CSE 515: Multimedia and Web Databases - Project Phase 3

Nearest Neighbors using VA,LSH and Relevance Feedback classifiers

Group Members of Group 4	
Glen Dsouza	gsdsouza@asu.edu
Khushal Modi	kcmodi@asu.edu
Manthan Agrawal	magraw12@asu.edu
Nishtha Bhimte	nbhimte@asu.edu
Shivani Priya	spriya6@asu.edu
Sowmith Reddy	rchityal@asu.edu

Abstract

This report contains the details CSE 515: Multimedia and Web Databases project Phase 3 implementation and design details. This phase of the project is focused on image classification, indexing and retrieval. Various approaches to classify the images based on type, subject id and sample id were implemented like SVM classifier, decision tree classifier and PPR based classifier, images are indexed based on LSH and VA files and content based image retrieval is done using Decision-tree-based relevance feedback and SVM-based relevance feedback.

Keywords

Feature Descriptors, Color Moments, Local Binary Patterns, Histogram of Oriented Gradients, PCA, SVD, LDA, K-means, Latent Semantics, Distance, Similarity, Similarity Matrix, PageRank, Seed Node, Personalized Pagerank, Noise Removal, Dimensionality Curse, SVM, Decision Tree, Relevance Feedback, Classification, Indexing, LSH, VA Files.

1. Introduction:

1.1. Terminology

- Media: A way to exchange information.
- Feature Descriptor: Quantified values used to represent properties of a media.
- Similarity Measure: A measure describing the similarity between 2 images.
- Distance Measure: A measure describing the distance between 2 data points in the given vector space.
- Feature Model: Algorithm used to derive feature descriptors of a media.
- Latent Semantics: Feature descriptors derived after applying some transformations on the existing feature descriptors of a media with an aim to reduce redundancy among dimensions.
- PCA: Principal Component Analysis; used for dimensionality reduction when protecting the discrimination power of the feature descriptors.
- SVD: Singular Value Decomposition; algorithm used for dimensionality reduction when minimizing the data perturbation.
- LDA: Latent Dirichlet Analysis; algorithm used for dimensionality reduction when objects are of different sizes.
- K-means: Algorithm used for dimensionality by clustering the data points into K clusters.
- PageRank: Algorithm used to rank objects based on a graphical representation.
- Seed Node - Nodes that are used to bring the context to the graph.
- Personalized Pagerank - Algorithm to calculate the Page rank based on the context of the query.
- Classification - Process to predict the class of an unknown data point based on the labelled examples.
- LSH - Locality Sensitive Hashing
- VA Files - Vector Approximation Files

1.2. Goal Description

The aim of this phase is to learn about vector models, indexing and search on an image database, classification and relevance feedback while using the programs we implemented in phase-1 and phase-2. Phase-3 of the project is as described below →

- **Task1**: 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 classifier
        - A decision tree classifier
        - A PPR based classifier
    - Also compute and print the false positive and miss rates.

- **Task2**: 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 classifier
        - A decision tree classifier
        - A PPR based classifier
    - Also compute and print the false positive and miss rates.

- **Task3**: 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 classifier
        - A decision tree classifier
        - A PPR based classifier
    - Also compute and print the false positive and miss rates.

- **Task4**: Locality-Sensitive Hashing

    - Implement a Locality Sensitive Hashing (LSH) tool, which takes as input (a) the number of layers, L, (b) the number of hashes per layer, k, 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.

    - 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.

- **Task5**: VA-Files

  Implement a VA-file index tool and associated nearest neighbor search operations. The data structures and relevant algorithms are described in the given papers.

  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.

  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.

- **Task6**: Decision-tree-based relevance feedback: Implement a decision tree based relevance feedback system to im- prove 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.

- **Task7**: Content based image retrieval using SVM-classifier-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.

- **Task8**: 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 the user to provide positive and/or negative feedback for the ranked results returned by the system.

1.3. Assumptions

- There will be consistency between the feedback the user makes for relevance feedback so that equal importance is given to each iteration of feedback taken from the user.
- All the input folders would be put inside alongside the src directory and all the absolute paths will be provided in inputs.
- The constants path will be updated prior to the run in the src/constants.py file.
- Users will mark at least one image as relevant and irrelevant when asked for feedback.
- Users will stop the feedback process when it is satisfied with the nearest neighbor result.

2. Implementation Details

This section describes the implementation and design details about the dimensionality reduction techniques, PageRank algorithm, derivation of similarity matrices and top k retrieval algorithms.

2.1 Task 1

Implement a program which, (a) Given a folder of images, (b) one of the three feature models, and (c) 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 classifier
- A decision tree classifier
- A PPR based classifier

Also compute and print the false positive and miss rates.

2.1.1 Image Classification using SVM

Support Vector Machines (SVM) is a classification technique used for binary or multi label classification purposes. We have implemented linear SVM classifiers as discussed in class lectures, to derive label 'X' for the images. Linear SVM classifiers use a line or plane or hyperplane to create a boundary between all the elements which belong to the class and those which do not belong to the class. This is the main idea behind a binary SVM classifier. To define formally [4]→

> "*An SVM classifies data by finding the best hyperplane that separates all data points of one class from those of the other class. The best hyperplane for an SVM means the one with the largest margin between the two classes. Margin means the maximal width of the slab parallel to the hyperplane that has no interior data points.*"

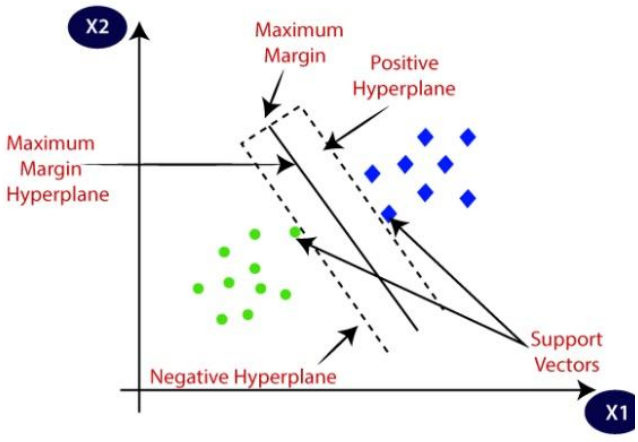


Figure 1. Binary SVM Classifier

As shown in the figure above, a binary SVM classifier creates a hyper plane with maximum margin to generate a line or plane or hyper plane division between the two classes in such a way that the margin is of maximal width.

Linear SVM can be defined mathematically as →

$$f(x) = sign\left(\mathbf{w}^* \cdot x + \mathbf{b}^*\right)$$

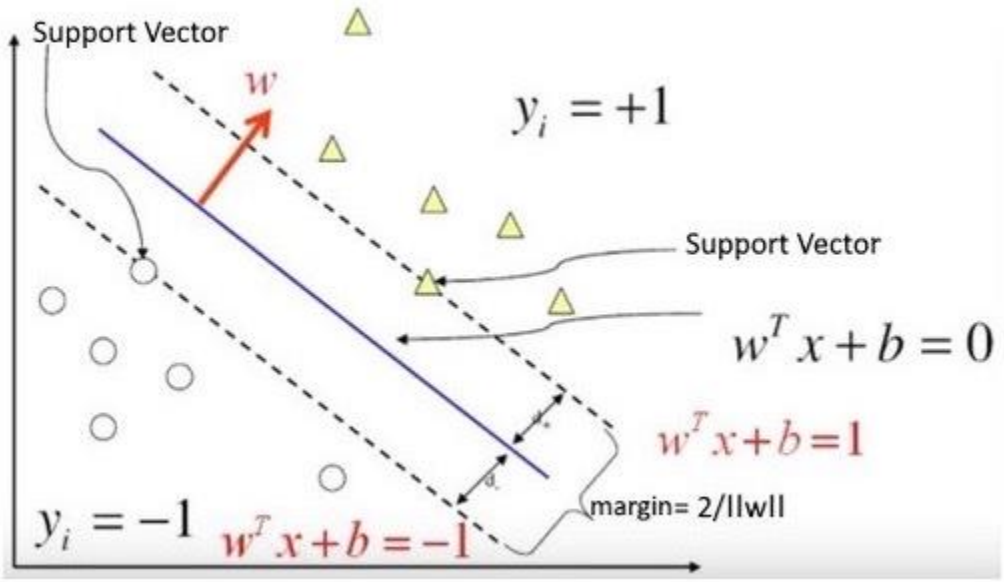


Figure 2. Binary SVM Classification

As shown in the figure above, in the binary SVM classifier, the distance from the hyperplane is used to classify objects to the 2 classes. As shown, in the above figure, any data point with a positive value of SVM equation, will be classified as a triangle otherwise it will be classified as a circle. This is the main idea behind SVM.

A binary SVM classifier works in the following way →

1. Preprocess the train data such that all the points which belong to the class are given a label of 1.0 and all other points are given a label of -1.0
2. Initialize a vector *W* and a scalar *b*
3. Now start the gradient descent on *W* and *b* using the labels and train data.
4. Keep iterating and updating the vector *W* and scalar *b* according to the learning rate and the gradient descent.
5. Stop the iteration once the cost has converged.
6. Now, to classify the data, all the test points having positive value of f(x) lie in the class and all other points lie outside the class or on the other side of the hyperplane created.

However, a multi label linear SVM classifier is a combination of multiple linear SVM classifiers which are combined to generate a single output. In other words, multiple binary classifiers are trained to predict whether an object belongs to a particular class, which are then combined to form a single output based on the distance from the decision boundary. To define multi label classification formally [5]→

> *"In its most basic type, SVM doesn't support multiclass classification. For multiclass classification, the same principle is utilized after breaking down the multi-classification problem into smaller subproblems, all of which are binary classification problems."*

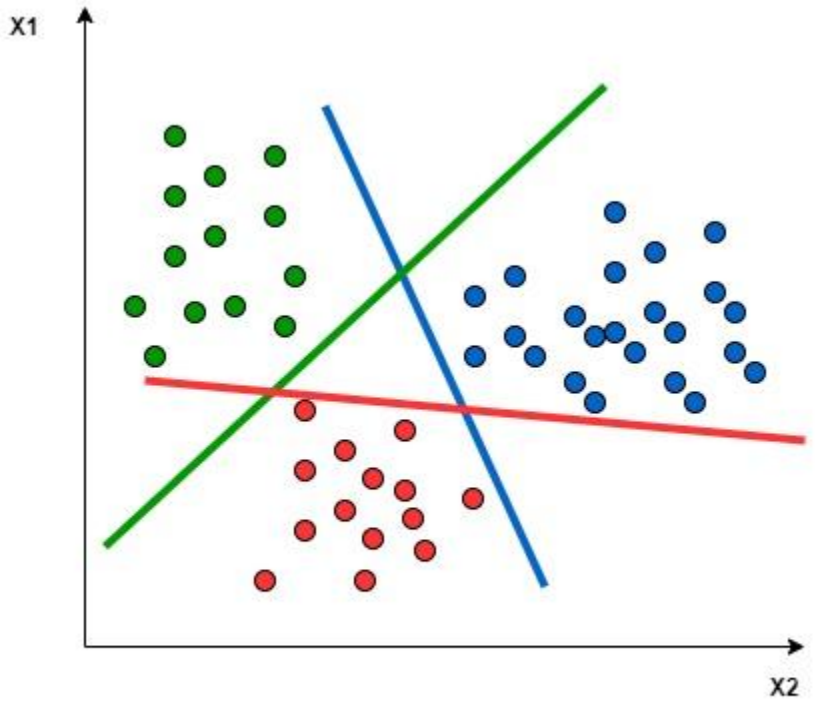


Figure 3. SVM: One VS All Approach

Like explained above, in multiclass classification, we break the problems into smaller problems or in this case, multiple binary classifiers. To assign an X value to the images in the given folder, we train X different binary classifiers to decide whether the label X can be given to the image or not. This approach is shown in the figure above and is known as the "one vs all" approach. The red hyperplane is used to differentiate red points from all other points. Similarly, the green hyperplane is used to differentiate all green points from other points and the blue line is used to differentiate all blue points from other points. This approach however is time consuming because we need to train multiple linear SVMs.

In this phase, we have implemented a linear multiclass SVM classifier with the "one vs all" algorithm mentioned above. Following algorithm is used to determine the label X →

1. Train a binary classifier using the “one vs all” approach for each label. Here, the points belonging to the class are given a label of 1.0 and all other data points are given a label of -1.0.
2. Now use the binary SVM classifier algorithm to determine the hyperplane dividing the data points.
3. Once we have all the binary classifiers as shown in the image above, find the distance of the points from each hyperplane. The test data point will belong to the class whose hyperplane is farthest from the data point or whose value of the f(x) is maximum.

This way, we can combine multiple binary SVM classifiers and form a multiclass classifier. The approach we have used is generating an accuracy of close to 50% for this task. We are calculating the false positive rate and miss rate for each class in the following way →

```
fp = confusion_matrix.sum(axis=0) - np.diag(confusion_matrix)
fn = confusion_matrix.sum(axis=1) - np.diag(confusion_matrix)
tp = np.diag(confusion_matrix)
tn = confusion_matrix.sum() - (fp + fn + tp)
false_pos_rate = fp/(tn+fp+fn+tp)
miss_rate = fn/(fn+tp)
```

Each label has a false positive and a miss rate in this case since we are doing multiclass classification. The confusion matrix and the result calculation is done according to the below image →

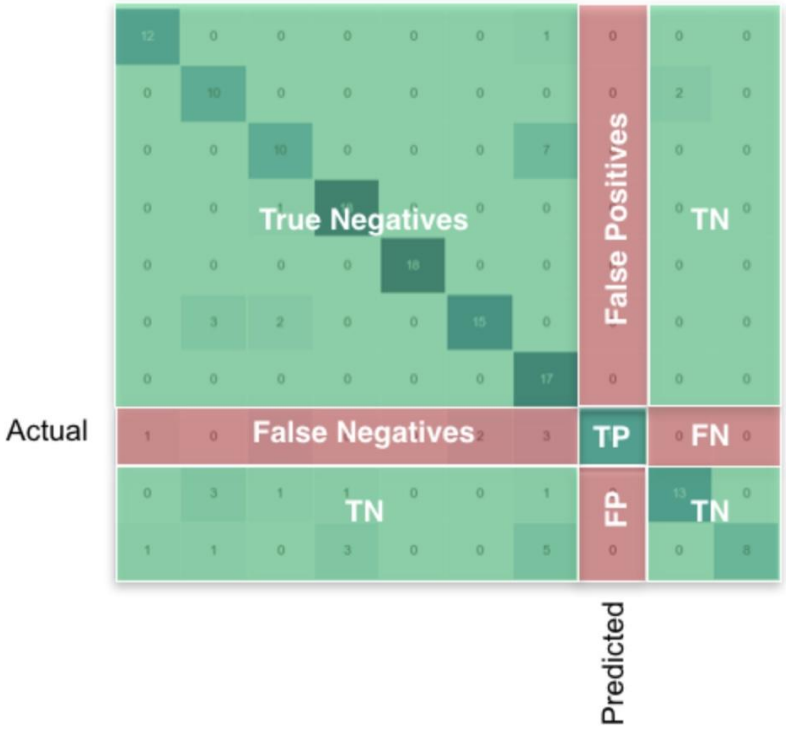


Figure 4. Confusion Matrix for Multilabel Classification

2.1.2 Image Classification using Decision Trees

Decision tree is similar to a flowchart which takes decisions at each level and reaches a final conclusion of the data at the end. It comes under supervised machine learning where the data is continuously split based on the data parameters at each depth. The decision for any given data is present at the leaf node. Hence a decision tree has two components one which is decision nodes which are intermediate nodes and leaf nodes which are the decision.

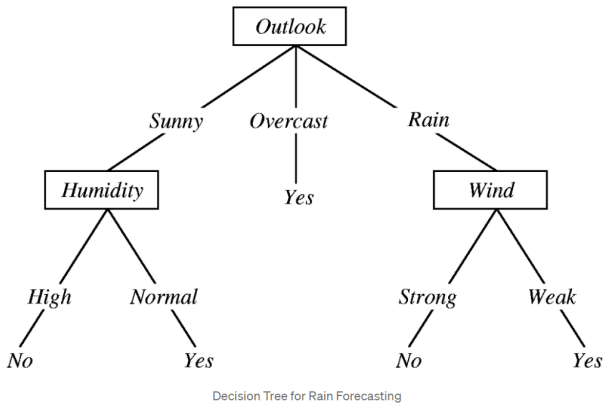


Figure 5. Decision Tree

In the above image[12] humidity, outlook, and wind are the feature variables of the data. Based on the values of that feature we need to forecast whether there will be rain or not. At each level if traverse the node by taking the decision based on the feature variables value. For example If the outlook is sunny we go left then check for humidity and now check for humidity, If humidity is normal we go right and now we are at the leaf node which tells there will be a rain. So we decide to take which path based on the intermediate nodes and once we reach the leaf node we know the outcome of that data.

There are two types of decision trees based on the type of the outcome variable. We use classification trees for categorical outcome data and regression trees if the outcome of the data is continuous. The outcome for our image classification task is categorical since we need to classify each based on the type, subject id and image sample id. So we will be using classification trees and not regression trees.

The decision on which feature variable to use at each level of classification is critical for the decision tree classification. Generally the feature which has the highest classification/descrimination power in the given dataset is chosen as the first node. Now once the data is divided into a number of sets based on this feature value. For each divided data set we again find the feature variable with highest classification power among the remaining features. We repeat this until we exhaust all the features or we reach an outcome.

For implementing our task we choose the CART algorithm(Classification and Regression Trees) which uses Gini impunity to decide which feature is taken for making decisions at each level. Each node is split so that the Gini impurity of the children(more specifically the average of the Gini of the children weighted by their size) is minimized.

Gini impurity measures how impure the node is for a given data set. We always select a feature which divides the data in such a way that the Gini impurity is minimum. [12]

More formally the Gini impurity of n training samples split across k classes is defined as

$$G = 1 - \sum_{k=1}^{n} {p_k}^2$$

For example, if X = [[1], [1.5], [2.1], [2.5], [3]] and y = [0, 0, 1, 1, 2]. If we split the above data using the condition X[0] > 2. Then we would have the Gini Impurity calculated as below[12]:

$$G = \frac{2}{5} G_{left} + \frac{3}{5} G_{right} = \frac{2}{5} \times 0 + \frac{3}{5} \times \left(1 - \frac{2}{3}^2 - \frac{1}{3}^2\right) = 0.27.$$

The Gini impurity calculated for the above split would be minimum among all other splits and also we can clearly see that if we split the above way can at least predict the class zero for the given data.

The important part of the CART algorithm is therefore finding the best split from all possible splits. To do this we need to find all possible splits and find the best split. This would be time consuming and the time complexity would be O(n ^ 2) which is not generally recommended. To optimize this we do as follows:

a) Sort the feature values
b) Iterate through each value as a possible threshold value.
c) Keep track of the number of samples per class on the left and on the right.
d) Increment or decrement them by 1 after each threshold.

For a given feature k we calculate the gini impurity at each split i and we do this for all the features. The feature and the threshold which gives the minimum value is then used for classification. We repeat this until we are done with all features or the required depth of the tree is reached. The images below show the mentioned calculations [12]

$$G_i^{left} = 1 - \sum_{k=1}^{n} \left(\frac{m_k^{left}}{i}\right)^2 \qquad G_i^{right} = 1 - \sum_{k=1}^{n} \left(\frac{m_k^{right}}{m-i}\right)^2 .$$

$$G_i = \frac{i}{m} G_i^{left} + \frac{m-i}{m} G_i^{right}.$$

Results:

As the professor mentioned we ran the decision tree classifier for all the three models and k = 50 and we observed that the results were good for HOG which gave a **false positive rate of 0.018** and **miss rate of 0.18**. From this we can conclude that the classifier was good at falsely not rejecting the type of the images. The **accuracy** of the decision tree classifier was **83%** (it is shown as 0.83 in the below figure). We can see the above results in the below image.

```
Input the image folder for classification: C:\Users\rchityal\Desktop\ASU\ASU\sem2\MWDB\3_phase\Phase3\100
Running PCA...
(888, 50) (888, 1) (100, 50)
FALSE Positive Rate:
0.018483478145954847
MISS RATE:
0.18645526192933315
Accuracy
0.83
```

2.1.3 Image Classification using PPR

Personalized PageRank (PPR) algorithm was implemented as a part of project phase2. The PPR classifier is based on the PPR algorithm and nearest neighbor algorithm implemented in phase2. We have tried 2 different algorithms in our project and went ahead with the one providing a better accuracy.

Method 1 →

1. Convert the training images and testing images to the latent space.
2. From the training images, find the *n* nearest neighbors of each test sample.
3. Now append the test sample at the end of the *n* nearest neighbor sample
4. Create an image - image similarity matrix by multiplying the matrix in step 3 with its transpose.
5. Pass this similarity matrix to the phase2's PPR algorithm to generate a transition matrix
6. Now as a part of the PPR algorithm, to get the top nodes given a seed node, pass the test image as seed node and get the new pagerank.
7. Pick the X label of the top node and assign that label to the test image.
8. Repeat step 2 through step 7 for each test image.

	Latent - 1	Latent - 2
Train image		
Train image		
Train image		
Test image		

Table 1. Sample of the vector used for generating similarity matrix in Step -4

Method 2 →

1. Convert the training images and testing images to the latent space.
2. From the training images, find the *n* nearest neighbors of each test sample.
3. Now append the test sample at the end of the *n* nearest neighbor sample
4. Create an image - image similarity matrix by multiplying the matrix in step 3 with its transpose.
5. Pass this similarity matrix to the phase2's PPR algorithm to generate a transition matrix
6. Now as a part of the PPR algorithm, to get the top nodes given a seed node, pass the test image as seed node and get the new pagerank.
7. Pick the top 3 nodes from the pagerank and compare their similarity with the given test image in the similarity matrix. The image with the highest similarity score is used to assign the label to the test image.
8. Repeat step 2 through step 7 for each test image.

The decision to use the nearest neighbor algorithm from phase2 was made to reduce the size of the similarity matrix and make the overall process faster. Also, since we are computing the similarity matrix only on the nearest neighbors, the chances of getting the correct label increases. Both the approaches look

similar but they differ in the way the pagerank is used to assign the X label to the test image. We went ahead with Method-1 because it leads to better accuracy.

Overall the accuracy of the PPR classifier was pretty low. We can account this to the fact that given the seed node, pagerank is used to rank the nodes based on their importance. However, the pagerank algorithm implemented in phase2 cannot be used to measure similarity between the images or to classify the images. This also depends on the latent semantics used to derive the pagerank transition matrix. Since it contains the nearest neighbors of the query image, we can prune the space but not that effectively because the latent semantics contains images from the entire folder and not just a particular type, subject or orientation.

The results of the PPR classifier in predicting X are shown below →

```
/Users/khushalmodi/opt/anaconda3/lib/python3.8/site-packages/pandas/core/fr
ame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retur
ning-a-view-versus-a-copy
  return super().drop(
Setting up the classifier...
accuracy on test dataset: 0.11
```

2.2 Task 2

Implement a program which, (a) Given a folder of images, (b) one of the three feature models, and (c) 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 classifier
- A decision tree classifier
- A PPR based classifier

Also compute and print the false positive and miss rates.

2.2.1 Implementation

All the algorithms used in this task are the same as the ones mentioned in *2.1 Task 1*. In this task though, instead of predicting the label type X for the image, we are predicting label subject Y for each test image. All the modules and data preprocessing along with the model parameters remain the same as the previous task.

SVM for predicting label subject Y uses subject Y from the train images to classify the test images to subclasses using the multiclass SVM classifier implemented in this phase. Decision tree classifier also uses the same technique mentioned in task1 where we built the decision tree using the training data to classify the images into different subjects and then predict the subject id for the given test images. PPR uses the similar technique as the previous task where we prune the search space using the nearest neighbor technique and then apply the PPR algorithm to classify the image to one subject Y.

2.2.2 Results

Decision Tree:

As the professor mentioned, we ran the decision tree classifier for all the three models and k = 50 and we observed that the results were good for HOG which gave a **false positive rate of 0.019** and **miss rate of 0.74**. From this we can conclude that the classifier was good at falsely not rejecting the type of the images. The **accuracy** of the decision tree classifier was **33%** (it is shown as 0.33 in the below figure). The accuracy with ELBP was very good but it had a very high miss rate so we concluded that the results are good with HOG. We can see the above results in the below image.

```
Input the image folder for classification: C:\Users\rchityal\Desktop\ASU\ASU\sem2\MWDB\3_phase\Phase3\100
Loading file  C://Users//rchityal//Desktop//ASU//ASU//sem2//MWDB//3_phase//Phase3//outputs\2021-11-28-18-03-33\rev-50-HO
G.npy
No Need to run the dimensionality reduction...
(888, 50) (888, 1) (100, 50)
FALSE Positive Rate:
0.01925313815738644
MISS RATE:
0.745515873015873
Accuracy
0.33
```

SVM

The accuracy of the SVM classifier lies somewhere close to 10%. The low accuracy of the classifier here can be due to the fact that here we are calculating latent semantics over the entire images folder and not just specific to a type of subject.

False positive rates ad miss rates for each class-->

```
Class:  1  fp rate:  0.1912479740680713  miss rate:  0.631578947368421
Class:  10  fp rate:  0.0  miss rate:  1.0
Class:  11  fp rate:  0.0  miss rate:  0.9444444444444444
Class:  12  fp rate:  0.0  miss rate:  1.0
Class:  13  fp rate:  0.0  miss rate:  1.0
Class:  14  fp rate:  0.008103727714748784  miss rate:  0.47058823529411764
Class:  15  fp rate:  0.0  miss rate:  1.0
Class:  16  fp rate:  0.0  miss rate:  1.0
Class:  17  fp rate:  0.0  miss rate:  1.0
Class:  18  fp rate:  0.0  miss rate:  0.9565217391304348
Class:  19  fp rate:  0.0  miss rate:  1.0
Class:  2  fp rate:  0.0  miss rate:  0.9090909090909091
Class:  20  fp rate:  0.0  miss rate:  0.8333333333333334
Class:  21  fp rate:  0.20583468395461912  miss rate:  0.0
Class:  22  fp rate:  0.0  miss rate:  1.0
Class:  23  fp rate:  0.0  miss rate:  1.0
Class:  24  fp rate:  0.0  miss rate:  1.0
Class:  25  fp rate:  0.0  miss rate:  0.9166666666666666
Class:  26  fp rate:  0.406807131280389  miss rate:  0.16666666666666666
Class:  27  fp rate:  0.0016207455429497568  miss rate:  0.875
```

```
Class:  28  fp rate:  0.0  miss rate:  0.9
Class:  29  fp rate:  0.0  miss rate:  1.0
Class:  3  fp rate:  0.0  miss rate:  1.0
Class:  30  fp rate:  0.06482982171799027  miss rate:  0.38095238095238093
Class:  31  fp rate:  0.0  miss rate:  1.0
Class:  32  fp rate:  0.0  miss rate:  1.0
Class:  33  fp rate:  0.0  miss rate:  0.85
Class:  34  fp rate:  0.0  miss rate:  0.8666666666666667
Class:  35  fp rate:  0.0  miss rate:  1.0
Class:  36  fp rate:  0.0  miss rate:  1.0
Class:  37  fp rate:  0.008103727714748784  miss rate:  0.6363636363636364
Class:  38  fp rate:  0.0  miss rate:  1.0
Class:  39  fp rate:  0.0  miss rate:  1.0
Class:  4  fp rate:  0.0  miss rate:  1.0
Class:  40  fp rate:  0.0032414910858995136  miss rate:  0.9375
Class:  5  fp rate:  0.0  miss rate:  1.0
Class:  6  fp rate:  0.0  miss rate:  1.0
Class:  7  fp rate:  0.0  miss rate:  0.875
Class:  8  fp rate:  0.0  miss rate:  0.8333333333333334
Class:  9  fp rate:  0.0  miss rate:  1.0
accuracy on test dataset: 0.11021069692058347
```

2.3 Task 3

Implement a program which, (a) Given a folder of images, (b) one of the three feature models, and (c) 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 classifier
- A decision tree classifier
- A PPR based classifier

Also compute and print the false positive and miss rates.

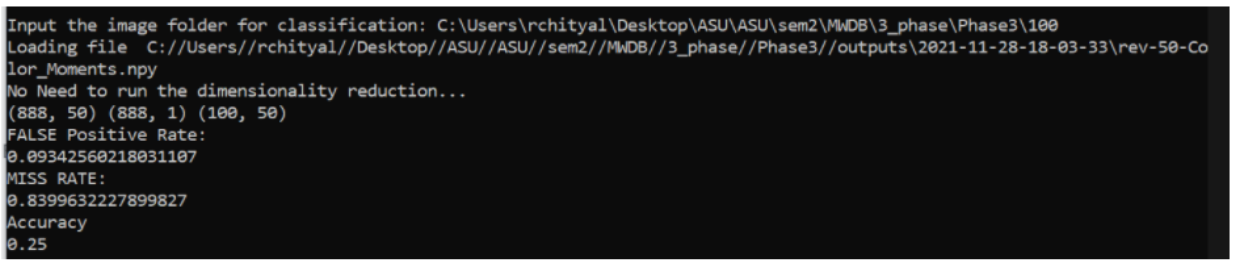
2.3.1 Implementation and Results

All the algorithms used in this task are the same as the ones mentioned in *2.1 Task 1*. In this task though, instead of predicting the label type X for the image, we are predicting label orientation Z for each test image. All the modules and data preprocessing along with the model parameters remain the same as the previous task.

SVM for predicting label orientation Z uses orientation Z from the train images to classify the test images to subclasses using the multiclass SVM classifier implemented in this phase. Decision tree classifier also uses the same technique mentioned in task1 where we built the decision tree using the training data to classify the images into different sample ids and then predict the sample id for the given test images. PPR uses the similar technique as the previous task where we prune the search space using the nearest neighbor technique and then apply the PPR algorithm to classify the image to one orientation Z.

2.3.2 Results

Decision Tree:
As the professor mentioned we ran the decision tree classifier for all the three models and k = 50 and we observed that the results were good for Color moments which gave a **false positive rate of 0.093** and **miss rate of 0.83**. From this we can conclude that the classifier was good at falsely not rejecting the type of the images. The **accuracy** of the decision tree classifier was **25%** (it is shown as 0.25 in the below figure). The miss rate and accuracy for this task is less when compared to the other two classification tasks. We think this is because we were not able to identify any significant relation with sample id and the image. We can see the above results in the below image



2.4 Task 4

Locality Sensitive Hashing –

In this task, we are implementing the Locality Sensitive Hashing(LSH) functionality to find the approximate nearest neighbors of the given query object.

LSH is a way to design a high efficient search function in which, similar to VA files a data space is divided into grids, each section of the grid can be treated as a hash bucket. The basic idea is that a hash function is used to divide the space into smaller chunks and when a query is done on this newly formed space we can search only these smaller chunks instead of the entire data space.

In LSH, the hash function depends on what kind of distance is used to calculate the distance and similarity between the objects. In the case of L2 distance, random vectors which are at different angles are used to create hash buckets. Once a random vector is created then incoming data points are projected onto this vector and depending on the length of the projection, the appropriate window is then selected for that data point. In this task, we took dot products of the data and random vector to decide the window for the data. Contrary to the normal hash function where the aim is to minimize the collisions in this case we want to maximize the collisions for similar objects and different hash buckets for different objects. So that when a query comes, we can return all the objects present in the buckets as a similar object to the user.

If the only hash function is used then it will not give results as results will have a lot of false positives. To solve this issue family of the hash function is created. The family of a hash function is the set of hash functions which have a similar efficiency curve but has one independent parameter from each other; in this case it is the angle to produce the randomized buckets. K such hash functions are used to create a hash family. Users can change the value of k. so the final bucket will be calculated as h_1^ h_2^ h_3^ h_4^ h_5^ ……..h_k. But the drawback of doing this is this will introduce a higher miss rate meaning lot of similar results will not be returned.

To solve this issue multiple layers are created with a similar hash function. In this task, the user can specify L the number of layers. For each layer, K hash functions will be created and buckets will be created accordingly. The union of the results from all the layers is then taken to minimize the miss rate. This way it will be able to minimize the missed and false positives.

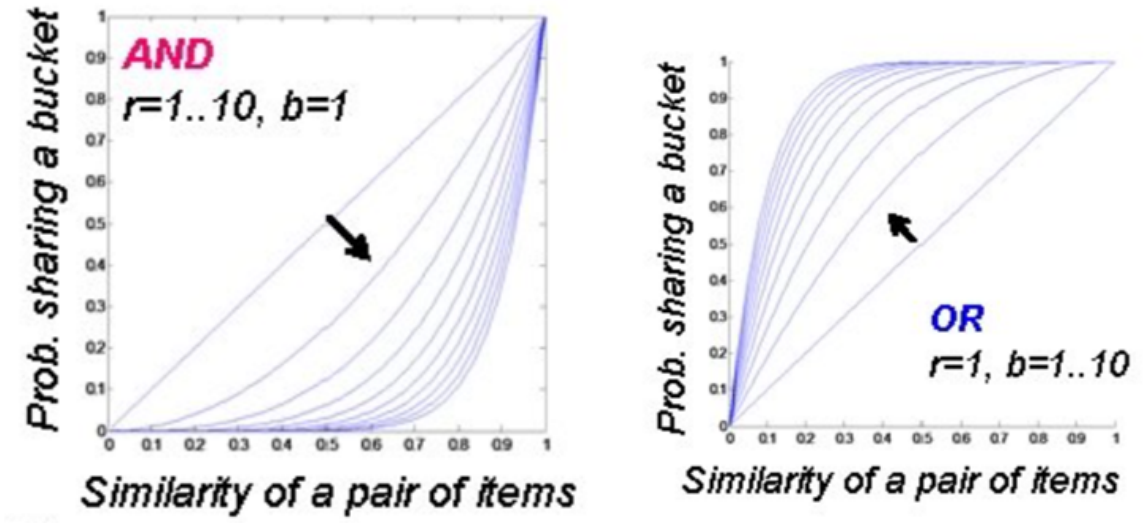


Figure 6. Shows the effect of increasing the value of hash function and number of layers []

The figure shows two graphs with the effect of increasing the number of hash function k (r in the shown fig.) and the number of layers L (b in the fig.). increasing the value of k reduces the false-positive images while increasing the value of L will result in a decrease in the misses in the data.

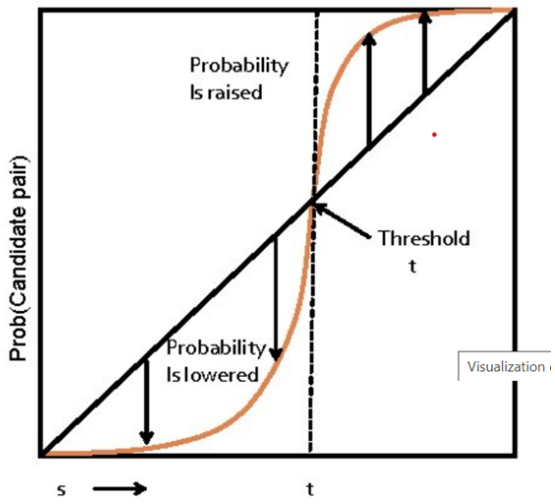


Figure 7.- Overall efficiency curve. []

The figure shows the combined effect of using k and l. the straight line shows the original efficiency curve while the orange ‘s ’ curve shows the updated efficiency curve.

Steps to calculate the LSH index and query the data from the index -

1. User will provide the value of the number of hash functions per layer k, Total number of layers l, the feature model for images, and number of nearest neighbor t.
2. Create a random vector as a hash function and find the dot product of each data point with the random vector.
3. If the dot product is positive, then assign the value as 1 else assign value 0 for that hash function. Perform this for all k random vectors.
4. End of step 3 we will have k bits for the data point, concatenate those k bits to create the hash bucket.
5. Create a map of buckets and club all the images with the same bucket id together.
6. Repeat step 2 to step 5 for each layer.
7. For a given query image, take the random vector generated in step 2 and find the corresponding hash bucket. Find all the images in that bucket.
8. Repeat step 7 for all the Layers and take the union of results of each layer.
9. If the result of step 8 provides more than 't' results then find the t Nearest neighbors and return the result.
10. if the result of step 8 produces fewer than 't' results then look into the neighboring buckets to find more results. Hamming distance is used to find the closest bucket to the given bucket with a max threshold value of 2.
11. If step 10 does not produce 't' results then we return whatever results we have found till this point.

If we do not get 't' images from step 11 then we take the following steps to find more results.

Steps to include more results -

1. Increase the size of the window so that it will have more images in the same bucket, drawback of this is it can lead to a huge number of false-positive results.
2. Increase the threshold of the hamming distance, meaning look into more neighboring buckets to find the results.
3. Increasing the number of layers will give us more images as it takes a union of images to find the images. This will take a significant amount of memory to store the index structure.
4. Reduce the number of hash functions in each layer, This is better than 3rd point as it does not use any additional memory.

Number of buckets and memory requirements for LSH –

Given,

d – Number of data points.

k – Hash function in each layer.

l – Number of layers

For each hash function, 2 windows can be created. So the maximum number of buckets possible for each layer are –

Minimum(2^k, d)

If d buckets are created it means that each data point is stored into a different bucket which is not good for the given use case. It is not possible to pre-calculate the exact number of buckets as a bucket is created only if we have at least one image in that bucket.

Memory required to store the index structure for each layer–

(no. of buckets)*(memory required to store the id of each bucket)

+

(d*memory required to store the identifier of each image.)

Where –

Memory required to store the id of each bucket will depend on the total number of hash functions in each layer.

The memory required to store the image identifier will depend on the number of images in the database.

Memory required to store entire index structure –

L * (Memory required to store index structure of each layer.)

In the code interface the size of the index structure also includes the size required by random vectors as the vectors are required to search the index structure.

Results -

In this task we are able to implement the LSH index structure successfully and able to retrieve the most similar t images to the query image. It is observed during the experiment that the miss rate and false positive rate highly depends on the value of k (hash function per layer) and l (number of layers).

If the value of k is increased then it results in decreased false positives but at the same time it will increase the number of misses as well. On the other hand if the value of l is increased it will improve the number of misses but the false positives will also increase significantly. So we need to find the right balance between k and l.

Result for the values -

K - 10

L - 5

Vector - phase3-task2-PCA-Color_Moments-10

Image folder path - C:/masters/ASU/Courses/mwdb/phase2/Multimedia-and-Web-Databases/Phase3/1000

Feature model - Color_Moments

Reduction - No

Query name - image-cc-3-3.png

t - 20

```
[?] What task do you need?: Task4: Implement a LSH tool, use LSH tool to find nearby t images.
   Task1: Associate label X to a folder of images.
   Task2: Associate label Y to a folder of images.
   Task3: Associate label Z to a folder of images.
 > Task4: Implement a LSH tool, use LSH tool to find nearby t images.
   Task5: Implement a VA-file index tool, use the tool to find near by images.
   Task6: Implement a decision tree based relevance feedback system
   Task7: Implement a SVM classifier based relevance feedback system
   exit
   change data path
   change output path

Enter Number of hash functions K in each Layer : 10
Enter number of layers l : 5
Enter the vector file name generted by other task : phase3-task2-PCA-Color_Moments-10
Enter the images folder path : C:/masters/ASU/Courses/mwdb/phase2/Multimedia-and-Web-Databases/Phase3/1000
[?] What feature model do you need?: Color_Moments
 > Color_Moments
   HOG
   ELBP

[?] Do you want to use reduction method? : No
   Yes
 > No

Input the query image path: C:\masters\ASU\Courses\mwdb\phase2\Multimedia-and-Web-Databases\Phase3\1000\image-cc-3-3.png
number of nearest neighbors want to find t :20
folder found, Extracting features for number of images 888
Saving file C:/masters/ASU/Courses/mwdb/phase2/Multimedia-and-Web-Databases/Phase3/outputs\phase 3 task 4 Color_Moments.npy
hamming distance used 0
overall images considered :  1783
unique images considered :  765
False positive rate is :  97.38562091503267
Miss rate is :  0.0
sized of index structure 83800
image id  image-cc-13-2.png
image id  image-cc-15-1.png
image id  image-cc-21-9.png
image id  image-cc-22-9.png
image id  image-cc-23-4.png
image id  image-cc-23-6.png
```

Detailed results for various test cases are provided in the separate file.

False Positive Rate- 97.38

Miss Rate - 0.0

Size of index structure - 83800 (This includes the random vector size as well)

2.5 Task 5

Implement a VA-file index tool and associated nearest neighbor search operations. The data structures and relevant algorithms are described in the following two papers:

- Stephen Blott and Roger Weber, “A Simple Vector-Approximation File for Similarity Search in HighDimensionalVectorSpaces”. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.9708, and
- Roger Weber, Hans-J¨org Schek, and Stephen Blott. 1998. “A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces". In Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB ’98), pp. 194-205. 1998.

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.

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.

2.5.1 Index Structures

Index file structures are very efficient for low dimensional representation of multimedia objects. But as dimensionality increases we notice a drastic reduction in the performance of index files. This is termed as dimensionality curse. In this task we implement an index structure called ‘VA-Files’. This index structure avoids dimensionality-curse by not partitioning the space based on data but based on a simple filter based approach.[10] VA-Files contains smaller approximations of the objects. These approximations are binary strings, whose bits are divided among the many dimensions that are used to represent the object in the space. Each dimension will have a set of bits which will be used to identify the regions in that dimension an object is present in. The concatenation of all these binary representations for every dimension of the object will represent the object altogether in the VA Files index structure.[9]

2.5.2 VA-File Structure

As we represent the objects as a list of binary strings we first need to determine the number of bits that we will allocate to every dimension. The paper mentions the following formula to allocate the bits:

$$b_j = \left\lfloor \frac{b}{d} \right\rfloor + \begin{cases} 1 & j \leq b \bmod d \\ 0 & otherwise \end{cases}$$

Figure 8.

Once we have this we can use the allocated bits for every dimension to determine the partition points for each of the dimensions. Picking the partition points is important as that will help us to prune the object search space. We must choose the partition space such that the values are split into equally full regions.[9] The approach I have taken to partition the space is explained in the section for implementation details. Once we have the approximations we can use it to calculate the lower bounds and upper bounds to the query object.

Once we have the data-structure of VA-File to index our search space we will focus on the algorithm to be able to effectively search for the top objects that are similar to our query object. The paper mentions two algorithms for searching. First is the simple-search algorithm. In this algorithm, we perform a linear scan with all the approximations of all the objects in the feature space. For each of these approximations we find the lower bounds. If the lower bound we find is less than the max distance of a considered object then we consider the new object as a candidate and compute its true distance, If this distance is indeed less than the max distance of the considered object so far we will replace the least preferred object among those found so far with this object. This way by the end of the iteration on all approximations we will have the top k objects that we need to find.

The Near Optimal algorithm works similarly, it computes the lower bound and upper bound for all approximations with the query image and if the lower bound is less than the max true distance found so far, we will insert the lower bound into the heap. Meanwhile we update the distance value to the upper bound value that we find from the approximation. Once this phase is over we start the second phase on this algorithm. In this second phase, we pop from the heap the lower bound values that we had stored in the first phase, this happens in increasing order of lower bounds. This phase ends when we encounter a lower bound that is greater than or equal to the nth value in the answer set.

2.5.3 Implementation

The first stage of the implementation is to take as inputs the directory of all the images in the dataset and convert this to the appropriate model which is Color Moments, HOG, or ELBP. We will also have to allocate the bits to all the dimensions of the feature space. We do this by using the formula stated in figure 8.

In order for us to represent our VA-Files data structure we use two dictionaries, the first dictionary is named as approximationDictionary. This will store the approximations of every image that exists in the feature space. The second dictionary is called partitionPoints and this dictionary will store the partition points in every dimension.

In order to find partition points for a dimension we must partition the space such that a dimension will have equally full regions. I have implemented this the following way. For every dimension I have sorted the values of all images for that dimension. Next I divide the total points in this dimension by the partitions that are needed in this dimension. This value gives me the intervals at which I need to have a partition. Next, I iterate over the sorted values at each interval, for every point in this interval I find the mean value between interval and interval + 1 and use the mean value as my partition point. I append this onto a list and create the partitionPoints dictionary from this. Once I have the partition points I can find the region in which an image lies for every dimension. The concatenation of the binary values for each of these dimensions gives me the approximate for an image. These values are stored in the approximation dictionary.

To find the approximation we use partitionPoints dictionary. The method findBucket will iterate over the values present in the partitionPoints dictionary and when the value is greater than a partition point and less than the next partition point it returns the bucket value. This will help create our approximationDictionary. Now we have the index structure ready.

When we get our query image we need to implement an algorithm that will make use of our created VA File index structure to effectively query the ‘t’ most similar images. We have two search algorithms to choose from in the paper, one VA-SSA and the other is VA-NOA. I have chosen to use VA-SSA for the project.

To the VA-SSA method we will pass the approximationDictionary, the partitionPoints dictionary, the target image which is converted into the same model that was used to store the database images and the Database Image vectors to be able to compute the true distance. In the algorithm we store two more lists, one is the answer list, which stores the current candidates whose true distances have been computed, and second is the distances between the query image and the images in the answer list. We initialize the distance vectors to infinity. Next, we start the sequential search of all the approximations and find its lower bounds. To find the lower bound we first find the region in which the approximation lies and then we use the formula stated in figure 8.

$$
\begin{aligned}
l_i &= \left(\sum_{j=1}^{d} {l_{i,j}}^p\right)^{\frac{1}{p}} \quad \text{where } l_{i,j} = \begin{cases} v_{q,j} - p_j[r_{i,j}+1] & r_{i,j} < r_{q,j} \\ 0 & r_{i,j} = r_{q,j} \\ p_j[r_{i,j}] - v_{q,j} & r_{i,j} > r_{q,j} \end{cases} \\
u_i &= \left(\sum_{j=1}^{d} {u_{i,j}}^p\right)^{\frac{1}{p}} \text{ where } u_{i,j} = \begin{cases} v_{q,j} - p_j[r_{i,j}] & r_{i,j} < r_{q,j} \\ \max(v_{q,j} - p_j[r_{i,j}],\ p_j[r_{i,j}+1] - v_{q,j}) & r_{i,j} = r_{q,j} \\ p_j[r_{i,j}+1] - v_{q,j} & r_{i,j} > r_{q,j} \end{cases}
\end{aligned} \tag{5}
$$

If the lower bound of the ‘ith’ image that we find for an approximation is smaller than the max distance found so far we find the true distance between the query image and the ‘ith’ image. We then call the Candidate function with parameters, true distance, dst which is the list of distances of the current candidates and ans which is the list of the current candidates.

In the candidate function if the distance computed between the query image and the new image is smaller than the largest distance found so far, we replace the last candidate in the dst list which is the farthest image from the query image and replace it with our new image. We then sort the list by the dst vector.

2.5.4 Observations and Results

The parameters that would depend on how well the index performs is the number of bits taken to represent the images in every dimension, and the decision taken on how to partition the spaces.

When we take bits that are nearly equal to the dimensions of the vector space we only end up using few bits for every dimension. Due to this a large number of images end up in the same bucket. This can make pruning more difficult. I have observed that the number of buckets searched when the number of bits are less would be lesser, than the number of unique images searched. This shows that findDistance method is called a lot more times when using lesser bits and thus the number of unique images considered is more. And also that multiple images exist in the same bucket. This also worsens the performance of miss rates. Whereas increasing the number of bits

For Color moments when we take number of bits as around 1000, we are getting good results. Where the miss rate is 0.0. Similarly for HOG we see very good performance when the number of bits used is greater than 16 thousand bits. The number of unique images considered was less than 5% of the total search space and even number of buckets searched was significantly lesser. These results were observed when we used euclidean distance as our distance measurement in findCandidate function and real distance calculation. When I used other distance measures such as Mean squared Error or hamming distance I found that the miss rate was around 0.2 to 0.4. Euclidean distance performs clearly better.

2.6 Task 6

Decision-tree-based relevance feedback: Implement a decision tree based relevance feedback system to im- prove 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.

2.6.1 Evaluation

As stated in section 2.1.2, A decision tree classifier is a binary tree where a prediction is made by traversing the tree and at each node a decision is made based upon a feature. The decision is made based on a threshold, if the feature is less than the threshold we go left and right otherwise. Constructing a decision tree is about deciding on the attributes that return the highest information gain(i.e. the most homogenous branch). Gini impurity is used for selecting which attribute to select for splitting.

After getting user feedback as relevant or non relevant, we train our decision tree with these labels. We get the k similar images, and predict the labels for the ones not yet labeled. After getting the predicted labels, we adjust the similarity scores accordingly and output the new ranked results by reordering the previous results.

2.6.2 Implementation

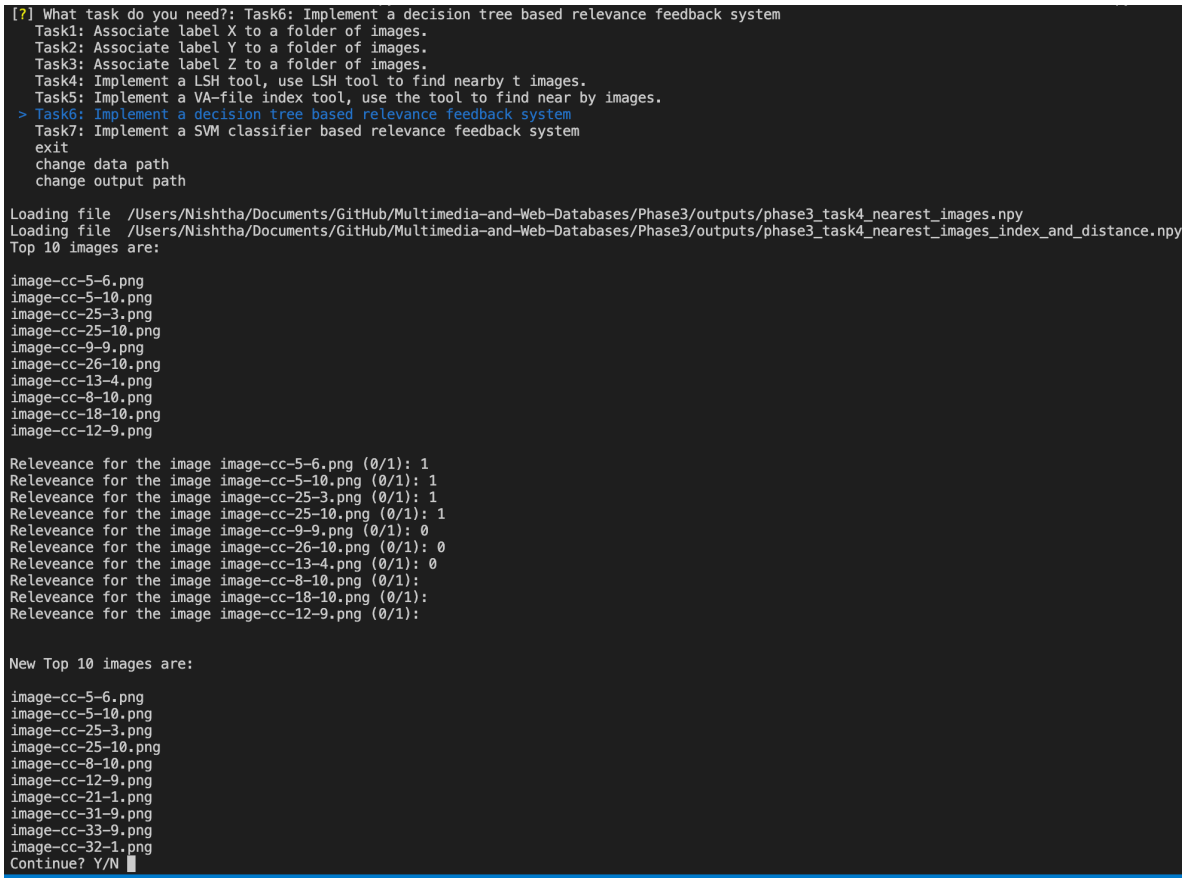
We get the results of the k nearest neighbor from Task 4 or Task 5,

The detailed algorithm used is as follows:

1. The features for each image in the image database are extracted, and stored as features descriptors.
2. Indexing of the images stored are done using either LSH or VA Files.
3. Nearest Neighbor is determined by using the Euclidean distance, the similarity between the features of the query image and the features of all images in the database is computed.
4. All images in the database are sorted based on a similarity.
5. The N images in the database with the highest similarity values are chosen and returned to the user.
6. The user is presented with the images and they label some of them as Relevant/Irrelevant.
7. Decision tree is constructed using the labelled images. The decision tree constructed is used to assign value to the unlabeled images.
8. The labels are used to get the new similarity scores for the images. New ranked images are presented with the images.
9. If the user is satisfied with the images shown, then the process is stopped. Else the algorithm is repeated.

2.6.2 Result

The nearest neighbour here was taken from task 4, and marked the relevant and irrelevant images. Top k nearest neighbour generated by using LSH as indexing structure are used as input file. The nearest neighbours are shown and then the relevance for the images are taken. Then the decision tree is built and the new ranked results are made by the label predicted by the decision tree. The prompt asks to continue if needed further classification.



2.7 Task 7

2.7.1 Support Vector Machine Classifier :

Support vector machine (SVM) is a universal classification algorithm proposed by Vapnik which uses the statistical learning theory. It is a non-probabilistic binary linear classifier which marks the data into two categories based on the training data. The goal of SVM is to maximize the margin of the separator of the two categories. Once the space has been defined new points can be categories into it to assign them their categories.
Even though SVM performs linear classification, it can be used to perform non-linear classification as well using kernel trick, which is implicitly mapping their inputs into high-dimensional feature spaces. Cases when the data is unlabelled an unsupervised learning approach the support-vector clustering algorithm, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications.[1]
To classify the data using feedback provided by the user, svm was implemented with a hard margin with an aim to have maximum margin. This was done using the notation and steps provided by Tristan Fletcher[2] .

In order to use an SVM to solve a linearly separable, binary classification problem we need to:

- Create $\mathbf{H}$, where $H_{ij} = y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$.
- Find $\boldsymbol{\alpha}$ so that

$$\sum_{i=1}^{L} \alpha_i - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha}$$

  is maximized, subject to the constraints

$$\alpha_i \geq 0 \quad \forall_i \text{ and } \sum_{i=1}^{L} \alpha_i y_i = 0.$$

  This is done using a QP solver.

- Calculate $\mathbf{w} = \sum_{i=1}^{L} \alpha_i y_i \mathbf{x}_i$.
- Determine the set of Support Vectors S by finding the indices such that $\alpha_i > 0$.
- Calculate $b = \frac{1}{N_s} \sum_{s \in S} (y_s - \sum_{m \in S} \alpha_m y_m \mathbf{x}_m \cdot \mathbf{x}_s)$.
- Each new point $\mathbf{x}'$ is classified by evaluating $y' = sgn(\mathbf{w} \cdot \mathbf{x}' + b)$.

Thus all the unlabelled images are labelled using the equation mentioned above. Depending on the data distribution, the kernel trick used to classify the data can vary. The method mentioned above provides the flexibility to use various types of kernel tricks to classify the data.

In order to use an SVM to solve a classification on data that is not linearly separable, the first step is to choose a kernel and its parameters so that the non-linearly separable data can be mapped into a feature space where it is linearly separable. Sensible kernels to start with are the Radial Basis, Polynomial and Sigmoidal kernels.
For classification, we would then need to[shivi2] :

- Create $\mathbf{H}$, where $H_{ij} = y_i y_j \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$.
- Choose how significantly misclassifications should be treated, by selecting a suitable value for the parameter C.
- Find $\boldsymbol{\alpha}$ so that

$$\sum_{i=1}^{L} \alpha_i - \frac{1}{2}\boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha}$$

  is maximized, subject to the constraints

$$0 \leq \alpha_i \leq \ C \ \ \forall_i \text{ and } \sum_{i=1}^{L} \alpha_i y_i = 0.$$

  This is done using a QP solver.

- Calculate $\mathbf{w} = \sum_{i=1}^{L} \alpha_i y_i \phi(\mathbf{x}_i)$.
- Determine the set of Support Vectors S by finding the indices such that $0 < \alpha_i \leq C$.
- Calculate $b = \frac{1}{N_s} \sum_{s \in S} (y_s - \sum_{m \in S} \alpha_m y_m \phi(\mathbf{x}_m) \cdot \phi(\mathbf{x}_s))$.
- Each new point $\mathbf{x}'$ is classified by evaluating $y' = sgn(\mathbf{w} \cdot \phi(\mathbf{x}') + b)$.

2.7.2 SVM Active Learning

Unlike supervised learning where all the data is labelled before hand, and the only work during classification is to determine the support vectors using the training data and testing the accuracy of the method using the testing data, for scenarios where the data is unlabelled and the labelling is done

iteratively using the user's feedback it is necessary to determine the images which would be used to get the user feedback such that they will have maximum impact on the feedback process.

Now, since the user labels only a small subset of unlabelled images, it is necessary to determine which subset will be most valuable among the unlabelled data. Thus, associating a cost function with the unlabelled images is required. This cost associated with each unlabelled image is determined in terms of the distance between the image in the feature space and the support vectors. According to [3] when the nearest images to the relevant images are chosen for feedback there is less shift in the separator between the two classes. Thus, it was decided to calculate the distance with respect to the support vectors.

2.7.4 Image Retrieval System

For image retrieval using user feedback it is important to choose most semantically relevant images after each previous relevance feedback iteration. It is also important to understand that not many images will be labelled by the user. Since each feedback can be useful to determine the label of the unlabelled images and since each iteration will have an impact in determining the most semantically relevant images, thus the unlabelled images are reordered every time based on the previously labelled images. Two options are generally used when query image, result and relevance feedback is available. One is query rewriting to discover missed relevant images, while the other is to reorder the results. Since the first option requires processing the whole query it is very expensive. Thus, we decided to go forward with the second option as the results could be improved based on the feedback provided by the user. One of the disadvantages of the approach selected is that the images that were missed due to the query can not be retrieved back. Following steps were followed to improve the nearest neighbor images generated based on LSH or VA Files:

1. The low-level features for each image in the image database are extracted, and stored as features descriptors. Feature reduction is applied on the feature descriptors based on the user preference.
10. Indexing of the images stored are done using either LSH or VA Files.
11. Nearest Neighbor is determined by using the Euclidean distance, the similarity between the features of the query image and the features of all images in the database is computed.
12. All images in the database are sorted based on a similarity.
13. The N images in the database with the highest similarity values are chosen and returned to the user.
14. Every time a user labels an image it is added to the labelled image database and all the other unlabelled images's label is predicted based on them.
15. SVM classifiers are constructed using the labelled and unlabelled images.
16. If the user is satisfied with the images shown, then the process is stopped.
17. Else, the most relevant images closest to the separator are determined to get valuable feedback from the user and repeat the process from 3 to 8.

2.7.4 Result

After considering various types of kernel tricks for classifying the data based on the relevance feedback it was observed that the polynomial kernel trick was most effective at classifying the images. It was also observed that the effectiveness of the relevance feedback also depends upon the underlying indexing structure. According to the various design decisions taken for indexing using LSH, it was found that the user gets relevant images in very few iterations. Also, since we are just reordering the images retrieved while improving the result for nearest neighbor thus, images missed due to the underlying indexing structure can not be retrieved again. But since, it is quite computational intensive to rewrite the query image and to process the nearest neighbor again and again based on the user feedback it was found the this approach holds a better place when the image database is large.

2.8 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 the user to provide positive and/or negative feedback for the ranked results returned by the system.

The entire phase3 is integrated as shown in the interface specifications section.

3. Interface Specifications

The interface looks like the following when you run the *driver.py* file in *code/src*. Use the up and down arrows to choose between the available options. →

```
[?] What task do you need?: Task1: Associate label X to a folder of images.
 > Task1: Associate label X to a folder of images.
   Task2: Associate label Y to a folder of images.
   Task3: Associate label Z to a folder of images.
   Task4: Implement a LSH tool, use LSH tool to find nearby t images.
   Task5: Implement a VA-file index tool, use the tool to find near by
images.
   Task6: Implement a decision tree based relevance feedback system
   Task7: Implement a SVM classifier based relevance feedback system
   exit
   change data path
   change output path
```

```
Input the image folder for latent semantics:
/Users/khushalmodi/Desktop/ASU/Fall_2021/MWDB/Phase-3/Multimedia-and-Web-Da
tabases/Phase3/4000
Input the value of k: 15
[?] What feature model do you need?: ELBP
   Color_Moments
   HOG
 > ELBP
```

```
[?] Choose a classifier: Decision Tree
   SVM
 > Decision Tree
   PPR
```

The following file structure is followed and is to be maintained when storing the latent semantics →

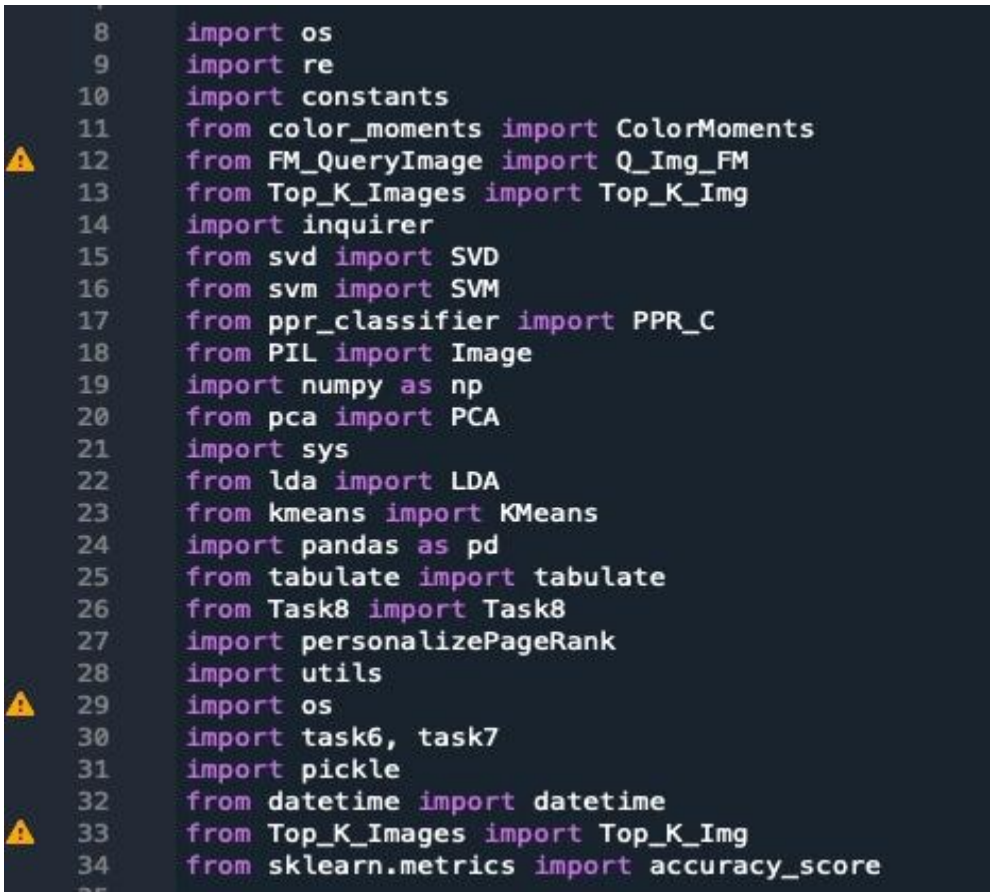
- Each train folder is given a timestamp when it is first entered into the system. A directory with that timestamp is used to store all files related to all the code runs made on that folder.
- All the outputs are stored in the similar directory as well.
- User needs to update the dict file path and output path in constants.py file before starting the execution.

We have the following classes created for the above implementation →

```
Class TaskDriverPhase3: Drives the execution of the entire project
Class SVM: SVM classifier implemented as a part of the project
Class DecisionTreeClassifier: Decision Tree implemented as part of the
project
Class PPR_C: PPR classifier implemented as part of the project
Class LSH: LSH algorithm implemented as a part of the project in phase 3
Class Task7_feedback: Contains implementation of task 7 to run user
relevant feedback mechanism
```

4. System Requirements/ Execution instructions

1. The code is in the *code/src* folder.
2. Requires python3 setup to run the code.
3. Run the *driver.py* file in the terminal to start the execution.
4. Please change the output paths before executing any tasks if tasks are being executed on a different machine.
5. Follow the instructions in the console to run the program properly
6. Do not enter an invalid input otherwise the program will crash.
7. Need following python packages installed → sklearn, scipy, matplotlib, numPy, DateTime, sys, os, PIL, skimage, pandas, tabulate, inquirer, tkinter,
8. Please try to run the code in a terminal because the inquirer UI works better in a terminal than in any IDE shell.



Packages needed

5. Related Work

[11] Discusses how VA Files index structures were compared on high dimensional feature vectors and compared with the performance of R* index structures on these same feature spaces. The study didn't lead to a conclusive winner on which performance is better.

[3] discusses the use of svm classification for active learning where it shows how the approach used to classify unlabelled images based on the feedback provided by the user interface can be used improved by using svm ensembles such that each classifier is made for each feature subset. Active learning is close to supervised learning, except that training data are not independent and identically distributed variables. It also gives different weightage to different classifiers based on the accuracy of each classifier. The approach also discusses the use of lagrange multiplier and active learning methods to derive the support vectors and to choose the most suitable data points to the user. The paper also points out how rewriting the query is an expensive approach compared to reordering the result dataset. The experiment result shows how the proposed approach is better than the other state-of-the-art approaches. In this paper, it is proposed that the use of SVM-based active feedback using ensemble multiple classifiers. The paper shows that for each iteration, most informative images by using active learning methods are chosen for the user to label, so that boundary with maximum separation could be learnt after each iteration. Then a set of classifiers based on different sub-feature subsets are trained separately so that error in one classifier can be avoided in others. Lastly, the weight vector of component SVM classifiers is computed dynamically by using the parameters for positive and negative samples. This overall approach helps to improve content based image retrieval based on user feedback.

6. Conclusion

In this phase of the project we experimented with the different types of the classifier to classify the image data of 40 subjects and for each subject 13 types of transformations are done, for each of the transformed image images are oriented slightly changed to create the different images. In Task 1 we applied different classification techniques on the type of the images. Similarly for task 2 we classified the data according to the subject id of the images and in task 3 we classified the images with respect to orientation of the data. For task 4 we have implemented the LSH index and implemented an algorithm for the most similar 't' images given a query image. In task 5 we have created a VA files index structure and implemented a logic to retrieve similar top t images given a query image. In task 6 we ask users to mark relevant and irrelevant images and implement a decision tree based classification to further improve the classification results. Task 7 is very similar to task 6 but we are using SVM classifier to re-classify the images.

The design decisions taken for improving the nearest neighbor results based on user's relevance feedback has several advantages and disadvantages. Since the operation was done based on the initial query image and the underlying indexing technique, images which were missed due to the preprocess couldn't be retrieved. Also, since the cost of computing the indexes and the nearest neighbor based on the rewritten query image it was decided to improve results only based on the reordering of the images. The output of the SVM based relevance feedback also depends upon the selection of kernel tricks based on the data distribution and the selection of various parameters. Many relevant images could be missed if the selection is not done correctly.

In task 4 we were successfully able to create the index structure and retrieve the relevant images from the index structure. It is observed that the false positive rate is inversely proportional to the number of hash functions per layer. The miss rate is also inversely proportional to the number of layers in the LSH. More the number of layers less the miss rate but it will increase the false positives rates. It is generally observed that increasing the number of hash functions per layer does not have significant increase in the memory requirement but an increase in the number of layers increases the memory of index structure significantly.

One of the main things we notice in our implementation is that the accuracy of classifiers is pretty low. Upon analyzing the phase2 outputs and comparing them with the phase3 outputs, we found out that in phase2, we were building our latent semantics based on a specific X or Y value. But in phase3, we are basically deriving latent semantics from all the images. We came to a conclusion that deriving latent semantics based on all the images in the database produces results which are evenly distributed. Hence, we are not getting accuracy in our classifiers. Because there is no bias in the classifier towards any one type, subject or orientation, the classifiers are not able to capture the exact type subject or orientation of the given query image.

7. Bibliography

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

[2] Support Vector Machines Explained Tristan Fletcher www.cs.ucl.ac.uk/staff/T.Fletcher/ March 1, 2009

[3] A new SVM-based active feedback scheme for image retrieval Xiang-Yang Wang, Hong-Ying Yang, Yong-Wei Li, Wei-Yi Li, Jing-Wei Chen, 19 September 2014

[4] https://www.mathworks.com/help/stats/support-vector-machines-for-binary-classification.html

[5] https://www.analyticsvidhya.com/blog/2021/05/multiclass-classification-using-svm/

[6] https://slidetodoc.com/theory-of-locality-sensitive-hashing-recap-finding-similar/

[7] https://towardsdatascience.com/decision-tree-algorithm-in-python-from-scratch-8c43f0e40173

[8]https://stackoverflow.com/questions/50666091/true-positive-rate-and-false-positive-rate-tpr-fpr-for-multi-class-data-in-py

[9] Stephen Blott and Roger Weber, "A Simple Vector-Approximation File for Similarity Search in HighDimensionalVectorSpaces". http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.9708,

[10] Roger Weber, Hans-J¨org Schek, and Stephen Blott. 1998. "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces". In Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB '98), pp. 194-205. 1998.

[11] https://link.springer.com/chapter/10.1007/11547686_12

[12] https://towardsdatascience.com/decision-tree-from-scratch-in-python-46e99dfea775

Role of Group Members

All the design decisions were made after a discussion with all the team members. The tasks were divided based on the availability and preferences of each member. All the members contributed to the report based on the things they worked on.

Roles	
Glen Dsouza	Task 5
Khushal Modi	Task 1, 2, 3, 8
Manthan Agrawal	Task 4
Nishtha Bhimte	Task 6
Shivani Priya	Task 7
Sowmith Reddy	Task 1, 2, 3, 8

Results and outputs -

We have tested all the tasks with different combinations of feature models, reduction techniques, and types of images. The results for each are included in the separate file outputs folder of the submission.