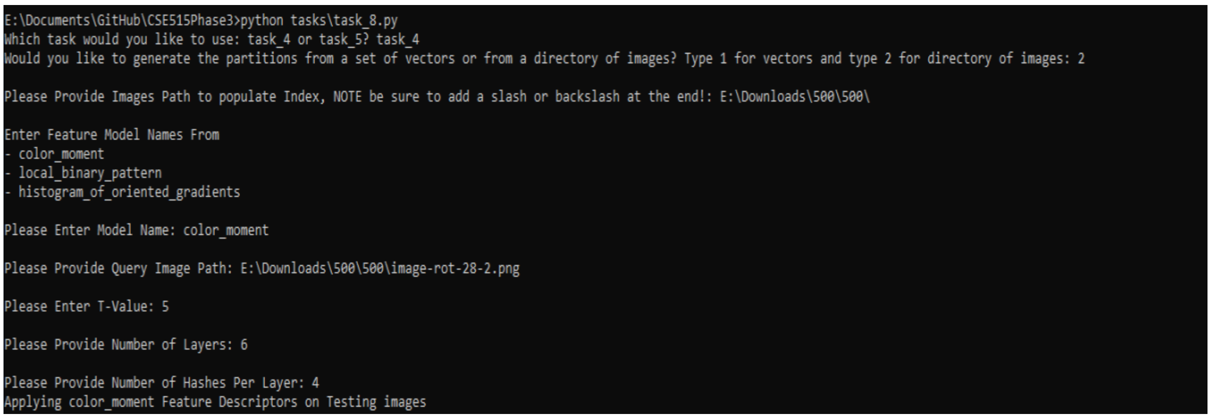
```
E:\Documents\GitHub\CSE515Phase3>python tasks\task_3.py

Please Provide Image Folder Path for TrainingE:\Downloads\500\500

Please Provide Image Folder Path for TestingE:\Downloads\100\100

Enter Feature Model Names From
- color_moment
- local_binary_pattern
- histogram_of_oriented_gradients

Please Enter Model Name: local_binary_pattern

Enter Dimensionality Reduction Model Names From
- PCA
- ALL

Please Enter Model Name: PCA

Enter Image Classifier Names From
- SVM
- DT
- PPR

Please Enter Classifier Name: PPR

Please Enter K-Value: 4



STARTING TASK 3...


Fetching Training images from folder: E:\Downloads\500\500
All Training images are fetched.
Applying local_binary_pattern Feature Descriptors on Testing images
Feature Descriptors on Testing images Done.
Applying PCA Dimensionality Reduction on Testing images
Similarity matrix complete
[illegible]

[177 rows x 177 columns]
Calculating similarity graph
Similarity graph created
Starting ppr score calculation
Calculating ppr for node scores
[illegible]
```



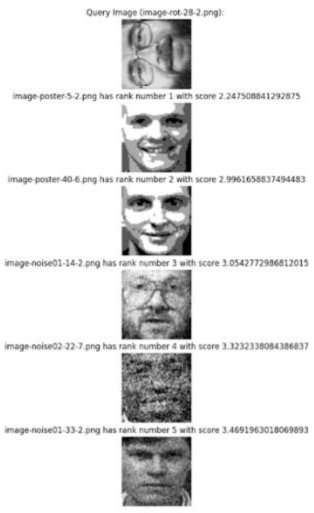
```
Analysis Results



Feature Selected: 2
Miss Score = 7 | Percentage = 29.166666666666668
Correct Score = 1 | Percentage = 4.166666666666666
False Positive Score = 16 | Percentage = 66.66666666666666


Feature Selected: 5
Miss Score = 15 | Percentage = 55.55555555555556
Correct Score = 0 | Percentage = 0.0
False Positive Score = 12 | Percentage = 44.44444444444444


Feature Selected: 7
Miss Score = 7 | Percentage = 58.333333333333336
Correct Score = 0 | Percentage = 0.0
False Positive Score = 5 | Percentage = 41.66666666666667


Feature Selected: 10
Miss Score = 9 | Percentage = 56.25
Correct Score = 3 | Percentage = 18.75
False Positive Score = 4 | Percentage = 25.0


Feature Selected: 9
Miss Score = 11 | Percentage = 50.0
Correct Score = 2 | Percentage = 9.090909090909092
False Positive Score = 9 | Percentage = 40.909090909090914


Feature Selected: 6
Miss Score = 8 | Percentage = 50.0
Correct Score = 0 | Percentage = 0.0
False Positive Score = 8 | Percentage = 50.0


Feature Selected: 4
Miss Score = 6 | Percentage = 54.54545454545454
Correct Score = 0 | Percentage = 0.0
False Positive Score = 5 | Percentage = 45.45454545454545


Feature Selected: 8
Miss Score = 10 | Percentage = 38.46153846153847
Correct Score = 2 | Percentage = 7.6923076923076925
False Positive Score = 14 | Percentage = 53.84615384615385


Feature Selected: 1
Miss Score = 7 | Percentage = 46.666666666666664
Correct Score = 0 | Percentage = 0.0
False Positive Score = 8 | Percentage = 53.333333333333336


Feature Selected: 3
Miss Score = 10 | Percentage = 47.61904761904761
Correct Score = 2 | Percentage = 9.523809523809524
False Positive Score = 9 | Percentage = 42.857142857142854


Total Analysis in No. = 190
Miss Rate = 90
Correct Rate = 10
False Positive Rate = 90


Total Analysis in %
Miss Rate = 47.368421052631575
Correct Rate = 5.263157894736842
False Positive Rate = 47.368421052631575




TASK ENDING...
```

Task 4

In order to execute task 5, the task_8.py script must be executed, to do this execute "python3 tasks/task_8.py". Upon running this command input "task_4" to specify the execution of task 4 from the interface. Then input either "1" or "2" for the decision input to decide whether the partition structure should be generated from a vector file or from a directory of images. Then input the path to the directory of images. Next, input the name of the model that you want to use, the options for this are "color_moment", "local_binary_pattern", or "histogram_of_oriented_gradients". Then specify the path to the query image and the t value, these will be used as the query image and t for the nearest search inside of the partition structure. Lastly, input the layers and hash value sthat will be used to generate the partitions inside of the LSH. Upon completion the output will be both put into the terminal and displayed visually. The specific output that was specified in the task description section for task 4 will be displayed. Then the top t results will be displayed visually for the user.



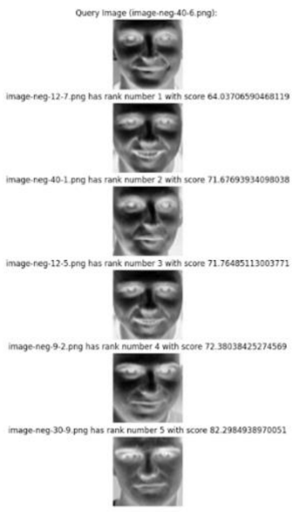
```
E:\Documents\GitHub\CSE515Phase3>python tasks\task_8.py
Which task would you like to use: task_4 or task_5? task_4
Would you like to generate the partitions from a set of vectors or from a directory of images? Type 1 for vectors and type 2 for directory of images: 2

Please Provide Images Path to populate Index, NOTE be sure to add a slash or backslash at the end!: E:\Downloads\500\500\

Enter Feature Model Names From
- color_moment
- local_binary_pattern
- histogram_of_oriented_gradients

Please Enter Model Name: color_moment

Please Provide Query Image Path: E:\Downloads\500\500\image-rot-28-2.png

Please Enter T-Value: 5

Please Provide Number of Layers: 6

Please Provide Number of Hashes Per Layer: 4
Applying color_moment Feature Descriptors on Testing images
```



```
Query Image (image-rot-28-2.png):
image-poster-5-2.png has rank number 1 with score 2.247508841292875
image-poster-40-6.png has rank number 2 with score 2.9961658837494483
image-noise01-14-2.png has rank number 3 with score 3.0542772986812015
image-noise02-22-7.png has rank number 4 with score 3.3232338084386837
image-noise01-33-2.png has rank number 5 with score 3.4691963018069893
```

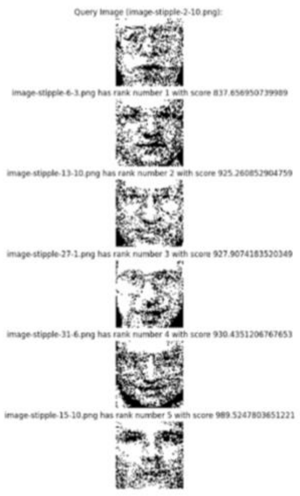
Task 5

In order to execute task 5, the task_8.py script must be executed, to do this execute "python3 tasks/task_8.py". Upon running this command input "task_5" to specify the execution of task 5 from the interface. Then input either "1" or "2" for the decision input to decide whether the partition structure should be generated from a vector file or from a directory of images. Then input the path to the directory of images. Next, input the name of the model that you want to use, the options for this are "color_moment", "local_binary_pattern", or "histogram_of_oriented_gradients". Then specify the path to the query image and the t value, these will be used as the query image and t for the nearest search inside of the partition structure. Lastly, input the b value that will be used to generate the partitions inside of the VA-File. Upon completion the output will be both put into the terminal and displayed visually. The specific output that was specified in the task description section for task 5 will be displayed. Then the top t results will be displayed visually for the user.

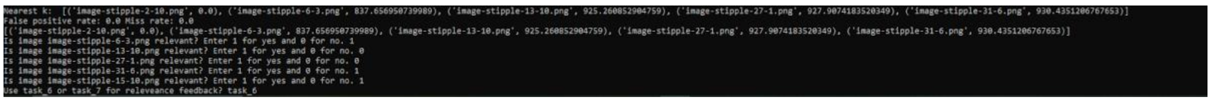

```
Query Image (image-neg-40-6.png):
image-neg-12-7.png has rank number 1 with score 64.03706590468119
image-neg-40-1.png has rank number 2 with score 71.67693934098038
image-neg-12-5.png has rank number 3 with score 71.76485113003771
image-neg-9-2.png has rank number 4 with score 72.38038425274569
image-neg-30-9.png has rank number 5 with score 82.2984938970051
```

Task 6

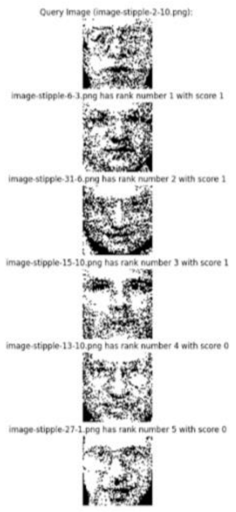
To run Task 6, the Task 8 interface must be used. This may be started on the command line via the command “python tasks/task_8.py”. From this, the nearest neighbor results must be generated via Task 4 or Task 5; these tasks may be selected by typing “task_4” or “task_5” when prompted. Following this, parameters for the chosen task should be entered. These parameters are the input modality, image directory path, feature model name, query image path, t value, and task parameters (relative to Task 4 and Task 5). These are discussed in further detail in the Task 8 Execution Instructions below. Once these parameters are entered, an output will be generated shortly thereafter. The user must analyze these results and classify each nearest neighbor as either relevant or irrelevant. This may be done by passing a “1” or “0” to the command line when prompted for each image. Then, the user must type “task_6” to execute Task 6. Task 6 will utilize the trained decision tree to reorder the top t nearest neighbors by using their relevancy scores.



```
Query Image (image-stipple-2-10.png):
image-stipple-6-3.png has rank number 1 with score 837.656950739989
image-stipple-13-10.png has rank number 2 with score 925.260852904759
image-stipple-27-1.png has rank number 3 with score 927.9074183520349
image-stipple-31-6.png has rank number 4 with score 930.4351206767653
image-stipple-15-10.png has rank number 5 with score 989.5247803651221
```



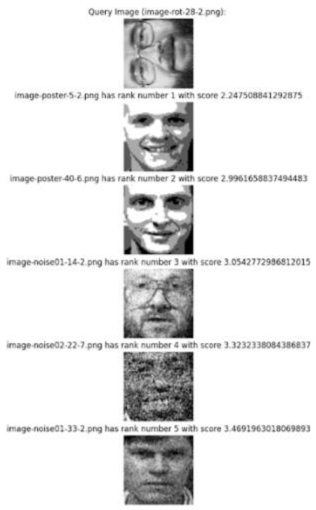
```
Nearest k:  [('image-stipple-2-10.png', 0.0), ('image-stipple-6-3.png', 837.656950739989), ('image-stipple-13-10.png', 925.260852904759), ('image-stipple-27-1.png', 927.9074183520349), ('image-stipple-31-6.png', 930.4351206767653)]
False positive rate: 0.0 Miss rate: 0.0
[('image-stipple-2-10.png', 0.0), ('image-stipple-6-3.png', 837.656950739989), ('image-stipple-13-10.png', 925.260852904759), ('image-stipple-27-1.png', 927.9074183520349), ('image-stipple-31-6.png', 930.4351206767653)]
Is image image-stipple-6-3.png relevant? Enter 1 for yes and 0 for no. 1
Is image image-stipple-13-10.png relevant? Enter 1 for yes and 0 for no. 0
Is image image-stipple-27-1.png relevant? Enter 1 for yes and 0 for no. 0
Is image image-stipple-31-6.png relevant? Enter 1 for yes and 0 for no. 1
Is image image-stipple-15-10.png relevant? Enter 1 for yes and 0 for no. 1
Use task_6 or task_7 for releveance feedback? task_6
```



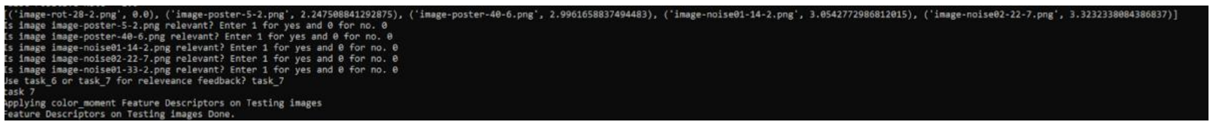
```
Query Image (image-stipple-2-10.png):
image-stipple-6-3.png has rank number 1 with score 1
image-stipple-31-6.png has rank number 2 with score 1
image-stipple-15-10.png has rank number 3 with score 1
image-stipple-13-10.png has rank number 4 with score 0
image-stipple-27-1.png has rank number 5 with score 0
```

Task 7

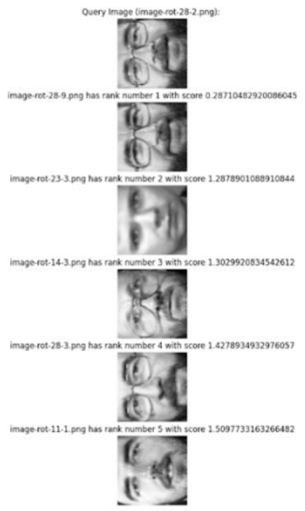
Task 7 also runs off of the Task 8 interface, started via “python tasks/task_8.py” from the command line. The nearest neighbor result parameters should be passed and generated using task 4 or 5 and then the relevance inputs should be given via a 1 or 0 input to the command line for each image in the prompts. Next, when prompted for the relevance feedback task the user should type “task_7” to use task 7. This will reorder the results using SVM based on the given relevance input by the user.



```
Query Image (image-rot-28-2.png):
image-poster-5-2.png has rank number 1 with score 2.247508841292875
image-poster-40-6.png has rank number 2 with score 2.9961658837494483
image-noise01-14-2.png has rank number 3 with score 3.0542772986812015
image-noise02-22-7.png has rank number 4 with score 3.3232338084386837
image-noise01-33-2.png has rank number 5 with score 3.4691963018069893
```



```
[('image-rot-28-2.png', 0.0), ('image-poster-5-2.png', 2.247508841292875), ('image-poster-40-6.png', 2.9961658837494483), ('image-noise01-14-2.png', 3.0542772986812015), ('image-noise02-22-7.png', 3.3232338084386837)]
Is image image-poster-5-2.png relevant? Enter 1 for yes and 0 for no. 0
Is image image-poster-40-6.png relevant? Enter 1 for yes and 0 for no. 0
Is image image-noise01-14-2.png relevant? Enter 1 for yes and 0 for no. 0
Is image image-noise02-22-7.png relevant? Enter 1 for yes and 0 for no. 0
Is image image-noise01-33-2.png relevant? Enter 1 for yes and 0 for no. 0
Use task_6 or task_7 for releveance feedback? task_7
task 7
Applying color_moment Feature Descriptors on Testing images
Feature Descriptors on Testing images Done.
```



```
Query Image (image-rot-28-2.png):
image-rot-28-9.png has rank number 1 with score 0.28710482920086045
image-rot-23-3.png has rank number 2 with score 1.2878901088910844
image-rot-14-3.png has rank number 3 with score 1.3029920834542612
image-rot-28-3.png has rank number 4 with score 1.4278934932976057
image-rot-11-1.png has rank number 5 with score 1.5097733163266482
```

Task 8

This task takes many different inputs since it is serving as an interface for task 4, task 5, task 6, and task 7. Since many of the inputs are repeated from these tasks we will only list the unique input decisions that are related to specifically handled by task 8. To execute this task the task_8.py script must be run. To execute this script please run “python3 tasks/task_8.py” and follow the prompts for user input. The inputs that must be passed into task 8 is the task decision, input decision, images path, model name, query image path, and t value. The task decision takes either “task_4” or “task_5” as input. This input decides which of the partition structures will be used, either task 4 or task 5. The input decision will either be a value of “1” or “2” which decides whether the partition structure is generated from a vector file or from the directory of images. The image path represents the path to the folder containing the relevant images. The model name takes the following values: “color_moments”, “local_binary_pattern”, and “histogram_of_oriented_gradients”. This input decides which of the following feature models is used. The query image path takes a path to an image in the images path that will serve as the query image for the t-nearest-neighbor search in either task 4 or task 5. Lastly, is the t value that is used inside of the t-nearest-neighbor search. Task 8 will then display the output of the user specified task and then visualize the top t results from that task. The user will then need to decide if each of the returned results is relevant to them or not. The last input is the choice to either use task 6 or task 7, this input takes either “task_6” or “task_7”. The relevancy scores and top t results previously calculated are then passed as input into the user specified task. The output from this task is displayed and then task 8 visualizes the top t results from the task.

Related Work

This phase has implemented dimensionality reduction and feature extraction models from the phase-1 using the vector space model. There are various feature descriptor algorithms in use today out which few are:

- DAISY is a feature descriptor like SIFT formulated in a way that allows for fast dense extraction.
- Peak Local Max, find peaks in an image as coordinate list or Boolean mask.
- Binary Robust Independent Elementary Features is an efficient feature point descriptor. It is highly discriminative even when using relatively few bits and is computed using simple intensity difference tests.

In order to calculate the distance among the image vectors we used the Euclidean Distance Formula. There are other various type of distance/similarity matrix that are in use today out of which few are:

- Cosine similarity is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space.
- Manhattan distance is a metric in which the distance between two points is the sum of the absolute differences of their Cartesian coordinates.
- Minkowski distance is a generalization of the Euclidean and Manhattan distances.
- Jaccard similarity is a statistic used for gauging the similarity and diversity of sample sets.

For this phase we use PCA as our dimensionality reduction algorithm. There is various other reduction algorithm that are being used for different cases:

- Factor Analysis is another algorithm that not just reduce the dimensionality of the data. Factor Analysis is a useful approach to find latent variables which are not directly measured in a single variable but rather inferred from other variables in the dataset.
- Kernel PCA is a non-linear dimensionality reduction technique that uses kernels. It can also be considered as the non-linear form of normal PCA.
- MDA is another non-linear dimensionality reduction technique that tries to preserve the distances between instances while reducing the dimensionality of non-linear data.
- Isometric mapping is non-linear dimensionality reduction through Isometric mapping. It is an extension of MDS or Kernel PCA.

For this phase we use SVM, Decision Tree, and PPR as our image classifier algorithms. There are various other classification algorithms that are being used for different cases:

- Convolutional Neural Networks (CNNs) is the most popular neural network model being used for image classification problems.
- Logistic regression is a machine learning algorithm for classification. In this algorithm, the probabilities describing the possible outcomes of a single trial are modelled using a logistic function.

- Naive Bayes algorithm based on Bayes’ theorem with the assumption of independence between every pair of features. Naive Bayes classifiers work well in many real-world situations such as document classification and spam filtering.

For this phase we use LSH and VA-Files as our image clustering algorithms. There is various other clustering algorithms that are being used for different cases:

- Agglomerative Hierarchical Clustering (AHC) is an iterative classification method whose principle is simple. The process starts by calculating the dissimilarity between the N objects. Then two objects which when clustered together minimize a given agglomeration criterion, are clustered together thus creating a class comprising these two objects.
- The K-Means algorithm splits the given dataset into a predefined(K) number of clusters using a particular distance metric. The center of each cluster/group is called the centroid.
- The most popular density-based method is Density-Based Spatial Clustering of Applications with Noise (DBSCAN). It features a well-defined cluster model called “density reachability”. This type of clustering technique connects data points that satisfy density criteria.
- Gaussian mixture model (GMM) is one of the types of distribution-based clustering. These clustering approaches assume data is composed of distributions, such as Gaussian distributions.

Conclusion

Data classification, indexing, search, retrieval, and relevance feedback all play vital roles in multimedia data management. By interlacing these concepts together, one may create a robust and effective database querying and storage system. The various tasks assigned for this project allowed our team to learn of these techniques through practical application. Tasks 1-3 taught us not only how the SVM, Decision Tree, and PPR classifiers function, but also how to utilize these models to assign different types of labels to unknown images through the use of a training set and a testing set. Tasks 4-5 focused on imparting knowledge of indexing through LSH and VA-files and how these may be utilized to perform nearest-neighbor search to retrieve the top t images from the query space. Tasks 6-7 brought classifiers to another possible implementation: relevance feedback. By using SVM or Decision Trees to take in the nearest neighbor results of Tasks 4-5, as well as user relevance scores, we may improve the aforementioned results. Finally, Task 8 acts as an interface for tasks 4-7, so as to improve user experience and streamline the task interaction process. By programming these tasks, we have gained valuable experience with interpreting, querying, and applying multimedia data for further understanding of new data.

Bibliography

1. Candan, K. S. (2021). CSE 515 Multimedia and Web Databases Phase #3 Project Description (pp. 1-2). Published on Piazza
2. Noah Keen. (2005). Color Moments, https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0405/KEEN/
3. L. Mdakane and F. van den Bergh (2001). Extended Local Binary Pattern Features for Improving Settlement Type Classification of QuickBird Images, CSIR Research Space
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. Mathieu Blondel, Akinori Fujino, Naonori Ueda (2014). Large-scale Multiclass Support Vector Machine Training via Euclidean Projection onto the Simplex
6. Alexandr Andoni and Piotr Indyk (2008). Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions
7. Stephen Blott and Roger Weber (1997), "A Simple Vector-Approximation File for Similarity Search in HighDimensional Vector Spaces".

Appendix

During this phase of the project everyone was responsible for implementing certain task and algorithms to keep work divided evenly. Members coordinate to help validate data outputs and discuss a variety of techniques for calculating intermediary values. Code was maintained on GitHub, and we use online word document to collaborate on report writing task

Preston Mott – Task-5, Task-8, VA-Files, Report

Brandon Bayles – Personalized PageRank, Task-6, Task-8, Report

Ian Bolton – Decision Tree, Task-6 (partial), Modification on Task-4, Report

Keenan Rahman – Task-1, Task-2, Task-3, SVM.py, Task-4, Task-7, Report

Kunhao Zhang –

Tanya Aggarwal –