CSE 515: Phase 3

Group 21: Ariana Bui, Maryam Cheema, Peter Kim, Rajat Pawar, Yash Dhamecha

Abstract

This phase delves into the exploration of multimedia and web databases using the Caltec101 dataset in continuation of phase 2. It encompasses tasks focusing on clustering, indexing, and classification/relevance feedback. Leveraging previous phase-developed feature models, similarity/distance functions, and latent space extraction algorithms, the project mandates the implementation of distinct functionalities.

The tasks involve an investigation into inherent dimensionality assessment for even-numbered Caltec101 images and their unique labels. Task 1 requires the computation of k latent semantics associated with each unique label for even-numbered images. Subsequently, the odd-numbered images undergo label prediction using computed distances/similarities under label-specific latent semantics, yielding precision, recall, F1-score, and overall accuracy values. Task 2 involves the creation of significant clusters associated with even-numbered Caltec101 images, visualized in 2D MDS space and as image thumbnails. Predictions for odd-numbered images are made using label-specific clusters, generating evaluation metrics similar to Task 1. Task 3 involves implementing m-NN, Decision Tree, and PPR classifiers using user-specified features, predicting labels for odd-numbered images, and providing evaluation metrics. Furthermore, Task 4 comprises the implementation of a Locality Sensitive Hashing (LSH) tool and a similar image search algorithm. The LSH tool generates an in-memory index structure for approximate nearest neighbor searches based on Euclidean distance, while the search algorithm identifies similar images using the constructed index and a chosen visual model. Lastly, Task 5 implements SVM and probabilistic relevance feedback systems. These systems allow users to tag retrieved results, enhancing retrieval performance by refining queries or reordering existing results based on user feedback. Deliverables include properly commented code, outputs for provided sample inputs, and a concise report detailing the undertaken work and results. The submission follows a specific directory structure, ensuring an organized submission process.

Introduction

In continuation of our exploration into multimedia and web databases, Phase #3 of the CSE 515 project delves deeper into clustering, indexing, classification, and relevance feedback methodologies. Continuing our utilization of the Caltec101 dataset, this phase focuses on the refinement and implementation of feature models, similarity/distance functions, and latent space extraction algorithms developed in the previous phase.

A series of diverse tasks define this phase's objectives. From assessing inherent dimensionality associated with Caltec101 images and unique labels to predicting labels for odd-numbered images using latent semantics, the tasks span across varied functionalities. Visualizing significant clusters in 2D MDS space, predicting labels based on these clusters, and employing classifiers such as m-NN, Decision Tree, and PPR for predicting odd-numbered images align with the project's objectives.

Furthermore, the implementation of a Locality Sensitive Hashing (LSH) tool, an image search algorithm using LSH, and the development of relevance feedback systems using SVM and probabilistic techniques are key tasks driving this phase.

The project's deliverables encompass properly commented code, outputs for sample inputs, and a succinct report detailing the undertaken work and results. Adhering to specific directory structures ensures a systematic submission process, providing insights into the evolution and enhancements of the applied methodologies from the preceding phases.

Description of Proposed Solution/Implementation

1. Task 1

- Problem Statement:
    - The primary goal of this task is to evaluate the precision of predicted labels of odd images in the Caltech dataset based on the similarity score with the even numbered images in the Caltech dataset.
- Input
    - The value of k which is the number of latent features the user wants to associate with the even and odd numbered images.
    - The feature model: The user can select the feature model that they want to use for the image features.
- Output
    - Outputs the precision, recall, f-1 score and the overall accuracy of the system.
- Implementation:
    - Feature extraction and storage: In this project the extracted features are being stored in a CSV file. The features are then parsed and stored in a variable as rows of arrays of floating point values.
    - Dimensionality reduction: For each unique label, the features of the images in each label are reduced by the dimensionality reduction of Singular Value Decomposition (SVD). SVD was used because after testing, it gave the highest accuracy among the dimensionality reduction techniques. The dimensionality reduction was done for both the even and odd datasets. The LS conversion was applied to each of the labels so we can find label specific patterns even after the dimensionality reduction. If SVD was applied to the entire dataset, it would give us more global semantic relationships which are not as helpful when predicting labels. The labels were also not evenly distributed, this means that some labels had more images while the other labels had fewer images. If SVD would have been applied as a whole, it would have resulted in latent features that were more representative of labels with larger numbers of images.
    - Label specific similarity measure: For each odd image, the similarity with every even image was computed and stored, the labels of the five images that had the highest similarity with the specific odd image were stored. The label that had the highest occurrence in the stored labels was then predicted as the likely label. If only one instance of label was present, the label with the highest similarity was selected. The reason for choosing the highest occurrence was to prevent outliers in label prediction. It could be that for example an image in label 100 might have the highest similarity with an image in label 1. However by choosing 5 highest similarity images, it would be unlikely that it would score high similarity with other images in the label. We chose five as the number after trial and error. Numbers greater than five also reduced accuracy, this is most likely because some labels had more images, hence resulting in matching with the odd labeled image. For example, the images in label 0 showed high similarity with a lot of odd label images, this resulted in inaccurate prediction of the label.

- The precision, recall, f-1 score and the overall accuracy were then calculated based on the predicted labels and the actual labels for the odd numbered images.

2. Task 2

Problem Statement

The primary goal of this task is to compute the *c* most significant clusters associated with the even numbered Caltec images using the DBScan algorithm. The clusters are then used to predict the most likely labels for the odd numbered images.

Input

- *c*: the number of clusters chosen to be most significant

Output

- The *c* most significant clusters associated with the even numbered images
  - Visualization of these clusters as differently colored point clouds
  - Visualization of these clusters as groups of image thumbnails
- The most likely labels of the odd numbered images
- The precision, recall, and F1-score values and an overall accuracy value

Implementation

Caltech Data
The data was split into two – even and odd images. The even images were used to create the clusters, and the odd images were used for testing.

The RESNET-FC feature data was used to represent the images. The feature data was first normalized using min-max normalization. Both min-max and z-score normalization were tested, but the min-max normalization was preferred.

DBScan
To create the clusters, DBScan was used.

The DBScan has two important parameters - eps and minPts. Eps is the neighborhood around a data point. If the distance between two points is lower or equal to eps, they are considered neighbors. MinPts is the minimum number of points required to form a cluster.

Different combinations of eps and minPts were tested to find a good balance of clusters. In order to achieve better results, eps and minPts had to be tailored depending on the label. To determine eps, the average pairwise distance between all points in the label was calculated then multiplied by a constant. The minPts parameter depends on the number of points in the label. The more points there were, the greater minPts was set to.

Algorithm:
All points begin as unvisited. The algorithm goes through the unvisited points and finds their neighbors. The neighbors were found using the Euclidean distance. Cosine similarity was also tested, but did not yield as good of results as Euclidean. If the distance between two points was less than eps, then they were deemed neighbors. If the number of neighbors of a point was less than the minPts parameter, then the point was marked as noise. If it was greater than every point in the neighborhood was processed. If one of the neighbors was originally noise, it was assigned to the cluster. If it was previously unvisited then the point was assigned to the cluster and its neighbors were found and also added to the neighborhood.

Top Cluster Selection
The top cluster selection is based on cluster size. Once the clusters are formed they are sorted by size, and the largest *c* clusters are chosen. For some of the labels, DBScan did not yield an adequate amount of clusters. As a result, some labels have less than *c* associated clusters.

Dimensionality Reduction for Visualization
In order to visualize the data in a 2D point cloud, dimensionality reduction, specifically classical MDS (multidimensional scaling), was used. This involved creating a dissimilarity matrix, double centering, eigenvalue decomposition, selecting the top k eigenvectors and eigenvalues, and multiplying them to create a 2D result.

Likely Label Prediction
Each cluster has a representative point, the centroid of the cluster (which was determined by averaging all points in the cluster). The decision to use the average as the centroid instead of a middle point was due to the fear that a middle point would not accurately represent the cluster if it had an abnormal shape.

The distance from the image's feature vector to each of the representative points of the *c* significant clusters across all labels is calculated and compared. The label of the nearest cluster representative is assigned as the prediction label.

Precision, Recall, F1, and Accuracy
These metrics are calculated to evaluate the quality of predictions.
The precision, recall, and F1 scores are calculated for each label.
The accuracy is calculated for all predictions.

3. Task 3

Problem Statement

The primary goal of this task is to develop and evaluate algorithms for image classification using extracted features from the ResNet_FC_1000 model. The

implemented algorithms include the m-NN Classifier, Decision Tree Classifier, and Personalized PageRank (PPR) Classifier. These classifiers aim to accurately predict image labels from a given dataset.

Problem Description

Data Description: The dataset consists of image features extracted using the ResNet_FC_1000 model, capturing high-level representations of images.

Classification Task: Given these image features, the task is to predict respective labels or categories associated with the images.

PLEASE NOTE: In this task, we are utilizing even-numbered images for training and odd-numbered ones for testing. I've been granted the flexibility to employ output from any layer, and for this task, I've opted to utilize the output from the ResNet fully connected layer.

In this project, the extracted features are being stored in a pandas DataFrame after being parsed from a CSV file. Specifically, the ResNet features are extracted and stored as columns in the DataFrame. These features, obtained from the ResNet fully connected layer, are represented as arrays of floating-point values.

The vectors are 1000-dimensional, and the dataset comprises approximately 8,678 images, split evenly between training and testing sets, each set containing 4,339 images. Before making predictions with any classifier, the system prompts the user to specify whether they want predictions for a particular image.

A deep dive into different classifiers

m-NN (m-Nearest Neighbors) Classifier

The m-NN (m-Nearest Neighbors) Classifier embodies a simple yet powerful concept in machine learning for classification tasks. It operates on the principle of proximity: given a new data point, it identifies its k nearest neighbors in the feature space based on a predefined distance metric. In this specific implementation, the classifier utilizes the extracted feature vectors from the ResNet_FC_1000 model, which capture intricate visual representations of images. The algorithm, upon receiving the even subset of the dataset for training, fits itself by storing these feature vectors alongside their corresponding labels. Predictions for odd subset images are made by calculating the k nearest neighbors based on feature similarity. The choice of 'm' or the number of neighbors significantly influences the classifier's behavior. A smaller 'm' might lead to increased sensitivity to local variations, potentially introducing noise, while a larger 'm' might oversimplify the decision boundaries, risking over-generalization. Through this approach, the m-NN Classifier in this context aims to accurately categorize images by leveraging the similarity of their feature representations, offering a fundamental yet effective methodology for image classification tasks.

Let's discuss the approach followed to implement this:

The m-NN (m-Nearest Neighbors) classifier is implemented as a part of a broader machine learning framework to classify images based on their feature vectors. This classifier leverages the principle of similarity among data points to make predictions. The approach followed in implementing the m-NN classifier involves several key steps:

1. **Initialization**: The classifier is initialized with the value of 'm', indicating the number of nearest neighbors to consider for classification.

2. **Fit Function**: During the fitting process, the classifier stores the training data—both the feature vectors (images) and their corresponding labels.

3. **Prediction Function**: When predicting labels for new images, the classifier calculates the similarity between the new image and all the images in the training set. This is typically done using distance metrics like Euclidean distance or cosine similarity.

4. **Nearest Neighbor Selection**: The classifier identifies the 'm' nearest neighbors to the new image based on the calculated similarities. These nearest neighbors are determined by sorting and selecting the 'm' data points with the smallest distances or highest similarities.

5. **Majority Voting**: The classifier then employs a majority voting mechanism among these 'm' neighbors to determine the label for the new image. The label most commonly represented among the nearest neighbors is assigned as the predicted label for the new image.

6. **Prediction Output**: Finally, the classifier outputs the predicted label for the new image based on the majority vote among its 'm' nearest neighbors.

The m-NN classifier's simplicity lies in its reliance on neighbor-based inference, where predictions are made based on the labels of the nearest data points in the feature space. This approach is intuitive and effective for many classification tasks, especially when dealing with diverse and complex datasets, as it capitalizes on the local relationships between data points. Adjusting the value of 'm' allows for flexibility in the trade-off between bias and variance in the classifier's predictions.

Decision Tree Classifier

The Decision Tree Classifier is a powerful supervised learning algorithm used for both classification and regression tasks, renowned for its interpretability and ability to handle complex relationships within the data. Operating on a tree-like structure, it recursively partitions the feature space into smaller regions, making decisions based on learned rules derived from the training data. Each node of the tree represents a feature, with

branches denoting possible feature values and leaf nodes indicating the final class prediction. The construction of the tree involves selecting the best features to split the data, aiming to maximize information gain or minimize impurity at each step. This method allows for the creation of intuitive decision rules that are easily interpretable. However, while Decision Trees can efficiently handle both numerical and categorical data and are less sensitive to outliers, they are prone to overfitting, especially when the tree depth or complexity is not appropriately controlled. Techniques like pruning or setting constraints on tree depth mitigate this issue, making Decision Tree Classifier a versatile and widely used algorithm in various domains, including healthcare diagnostics, finance, and recommendation systems.

Let's discuss the approach followed to implement this:

The implementation of the Decision Tree Classifier involves several key steps that collectively construct a tree-like structure to facilitate the classification process. Initially, the classifier starts by evaluating different potential splits in the dataset based on various features. This involves recursively partitioning the dataset into subsets using specific thresholds on selected features. The tree-building process continues until a predefined stopping criterion is met, which could include reaching a maximum depth or having subsets with a minimum number of samples.

The core components of this implementation revolve around:

1. **Splitting Criteria Selection**: It assesses the quality of a split by measuring the homogeneity of classes within subsets. Popular metrics like Gini impurity or Information Gain (measured using entropy) guide the selection of the best split at each node.

2. **Tree Building**: The algorithm progressively constructs the decision tree by identifying the optimal feature and threshold to split the dataset. It examines each feature's potential threshold values to determine the split that maximizes the information gain or minimizes impurity.

3. **Stopping Conditions**: The tree-building process halts when predefined conditions are met, preventing overfitting. These conditions can include reaching the maximum depth of the tree or having nodes with a minimum number of samples.

4. **Leaf Node Assignment**: Once the tree is constructed, leaf nodes represent the final decision points where class labels are assigned. These leaf nodes contain the majority class or utilize some other criterion to determine the final label for a given subset.

5. **Prediction**: During prediction, new instances traverse through the decision nodes based on their feature values until they reach a leaf node, which assigns a class label.

The implementation focuses on efficiently identifying the most discriminative features and thresholds for partitioning the dataset, thereby creating a hierarchical structure that maximizes the information gain at each split. This approach enables the decision tree to learn patterns in the data and make accurate predictions for unseen instances based on the learned tree structure.

The Gini index is a criterion used in decision tree algorithms to evaluate the impurity or homogeneity of a dataset. Here's why it's used and its advantages:

Why Gini Index is Used:

1. Measure of Impurity: The Gini index measures how often a randomly chosen element would be incorrectly classified. It's a measure of impurity or randomness in a dataset.
2. Splitting Criterion: In decision trees, the algorithm chooses the feature and split point that minimizes the Gini index when creating branches. It aims to maximize the purity of the resulting subsets after the split.

Advantages of Gini Index:

1. Simple to Implement: Gini index calculation is relatively simple and efficient compared to other impurity measures like entropy.
2. Computationally Efficient: It requires less computational power because it doesn't involve logarithmic functions like entropy.
3. Favors Larger Classes: Gini index tends to favor larger partitions or classes, making it useful when dealing with imbalanced datasets. It may help in generating trees where major classes are better represented.

Effect on Results:

1. Impact on Tree Structure: The choice of impurity measure, such as Gini index, can affect the resulting decision tree's structure. It might prioritize certain types of splits over others, leading to different branches in the tree.
2. Performance: The Gini index might lead to different predictive performance compared to other impurity measures like entropy. It might perform better or worse depending on the nature of the dataset and the relationships between features and the target variable.
3. Bias towards Features: In some cases, Gini index may show a bias towards features with more levels (more categorical values) or towards features with a different distribution of values.

The decision to use Gini index as the splitting criterion in a decision tree depends on the nature of the data, the problem at hand, and sometimes, empirical testing to see which criterion yields better performance for a specific dataset.

Personalized PageRank (PPR) Classifier

The Personalized PageRank (PPR) Classifier represents an innovative approach to image classification by leveraging the principles of personalized PageRank algorithms within a machine learning framework. This classifier is designed to discern image labels by establishing intricate relationships among images based on their feature vectors. Unlike conventional classifiers, the PPR Classifier operates by constructing a similarity graph derived from image features, quantifying the resemblance between images using cosine similarity. This graph is dynamically updated during prediction, incorporating the queried image into the existing similarity structure. By personalizing the PageRank algorithm with vectors representing each unique image label, the classifier effectively computes personalized importance scores, enabling it to discern label associations and make predictions. Moreover, the PPR Classifier offers a distinctive capability to adaptively adjust its internal graph structure, ensuring continual learning and adaptation to new data. This adaptability, combined with its utilization of personalized importance rankings, empowers the classifier to make informed predictions, demonstrating its efficacy in handling image classification tasks with a high degree of accuracy and flexibility.

Let's discuss the approach followed to implement this:

The implementation of the Personalized PageRank (PPR) Classifier involves several key steps to effectively leverage the personalized PageRank algorithm within a machine learning context.

1. **Graph Construction**: The first step involves constructing a similarity graph based on the feature vectors of the images. This graph represents relationships between images, typically using cosine similarity to quantify the similarity between each pair of images.

2. **Personalization Vectors**: To incorporate labels into the PageRank algorithm, personalized vectors are created for each unique image label. These vectors serve as personalized preferences, indicating the association of each image with its respective label.

3. **PageRank Algorithm Customization**: The core of the PPR Classifier lies in adapting the PageRank algorithm to accommodate personalized vectors. By integrating these vectors into the PageRank computation, the algorithm can calculate personalized importance scores for each image based on its relationship with the labels.

4. **Graph Update during Prediction**: During prediction, when a new image is queried, the existing similarity graph is dynamically updated to include this new image. This

involves recalculating similarities between the new image and existing images and updating the graph structure accordingly.

5. **Prediction Process**: The PPR Classifier uses the personalized importance scores obtained from the PageRank algorithm to make predictions. It selects the label with the highest importance score as the predicted label for the queried image.

6. **Adaptability and Continual Learning**: The classifier's adaptability is a crucial aspect. It continually learns and adapts by updating its internal graph structure with new images and their relationships, ensuring that the classifier evolves with new data to maintain accuracy.

7. **Utilization of Similarity Scores**: Beyond prediction, the similarity scores generated by the algorithm can provide insights into the relationships among different labels and images, aiding in understanding label associations and image clusters.

The approach involves a fusion of graph-based similarity computations, personalized importance rankings, and continual learning mechanisms, enabling the PPR Classifier to make accurate predictions while dynamically adapting to changing data distributions.

4. Task 4

a. Task 4a:

i. Problem Statement:

To implement a Locality Sensitive Hashing (LSH) tool (for Euclidean distance) which takes the following input.

ii. Input
1. The number of layers, L
2. The number of hashes per layer, h, and
3. A set of vectors as input and creates an in-memory index structure containing the given set of vectors

iii. Output
1. Index in Pickle file

iv. Implementation:
1. Feature Selection
    a. We have selected the Resnet50 FC feature vector as it is sufficiently large in dimensions(1024>512) to be used in LSH hashing and similarity search using euclidean distance for Resnet50FC worked pretty accurately in Phase 1.
2. Classes:
    a. Layer Class:

i. Initializes with a hash size and random projections.
ii. The Layer class represents an individual layer in the LSH index.
iii. It is initialized with a specified hash size and random projections (hyperplanes) for hash code generation.
iv. Adds vectors to the layer and implements query methods.
v. Provides methods to add vectors to the layer along with their labels.
vi. Implements query methods to retrieve vectors based on hash codes.

b. LSH Class:

i. Initializes LSH with layers, hashes, and dimension of feature vector.
ii. The LSH class is the main class representing the Locality Sensitive Hashing index.
iii. It is initialized with the number of layers, hashes per layer, and the dimension of the input vectors.
iv. Adds vectors and implements query methods.
v. Utilizes multiple layers to organize and efficiently query vectors.
vi. Manages the addition of vectors to each layer and provides methods for querying.
vii. Describes hash tables for debugging.
viii. Provides a method to print and examine the hash tables for debugging purposes.

3. Hash Functions:
    a. Converts vectors into hash codes based on random projections.
    b. Implements a hash function that projects input vectors onto random hyperplanes, generating hash codes.
4. Index Creation:
    a. Reads feature vectors from the Caltech101 dataset.
    b. Retrieves feature vectors (ResNet FC 1000) from the Caltech101 dataset.
    c. Adds vectors to the LSH index using the LSH class.
    d. Utilizes the LSH index to organize and store the feature vectors efficiently.
    e. Serializes the LSH index and details into a Pickle file.
    f. Saves the LSH index, including layers, hashes, and other details, in a Pickle file for future use.

b. Task 4b:
    i. Problem Statement:

    To implement a similar image search algorithm using this index structure storing the even numbered Caltec101 images and a visual model of your choice (the combined visual model must have at least 256 dimensions): for a given query image and integer t

    ii. Input:
        1. A given query image
        2. Integer t
    iii. Output:
        1. Visualizes the t most similar images
        2. Outputs the numbers of unique
        3. Overall number of images considered during the process
    iv. Implementation:
        1. Nearest Neighbor Search Class:
            a. Initializes with the LSH index, layers, and hashes.
            b. The ApproximateNearestNeighborSearch class is initialized with parameters such as the number of layers and hashes.
            c. It also provides options to load an existing LSH index.
            d. Implements methods to train the LSH index and find t nearest neighbors.
            e. Provides a method to train the LSH index by adding feature vectors.
            f. Implements methods to find t nearest neighbors for a given query.
        2. Image Retrieval:
            a. Loads the LSH index from the Pickle file.
            b. Reads the serialized LSH index from the Pickle file for use in image retrieval.
            c. Preprocesses the query image to obtain its feature vector.
            d. Applies preprocessing to the query image to obtain the feature vector (e.g., ResNet FC 1000).
            e. Uses the LSH index to find t nearest neighbors for the query image.
            f. Utilizes the LSH index to efficiently retrieve similar images based on the query vector.
        3. Visualization:
            a. Plots the t most similar images alongside the query image.
            b. Creates a visual representation displaying the query image and its t most similar neighbors.
            c. Displays Euclidean distances for each retrieved image.

d. Shows the Euclidean distances between the query image and its retrieved neighbors.
e. Saves the visualization as an image file.
f. Stores the visualization output as an image file for later reference.

4. Post-Processing:
    a. Calculates unique and overall image counts.
    b. Determines the count of unique images considered during the retrieval process.
    c. Provides an overall count of images considered during the retrieval process.
    d. Prints the results.
    e. Outputs the results, including the unique and overall image counts.

5. Task 5

Problem Statement:

The goal of this task was to implement two types of relevance feedback systems: an SVM-based relevance feedback system and a probabilistic relevance feedback system. The goal is to enable users to provide feedback on the relevance of search results by tagging them as "Very Relevant (R+)," "Relevant (R)," "Irrelevant (I)," or "Very Irrelevant (I-)." The feedback provided by the user will then be used to generate a new set of ranked results. The system should support either revising the query based on user feedback or re-ordering the existing results according to the feedback received. The task involves the development of algorithms and mechanisms to incorporate user feedback and improve the relevance of search results in an interactive manner.

Input:

a. Task4b output
b. User Feedback

Output:

a. Reordered task4b results based on the user feedback

Implementation:

a. Data Retrieval:
    i. The retrieve_results function loads the initial search results from a CSV file.
b. User Tagging:
    i. The get_user_tags function allows users to tag search results with relevance labels (R+, R, I, I-). The user can either continue tagging or quit.

c. SVM-based Relevance Feedback System:
    i. The train_svm function trains an SVM model based on user-tagged results.
    ii. The rank_svm_results function uses the trained SVM model to predict relevance scores and re-ranks the search results accordingly.
d. Probabilistic Relevance Feedback System:
    i. The estimate_relevance_probabilities function uses logistic regression to estimate probabilities of relevance for each class.
    ii. The rank_probabilistic_results function combines relevance scores, user tags, and Euclidean distances to produce a new ranking.
e. Main Execution:
    i. The svm_based_feedback_system and probabilistic_feedback_system functions integrate the entire process for their respective feedback systems.
    ii. The main block allows the user to choose between the SVM-based and probabilistic systems.

Interface Specifications

The directory hierarchy adheres to the following structure:

```
Phase 3/
|
├── Code/
|   ├── LSH_INDICES/
|   |       ├──lsh_index_details.pkl
|   ├── task0.py
|   ├── task1.py
|   ├── task2.py
|   ├── task3.py
|   ├── task4a.py
|   ├── task4b.py
|   ├── LSH.py
|   ├── utilities.py
```

```
│   └── task5.py
│
├── Outputs/
│   ├── T2
│   │    ├── image_clusters/
│   │    │      └── t2_label<labelno>_c<clusterno>.png
│   │    ├── point_clusers/
│   │           └── t2_label<labelno>.png
│   │    ├── t2_prediction_scores.txt
│   │    └── t2_label_predictions.txt
│   ├── T4
│   ├──  ├── task4.csv
│   │    └──id_1_t_10_layers_3_hashes_3_ts_1700965023.png
├── README.md
└── requirements.txt
```

1. Task 1:

For this task, the user interface specifications involve:

- **User Inputs:**
    a. Choices of features
    b. Choice of k which is the number of latent semantics for the features of the images in the dataset.
- **Output Generation:**
    a. The system generates and displays performance metrics such as precision, recall, F1-score, and overall accuracy for the predicted labels.

2. Task 2:

For this task, the user interface specifications involve:

- **User Inputs**
    - a value *c* (the number of most significant clusters to find)

```
Please enter a value for c:
```

- **Output Generation**
    - The *c* most significant clusters associated with the even numbered images
        - Visualization as differently colored point clouds
        - Visualization as groups of image thumbnails
    - The most likely labels of the odd numbered images
    - The precision, recall, and F1-score values and an overall accuracy value

3. Task 3:

For this task, the user interface specifications involve:

- **User Inputs:**
    a. Choice of Classifier
    b. Classifier-Specific Parameters (e.g., k for K-Nearest Neighbors, jump probability for Personalized PageRank)
    c. Selection of Input Data Source (Even or Odd numbered images)
- **Execution Flow:**
    a. Based on the classifier choice and provided parameters, the system trains the selected model on the designated dataset (even or odd images).
    b. Optionally, the system asks the user if they want to predict a specific class for an odd-numbered image.
- **Output Generation:**
    a. If predictions are requested, the system provides the predicted class label for the specified image.
    b. For the main execution without specific predictions, the system generates and displays performance metrics such as precision, recall, F1-score, and overall accuracy for the chosen classifier.

```
Please input classifier
1.m-NN Classifier
2.Decision Tree Classifier
3.PPR Classifier
1
Do you want to find out prediction for any specific odd image?[y/n]y
Please enter image id: 999
Please choose value of m: 5
 14%|          | 609/4338 [00:11<01:09, 53.35it/s]
```

4. Task 4:

Files structure and description

a. Code/LSH_INDICES

i. It contains the **lsh_index_details.pkl** file generated using **task4a.py** as described in the implementation section i.e. it contains

```
{
    "layers": self.layers, #contains number of layers
    "hashes": self.hashes, #contains number of hashes per layer
    "lsh_index": lsh_index #contains the index class structure
}
```

b. Code/task4a.py

i. Input

1. The number of layers, L
2. The number of hashes per layer, h, and
3. A set of vectors as input and creates an in-memory index structure containing the given set of vectors

ii. Output

1. Index in Pickle file

iii. Description:

1. This file contains the implementation of Task 4a.
2. It includes classes and functions for Locality Sensitive Hashing (LSH) index creation based on given vectors.

c. Code/task4b.py

i. Input:

1. A given query image
2. Integer t

ii. Output:

1. Visualizes the t most similar images

2. Outputs the numbers of unique
3. Overall number of images considered during the process
4. Output of t most similar image id and euclidean distance to csv file

iii. Description:
1. This file contains the implementation of Task 4b.
2. It includes classes and functions for image search using the LSH index created in Task 4a.

d. Code/LSH.py
i. This file likely contains the classes and functions related to Locality Sensitive Hashing. It is part of the implementation of LSH used in both Task 4a and Task 4b.

e. Code/utilities.py
i. This file likely contains utility functions used across the implementation.
ii. It includes functions for hash plotting result, distance calculation, feature extraction and other shared functionalities.

f. Output/T4/
i. This directory stores the output of Task 4, which includes the visualizations.
ii. The files are named based on different runs or parameters of Task 4. Example: id_1_t_10_layers_3_hashes_3_ts_1700965023.png
It means
1. Image id is 1
2. T is 10
3. Number of layers is 3
4. Number of hashes is 3
5. Timestamp of creation is 1700965023
iii. Files within these subdirectories likely include images representing the results of Task 4b executions.
iv. This folder stores the task4.csv file which contains the image id and euclidean distance as a csv format for task 5 purposes.

5. Task 5:

a. User Interaction:
i. Users choose between the SVM-based and probabilistic systems.
ii. Users tag search results with relevance labels during the interaction.
b. System Training:
i. The SVM-based system trains an SVM model based on user tags.
ii. The probabilistic system estimates relevance probabilities using logistic regression.
c. Result Ranking:

i. Both systems re-rank search results using different methodologies (SVM predictions or probabilistic scores).

d. Output:
   i. The final ranked results are printed for user consideration.

System Requirements/Installation and Execution Instructions

System Requirements:

- Python (>= 3.11)
- MongoDB
- Required Python libraries (`pickle`, `re`, `datetime`, `PIL`, `numpy`, `torch`, `tqdm`, `torchvision`, `matplotlib`, `tensorflow`, and `pymongo`)

Installation:

1. **Library Installation:**
   1. Change the directory to codebase's root directory.
   2. Make sure that python version 3.11.5 is installed.
   3. Activate virtual environment to avoid dependency conflict
      a. **python -m venv venv**
      b. **source venv/bin/activate**
   4. Install all the dependencies by installing requirements file
      a. **pip3 install -r requirements.txt**

Execution Instructions:

1. Task 1

a. Input:

```
Please pick one of the below options
1. HOG
2. Color Moments
3. Resnet Layer 3
4. Resnet Avgpool
5. Resnet FC
--------------
3
Please enter the value of k: 5
```

b. Output:

```
Label 0: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 1: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 2: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 3: Precision: 0.995, Recall: 1.9702970297029703, f1 Score: 1.3222591362126246.
Label 4: Precision: 0.21428571428571427, Recall: 0.0297029702970297, f1 Score: 0.05217391304347826.
Label 5: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 6: Precision: 0.3, Recall: 0.0297029702970297, f1 Score: 0.05405405405405406.
Label 7: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 8: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 9: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 10: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 11: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 12: Precision: 0.09375, Recall: 0.0297029702970297, f1 Score: 0.045112781954887216.
Label 13: Precision: 0.041666666666666664, Recall: 0.009900990099009901, f1 Score: 0.016.
Label 14: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 15: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 16: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 17: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 18: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 19: Precision: 0.3548387096774194, Recall: 0.10891089108910891, f1 Score: 0.16666666666666666.
Label 20: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 21: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 22: Precision: 0.0625, Recall: 0.009900990099009901, f1 Score: 0.017094017094017096.
Label 23: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 24: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 25: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 26: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 27: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 28: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 29: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 30: Precision: 0.0, Recall: 0.0, f1 Score: 0.
```

```
Label 86: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 87: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 88: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 89: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 90: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 91: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 92: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 93: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 94: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 95: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 96: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 97: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 98: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 99: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Label 100: Precision: 0.0, Recall: 0.0, f1 Score: 0.
Total accuracy: 0.10235131396957123
```

2. Task 2

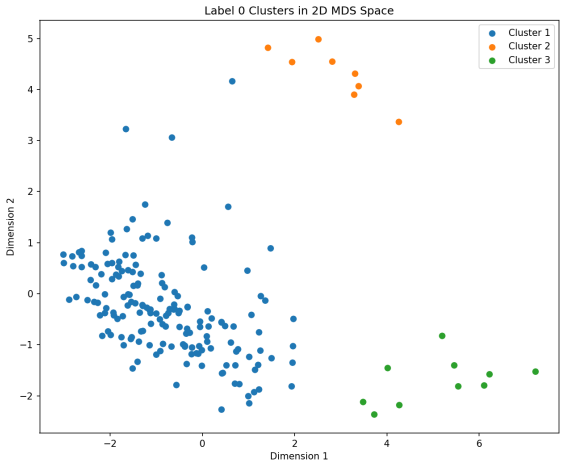
- Input: users must specify a value *c*. This value is used to determine the most significant clusters.

```
Please enter a value for c:
```

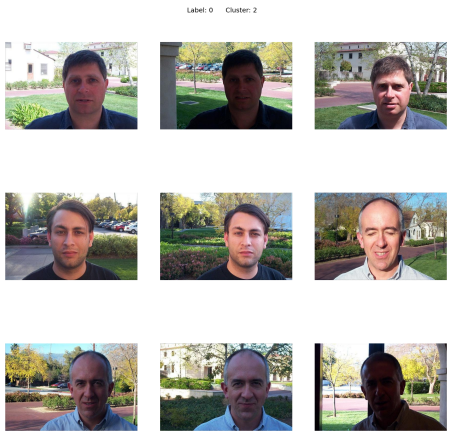
- Output:
  a. Most *c* significant clusters:
     - The output is printed to the console.

```
Label 6
c most significant clusters [1, 5, 4, 2, 3]
Label 7
c most significant clusters [2, 1, 3, 4, 5]
```

  b. Cloud Point Visualization:
     - The cloud point visualization of all the labels is printed into a png in the folder point_clusters



c. Thumbnail Visualization:
   - The visualization of each label is printed into a png in the folder image_clusters



d. Label Prediction:
   - This output is printed to the file [t2_label_predictions.txt]

```
Image ID: 1
 Predicted Label: 0

Image ID: 3
 Predicted Label: 0

Image ID: 5
 Predicted Label: 0
```

e. Precision, Recall, F1, Accuracy:
   - This output is printed to the file [t2_prediction_scores.txt]

```
Per-label Metrics:

Label: 0, Precision: 0.36660929432013767, Recall: 1.0, F1-Score: 0.5365239294710328

Label: 1, Precision: 1.0, Recall: 1.0, F1-Score: 1.0

Label: 2, Precision: 0.4444444444444444, Recall: 1.0, F1-Score: 0.6153846153846153
```

3. Task 3

m-NN Classification

Users can opt for `m-NN` classification and select `m` as `1`, `5`, or `10`. The program then executes predictions based on the chosen value of `m`. The output generated includes the predicted labels for the images under consideration. For instance:

```
Please input classifier
1.m-NN Classifier
2.Decision Tree Classifier
3.PPR Classifier
1
Do you want to find out prediction for any specific odd image?[y/n]y
Please enter image id: 999
Please choose value of m: 5
 14%|          | 609/4338 [00:11<01:09, 53.35it/s]
```

For m = 1: Prediction outcomes for images based on the nearest neighbor

```
Label 92 - Precision: 1.00, Recall: 0.00, F1-score: 0.00
Label 93 - Precision: 1.00, Recall: 0.00, F1-score: 0.00
Label 94 - Precision: 1.00, Recall: 0.01, F1-score: 0.02
Label 95 - Precision: 0.64, Recall: 0.00, F1-score: 0.00
Label 96 - Precision: 1.00, Recall: 0.00, F1-score: 0.00
Label 97 - Precision: 0.00, Recall: 0.00, F1-score: 0.00
Label 98 - Precision: 0.00, Recall: 0.00, F1-score: 0.00
Label 99 - Precision: 1.00, Recall: 0.00, F1-score: 0.00
Label 100 - Precision: 1.00, Recall: 0.01, F1-score: 0.01
Overall Accuracy: 0.22
```

For m = 5: Predictions considering the five nearest neighbors.

```
Label 92 - Precision: 0.95, Recall: 0.03, F1-score: 0.06
Label 93 - Precision: 1.00, Recall: 0.02, F1-score: 0.04
Label 94 - Precision: 0.97, Recall: 0.08, F1-score: 0.15
Label 95 - Precision: 0.60, Recall: 0.00, F1-score: 0.01
Label 96 - Precision: 1.00, Recall: 0.02, F1-score: 0.04
Label 97 - Precision: 1.00, Recall: 0.01, F1-score: 0.01
Label 98 - Precision: 0.92, Recall: 0.02, F1-score: 0.03
Label 99 - Precision: 0.86, Recall: 0.00, F1-score: 0.01
Label 100 - Precision: 0.80, Recall: 0.02, F1-score: 0.04
Overall Accuracy: 0.72
```

For m = 10: Predictions with a consideration of the ten nearest neighbors.

```
Label 92 - Precision: 0.88, Recall: 0.05, F1-score: 0.10
Label 93 - Precision: 1.00, Recall: 0.04, F1-score: 0.07
Label 94 - Precision: 0.92, Recall: 0.13, F1-score: 0.22
Label 95 - Precision: 0.71, Recall: 0.01, F1-score: 0.03
Label 96 - Precision: 1.00, Recall: 0.03, F1-score: 0.07
Label 97 - Precision: 1.00, Recall: 0.01, F1-score: 0.02
Label 98 - Precision: 0.79, Recall: 0.03, F1-score: 0.07
Label 99 - Precision: 0.57, Recall: 0.01, F1-score: 0.02
Label 100 - Precision: 0.76, Recall: 0.03, F1-score: 0.06
Overall Accuracy: 0.82
```

Decision Tree Classifier

```
Label 92 - Precision: 0.17, Recall: 0.00, F1-score: 0.01
Label 93 - Precision: 0.05, Recall: 0.00, F1-score: 0.00
Label 94 - Precision: 0.66, Recall: 0.03, F1-score: 0.05
Label 95 - Precision: 0.06, Recall: 0.00, F1-score: 0.00
Label 96 - Precision: 0.09, Recall: 0.00, F1-score: 0.00
Label 97 - Precision: 0.07, Recall: 0.00, F1-score: 0.00
Label 98 - Precision: 0.00, Recall: 0.00, F1-score: 0.00
Label 99 - Precision: 0.00, Recall: 0.00, F1-score: 0.00
Label 100 - Precision: 0.14, Recall: 0.00, F1-score: 0.00
Overall Accuracy: 0.36
```

In this case, users can choose the `Decision Tree` classifier. The program proceeds to execute predictions based on the decision tree model. The output comprises the predicted labels for the images in the dataset, showcasing the decision tree's classification results.

PPR Classifier:

Opting for the PPR (Personalized PageRank) classifier prompts the program to perform predictions using this specific algorithm. The output displays the predicted labels for the images, reflecting the results obtained from the Personalized PageRank classifier.

For random jump probability = 0.15

```
Label 92: Precision=0.17, Recall=0.00, F1-Score=0.01
Label 93: Precision=0.06, Recall=0.00, F1-Score=0.00
Label 94: Precision=0.69, Recall=0.03, F1-Score=0.05
Label 95: Precision=0.04, Recall=0.00, F1-Score=0.00
Label 96: Precision=0.06, Recall=0.00, F1-Score=0.00
Label 97: Precision=0.00, Recall=0.00, F1-Score=0.00
Label 98: Precision=0.06, Recall=0.00, F1-Score=0.00
Label 99: Precision=0.00, Recall=0.00, F1-Score=0.00
Label 100: Precision=0.11, Recall=0.00, F1-Score=0.00
Overall Accuracy: 0.62
```

For random jump probability = 0.5, the accuracy worsened with overall accuracy of 0.38

Individual Image Prediction:

For scenarios where users opt to predict for individual images, the program provides personalized outputs for each selected image. This output is tailored to the specific image's classification based on the chosen classifier, offering a detailed prediction outcome.

```
Prediction for provided image is: 1 & label is 2
```

For each of these scenarios, the program generates distinct outputs that are specific to the selected classifier, demonstrating its prediction performance and results for varying configurations.

4. Task 4:

a. Task 4a:

i. Input:

```
Please select numer of Layers, L

10

Please number of hashes per layer, h

10

LSH Index saved to ./LSH_INDICES/lsh_index_details.pkl
```

1. Number of layer **l**
2. Number of hashes per layer **h**

ii. Output:

1. **Code/LSH_INDICES/lsh_index_details.pkl** is generated which contains all the details required for lookup of image to get similar image result set

b. Task 4b:

i. Input:

```
Please select an image id or image path

2

Please select t to find t similar images

10

LSH Index loaded

Numbers of unique images considered during the process: 69

Overall number of images considered during the process: 83
```

1. **Image id / image path**
2. The parameter **t**

ii. Output:
Gives **t** similar image to the provided input image

1. Images in Output/T4/ as shown above

5. Task 5:

a. Input:

```
Select an option:
1 - SVM based relevance feedback system
2 - probabilistic relevance feedback system
Enter your choice: 1
```

i. Select which feedback system option you would like

```
    Image ID  Euclidean Distance User Tag
0       3100             0.00000
1       5752           195.92970
2       7556           197.23746
3       4372           199.55122
4       6300           200.45415
5       4004           201.11769
6       6688           201.34637
7       4494           205.79607
8       6346           207.51082
9       4948           209.26620
10      8180           209.86058
11      7582           220.53990
Enter image id for user tag or 'q' to quit: 4004
```

ii. Enter image id for the user to tag their feedback

```
Select an option:
1 - Very Relevant (R+)
2 - Relevant (R)
3 - Irrelevant (I)
4 - Very Irrelevant (I-)
5 - Stop
Enter your choice of user tag: 3
You selected Irrelevant (I) for Image ID: 4004
    Image ID  Euclidean Distance User Tag
0       3100            0.00000
1       5752          195.92970
2       7556          197.23746
3       4372          199.55122
4       6300          200.45415
5       4004          201.11769        I
6       6688          201.34637
7       4494          205.79607
8       6346          207.51082
9       4948          209.26620
10      8180          209.86058
11      7582          220.53990
```

iii. Enter the corresponding number for the feedback the user wants

b. Output

```
    Image ID  Euclidean Distance User Tag      Score
0       3100            0.00000        N  89.215960
2       7556          197.23746        R  10.620976
1       5752          195.92970        I  10.544080
3       4372          199.55122        N   9.395472
4       6300          200.45415        N   9.034300
5       4004          201.11769        N   8.768884
6       6688          201.34637        N   8.677412
7       4494          205.79607        N   6.897532
8       6346          207.51082        N   6.211632
10      8180          209.86058       R+   5.871728
9       4948          209.26620        N   5.509480
11      7582          220.53990        N   1.000000
```

i. Output results in the reordered images based on the user feedback.

Conclusions

1. Analysis of Task 1:

Overall, I felt the label predictions for task 1 were very low. There are a few reasons why the accuracy was low. Firstly, the dataset had imbalanced data distributions. Some labels had fewer images as compared to others. This imbalance results in a bias towards the majority label. Although we did try to lower the imbalance by performing individual Latent Semantic Analysis on each label, there was still some bias while calculating image

similarities. Secondly, there was not enough data for some of the labels. For example some of the labels had very few images, hence, not enough data to learn meaningful patterns.

2. Analysis of Task 2

As apparent by the cluster visualization, the clusters of some labels turned out better than others. This may be explained by a few aspects. Determining the eps and minPts parameters for the DBScan was difficult. Some of the parameter decisions may not be ideal for certain label data. Some of the clustering results may also be explained by a lack of data. The amount of data for each label varied greatly, some had hundreds while others only had ten. A difference in results was observed when this occurred.

With a score of 0.61, the overall accuracy of the predictions was average. It was not high, but it was not low. This may be explained by the feature descriptors used. The ResNet FC Layer was used, but other feature descriptors may have been better suited. Additionally, it may be because of the clustering. Better cluster combinations may have led to better precision, recall, F1, and accuracy scores.

3. Analysis of Task 3

m-NN Classifier

This output evaluates metrics of a K-Nearest Neighbors (KNN) classifier for different values of 'm' (the number of neighbors) on a multiclass classification problem. The metrics reported for each label (0 to 100) include precision, recall, and F1-score for each class, as well as an overall accuracy.

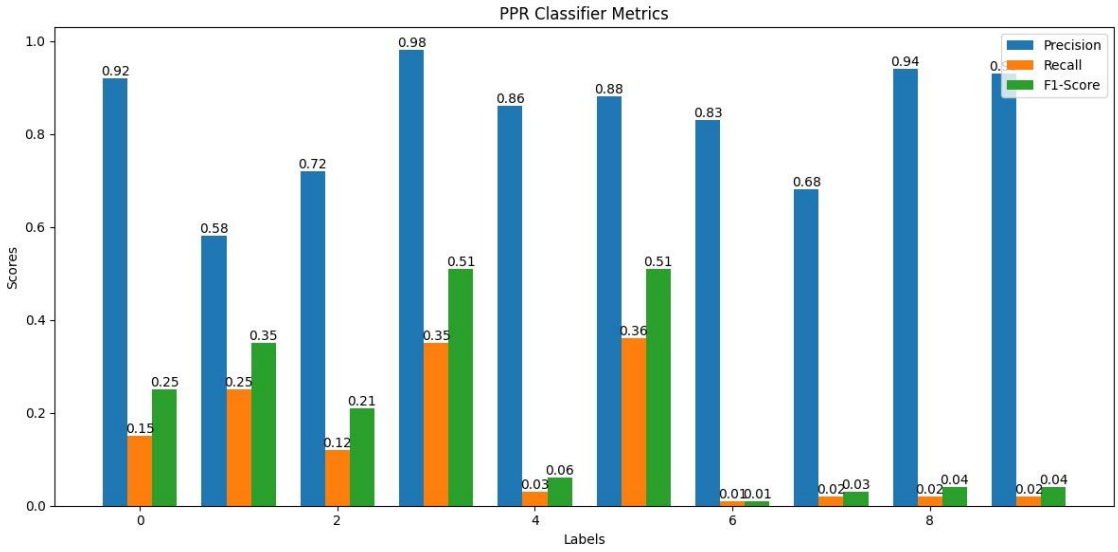
Here's a breakdown of what each metric means:

1. Precision: It measures the accuracy of the positive predictions. For each class, it's the ratio of correctly predicted instances of that class to the total instances predicted as that class.
2. Recall (also called Sensitivity): It measures how many of the actual positive instances were captured by the model. For each class, it's the ratio of correctly predicted instances of that class to the total actual instances of that class.
3. F1-score: It's the harmonic mean of precision and recall, providing a single score that balances both precision and recall. It's a good metric when there's an uneven class distribution.

Overall Accuracy: The ratio of correctly predicted instances to the total instances.

Looking at the output:

- For 'm = 1', the classifier doesn't perform well for most classes, as indicated by low precision, recall, and F1-scores. Overall accuracy is 0.22, suggesting the model's predictions are not very accurate.
- For 'm = 5', there's an improvement in the classifier's performance for many classes compared to 'm = 1'. Precision, recall, and F1-scores are generally higher across different classes, leading to a higher overall accuracy of 0.72.
- For 'm = 10', there's further improvement in precision, recall, and F1-scores for many classes compared to 'm = 5', resulting in an overall accuracy that might be even higher.



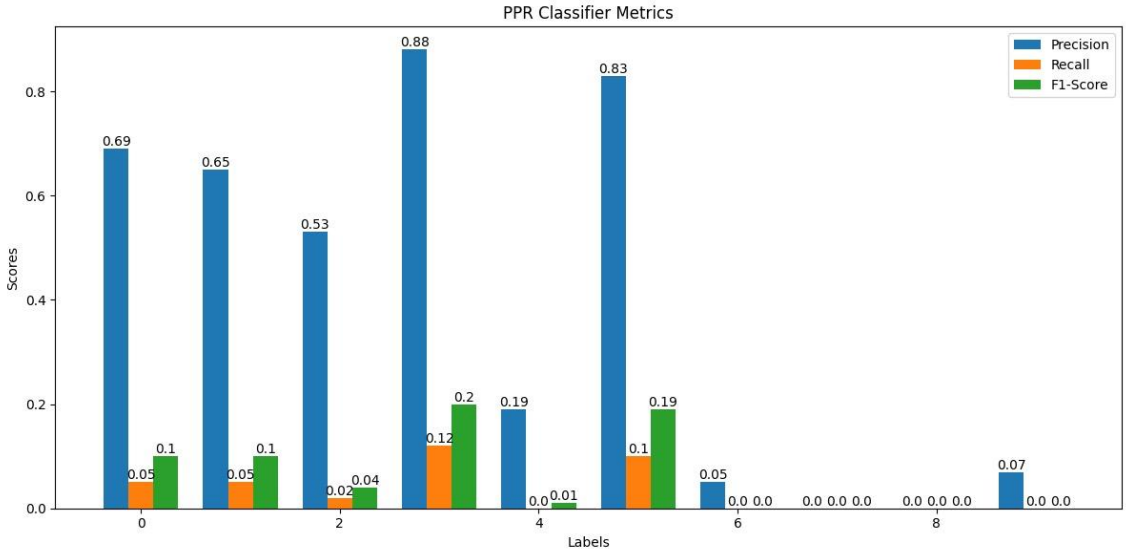
The increase in the number of neighbors used ('m') seems to generally improve the model's performance in correctly classifying instances across multiple classes. However, it's essential to consider other factors like data distribution, class imbalance, and the specific requirements of the problem when selecting the appropriate 'm' value.

In summary, higher values of 'm' seem to yield better performance based on these metrics, but a more detailed analysis of the problem and potentially other hyperparameters might be necessary for a comprehensive evaluation.

Decision Tree

- For most labels, precision is low, indicating that among the instances predicted as a certain class, only a small portion truly belong to that class.
- Recall is also low for most labels, suggesting that the classifier fails to capture a significant number of instances of each class.
- F1-scores are generally very low, which implies poor overall performance in terms of correctly identifying instances for each class.

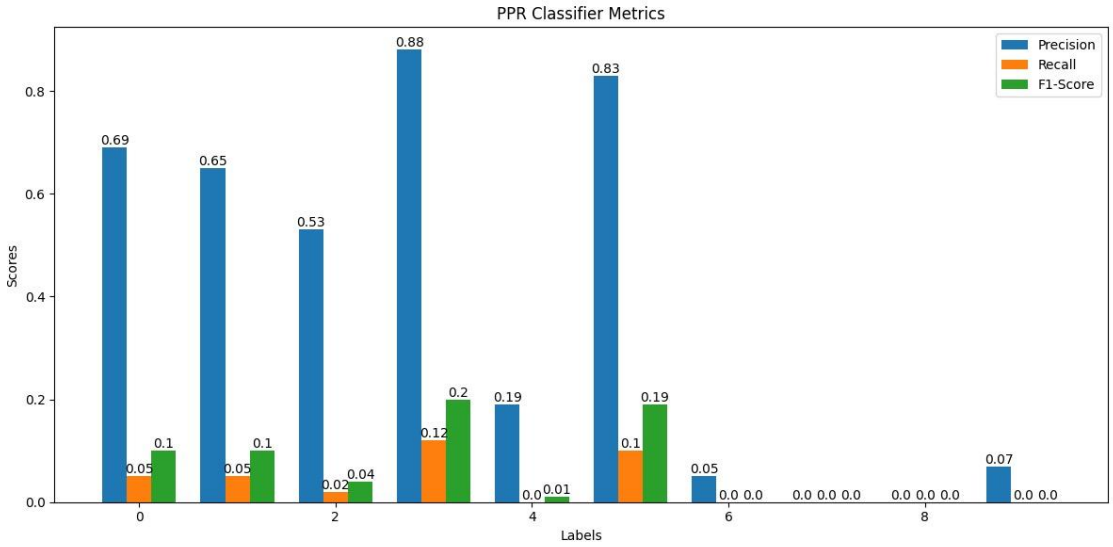
- The overall accuracy is 0.36, indicating that the classifier correctly predicts only 36% of the instances across all classes.
- The low precision and recall values across almost all labels indicate that the decision tree model struggles to effectively classify instances into their respective classes.
- A potential reason for this performance could be the limited depth of the tree (max depth = 3), which might not be enough to capture the complexity of the underlying relationships in the data.
- The imbalance in the dataset might also contribute to these low metrics, affecting the classifier's ability to learn from the minority classes.
- Increasing the maximum depth of the decision tree could capture more complex patterns in the data, potentially improving performance.
- Balancing the dataset or using techniques like stratified sampling might help the model better learn from the minority classes.
- Trying different splitting criteria or employing ensemble methods like random forests might also enhance the classifier's performance.



PRR Classifier

- The PPR classifier seems to have better performance compared to the earlier decision tree classifier, as evidenced by higher precision, recall, and F1-scores for various labels.
- Nevertheless, the overall performance remains moderate, with some classes having significantly better prediction scores than others.
- The overall accuracy of 0.62 indicates that the classifier correctly predicts 62% of instances across all classes.

- Like the decision tree, the PPR classifier also faces challenges in accurately classifying instances across multiple classes.
- It might be handling imbalanced classes better than the decision tree or employing more complex decision-making strategies.
- Fine-tuning model hyperparameters or considering ensemble methods could potentially enhance performance.
- Addressing class imbalance or exploring other algorithms might improve the classifier's ability to capture minority classes more effectively.
- Feature engineering or preprocessing techniques could also provide better insights into the data.



4. Analysis of Task 4

a. Test Case 1:

i. Inputs:

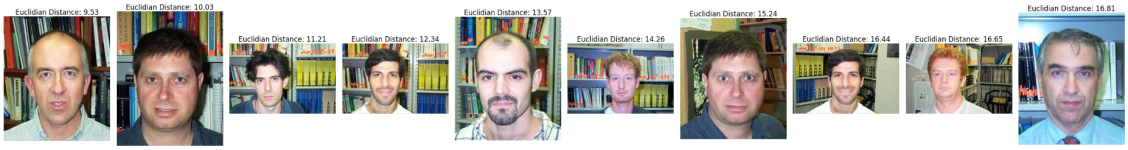
- **L=3**
- **h=3**
- **t=10**
- **Query image: 1**

ii. Outputs:

Numbers of unique images considered during the process: 1135
Overall number of images considered during the process: 1280

Input Image:

10 similar images using LSH with 3 layer(s) and 3 hash(es)



iii. Explanation:

In our Locality-Sensitive Hashing (LSH) implementation, we've opted for a random hyperplane as our hashing function. This hyperplane discretely assigns vectors to either 0 or 1 based on the result of their dot product with the hyperplane. The entire hash code is then derived by employing h such hash functions, effectively creating 2^h buckets or index keys. With our choice of h being 3, we find ourselves with a total of 8 plausible buckets, forming the basis for our result set.

Given that we've set h to 3, the number of plausible buckets is 8, which makes the result set returned by each layer slightly larger. This design choice is intentional, and we've also chosen a relatively large W to minimize misses in our retrieval process. While this might lead to a more extensive result set(1280), it is a deliberate trade-off to ensure we don't overlook potential matches. We've prioritized minimizing false negatives during the initial hashing process.

However, we recognize that false positives might arise due to this approach. To address this, we've incorporated a robust post-processing step. Any false positives in the result set can be effectively identified and eliminated during this stage. This strategic decision allows us to achieve a high level of accuracy in our results, even in the presence of potential false positives.

b. Test Case 2

i. Inputs:

- **L=3**
- **h=3**
- **t=10**
- **Query images: 881**

ii. Output

Numbers of unique images considered during the process: 1866

Overall number of images considered during the process: 2270

iii. Explanation:

In our Locality-Sensitive Hashing (LSH) implementation, we've opted for a random hyperplane as our hashing function. This hyperplane discretely assigns vectors to either 0 or 1 based on the result of their dot product with the hyperplane. The entire hash code is then derived by employing h such hash functions, effectively creating 2^h buckets or index keys. With our choice of h being 3, we find ourselves with a total of 8 plausible buckets, forming the basis for our result set.

Given that we've set h to 3, the number of plausible buckets is 8, which makes the result set returned by each layer slightly larger. This design choice is intentional, and we've also chosen a relatively large W to minimize misses in our retrieval process. While this might lead to a more extensive result set(2270), it is a deliberate trade-off to ensure we don't overlook potential matches. We've prioritized minimizing false negatives during the initial hashing process.

However, we recognize that false positives might arise due to this approach. To address this, we've incorporated a robust post-processing step. Any false positives in the result set can be effectively identified and eliminated during this stage. This strategic decision allows us to achieve a high level of accuracy in our results, even in the presence of potential false positives.

c. Test Case 3

i. Inputs:

- **L=3**
- **h=3**
- **t=10**
- **Query images: 2501**

ii. **Outputs**

Numbers of unique images considered during the process: 1941
Overall number of images considered during the process: 2734

Input Image:

10 similar images using LSH with 3 layer(s) and 3 hash(es)

iii. **Explanation:**

d. **Test Case 4**

i. **Inputs:**

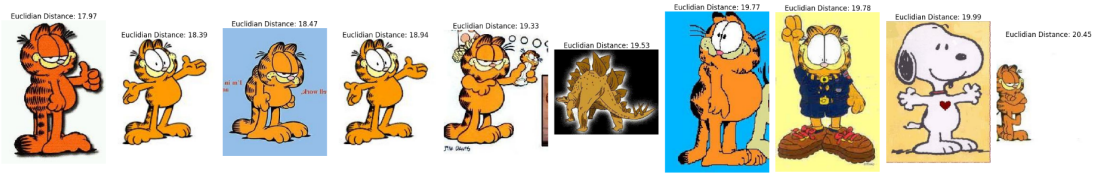
- **L=3**
- **h=3**
- **t=10**
- **Query images: 2501**

ii. **Outputs**

Numbers of unique images considered during the process: 1829
Overall number of images considered during the process: 2170

Input Image:

10 similar images using LSH with 3 layer(s) and 3 hash(es)



iii. **Explanation:**

In our Locality-Sensitive Hashing (LSH) implementation, we've opted for a random hyperplane as our hashing function. This hyperplane discretely assigns vectors to either 0 or 1 based on the result of their dot product with the hyperplane. The entire hash code is then derived by employing h

such hash functions, effectively creating 2^h buckets or index keys. With our choice of h being 3, we find ourselves with a total of 8 plausible buckets, forming the basis for our result set.

Given that we've set h to 3, the number of plausible buckets is 8, which makes the result set returned by each layer slightly larger. This design choice is intentional, and we've also chosen a relatively large W to minimize misses in our retrieval process. While this might lead to a more extensive result set(2170), it is a deliberate trade-off to ensure we don't overlook potential matches. We've prioritized minimizing false negatives during the initial hashing process.

However, we recognize that false positives might arise due to this approach. To address this, we've incorporated a robust post-processing step. Any false positives in the result set can be effectively identified and eliminated during this stage. This strategic decision allows us to achieve a high level of accuracy in our results, even in the presence of potential false positives.

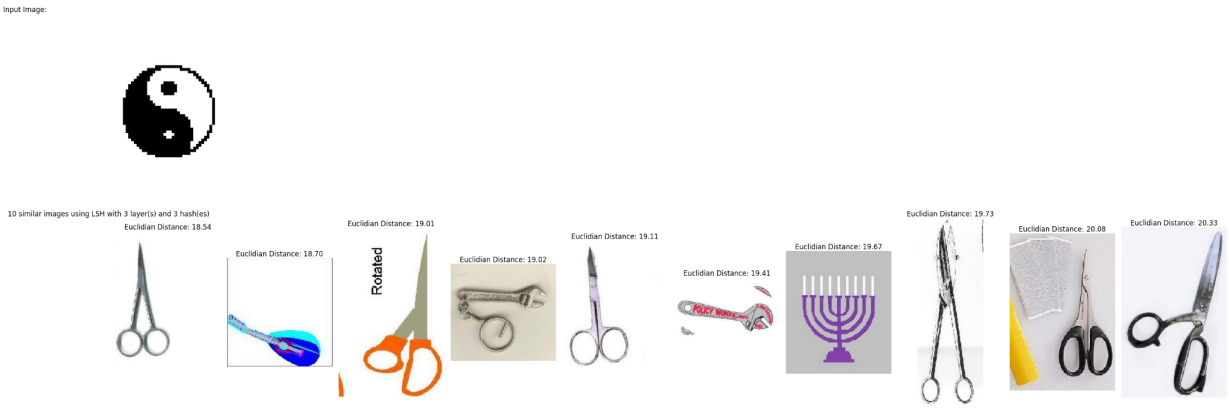
e. Test Case 5

i. Inputs:

- **L=3**
- **h=3**
- **t=10**
- **Query images: 5123**

ii. Output

Numbers of unique images considered during the process: 1866
Overall number of images considered during the process: 2270



iii. Explanation:

In our Locality-Sensitive Hashing (LSH) implementation, we've opted for a

random hyperplane as our hashing function. This hyperplane discretely assigns vectors to either 0 or 1 based on the result of their dot product with the hyperplane. The entire hash code is then derived by employing h such hash functions, effectively creating 2^h buckets or index keys. With our choice of h being 3, we find ourselves with a total of 8 plausible buckets, forming the basis for our result set.

Given that we've set h to 3, the number of plausible buckets is 8, which makes the result set returned by each layer slightly larger. This design choice is intentional, and we've also chosen a relatively large W to minimize misses in our retrieval process. While this might lead to a more extensive result set(2270), it is a deliberate trade-off to ensure we don't overlook potential matches. We've prioritized minimizing false negatives during the initial hashing process.

However, we recognize that false positives might arise due to this approach. To address this, we've incorporated a robust post-processing step. Any false positives in the result set can be effectively identified and eliminated during this stage. This strategic decision allows us to achieve a high level of accuracy in our results in general, even in the presence of potential false positives **but** in this case input image is pixelated and pixelation involves reducing the resolution of an image, and in the process, some fine details are lost. If the pixelated version lacks crucial information that ResNet50 relies on for distinguishing images, the feature vectors may not capture the essence of similarity accurately due to which we are seeing inaccurate output. In conclusion it is due to the nature of the image and its Resnet50 FC feature vector rather than LSH indexing.

f. Test Case 6

i. Inputs:

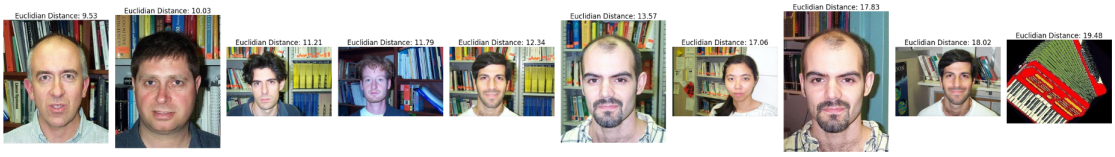
- **L=10**
- **h=10**
- **t=10**
- **Query images: 1**

ii. Outputs

Numbers of unique images considered during the process: 76
Overall number of images considered during the process: 91

Input Image:

10 similar images using LSH with 10 layer(s) and 10 hash(es)



iii. Explanation:

In our Locality-Sensitive Hashing (LSH) implementation, we've opted for a random hyperplane as our hashing function. This hyperplane discretely assigns vectors to either 0 or 1 based on the result of their dot product with the hyperplane. The entire hash code is then derived by employing h such hash functions, effectively creating 2^h buckets or index keys. With our choice of h being 3, we find ourselves with a total of 1024 plausible buckets, forming the basis for our result set.

Given that we've set h to 10, the number of plausible buckets is 1024, which makes the result set returned by each layer smaller. Similar to the previous case, the LSH process has effectively narrowed down the search space to 76 unique images that share similar hash codes with the query image (1), out of a total of 91 images considered during the process.

The parameters L, h, and t, along with the specific hash functions used, contribute to the behavior of the LSH algorithm and the resulting set of similar images.

The fact that the number of unique images considered is smaller than the overall number of images considered suggests that there is some degree of redundancy or overlap in the hash codes of different images, leading to fewer unique images being considered during the process. This strategic decision allows us to achieve a high level of accuracy in our results

g. Test Case 7

i. Inputs:

- **L=10**
- **h=10**

- **t=10**
- **Query images: 881**

ii. Outputs

Numbers of unique images considered during the process: 96
Overall number of images considered during the process: 127

iii. Explanation:

In our Locality-Sensitive Hashing (LSH) implementation, we've opted for a random hyperplane as our hashing function. This hyperplane discretely assigns vectors to either 0 or 1 based on the result of their dot product with the hyperplane. The entire hash code is then derived by employing h such hash functions, effectively creating 2^h buckets or index keys. With our choice of h being 3, we find ourselves with a total of 1024 plausible buckets, forming the basis for our result set.

Given that we've set h to 10, the number of plausible buckets is 1024, which makes the result set returned by each layer smaller. Similar to the previous case, the LSH process has effectively narrowed down the search space to 96 unique images that share similar hash codes with the query image (881), out of a total of 127 images considered during the process.

The parameters L, h, and t, along with the specific hash functions used, contribute to the behavior of the LSH algorithm and the resulting set of similar images.

The fact that the number of unique images considered is smaller than the overall number of images considered suggests that there is some degree of redundancy or overlap in the hash codes of different images, leading to fewer unique images being considered during the process. This strategic decision allows us to achieve a high level of accuracy in our results

h. Test Case 8

i. Inputs:

- **L=10**
- **h=10**
- **t=10**
- **Query images: 2501**

ii. Outputs

Numbers of unique images considered during the process: 287
Overall number of images considered during the process: 432

Input Image:

10 similar images using LSH with 10 layer(s) and 10 hash(es)

iii. Explanation:

In our Locality-Sensitive Hashing (LSH) implementation, we've opted for a random hyperplane as our hashing function. This hyperplane discretely assigns vectors to either 0 or 1 based on the result of their dot product with the hyperplane. The entire hash code is then derived by employing h such hash functions, effectively creating 2^h buckets or index keys. With our choice of h being 3, we find ourselves with a total of 1024 plausible buckets, forming the basis for our result set.

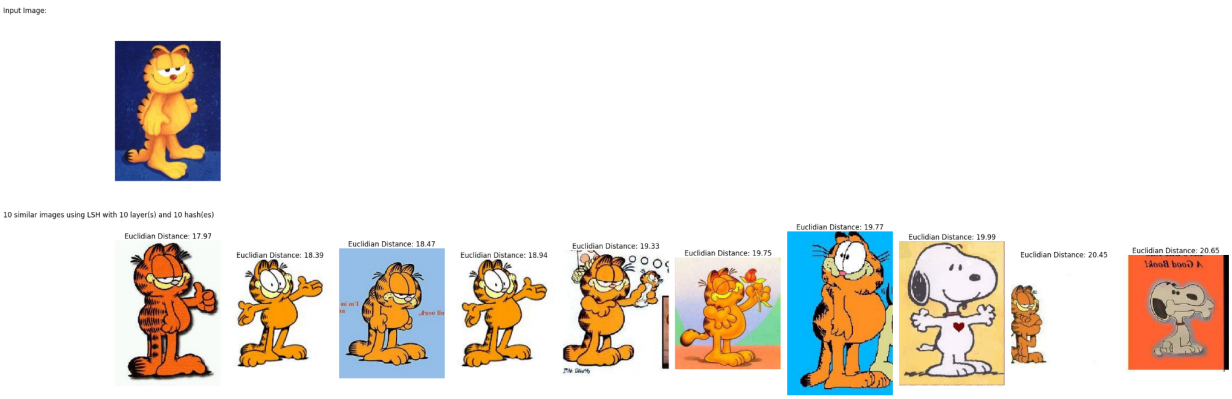
Given that we've set h to 10, the number of plausible buckets is 1024, which makes the result set returned by each layer smaller. Similar to the previous case, the LSH process has effectively narrowed down the search space to 287 unique images that share similar hash codes with the query image (2501), out of a total of 432 images considered during the process.

The parameters L, h, and t, along with the specific hash functions used, contribute to the behavior of the LSH algorithm and the resulting set of similar images.

The fact that the number of unique images considered is smaller than the overall number of images considered suggests that there is some degree of redundancy or overlap in the hash codes of different images, leading to fewer unique images being considered during the process. This strategic decision allows us to achieve a high level of accuracy in our results

i. **Test Case 9**
   i. **Inputs:**
      - **L=10**
      - **h=10**
      - **t=10**
      - **Query images: 5123**
   ii. **Output**

Numbers of unique images considered during the process: 69
Overall number of images considered during the process: 73



iii. **Explanation:**

In our Locality-Sensitive Hashing (LSH) implementation, we've opted for a random hyperplane as our hashing function. This hyperplane discretely assigns vectors to either 0 or 1 based on the result of their dot product with the hyperplane. The entire hash code is then derived by employing h such hash functions, effectively creating 2^h buckets or index keys. With our choice of h being 3, we find ourselves with a total of 1024 plausible buckets, forming the basis for our result set.

Given that we've set h to 10, the number of plausible buckets is 1024, which makes the result set returned by each layer smaller. Similar to the previous case, the LSH process has effectively narrowed down the search space to 69 unique images that share similar hash codes with the query image (5123), out of a total of 73 images considered during the process.

The parameters L, h, and t, along with the specific hash functions used, contribute to the behavior of the LSH algorithm and the resulting set of similar images.

The fact that the number of unique images considered is smaller than the overall number of images considered suggests that there is some degree of redundancy or overlap in the hash codes of different images, leading to fewer unique images being considered during the process. This strategic decision allows us to achieve a high level of accuracy in our results

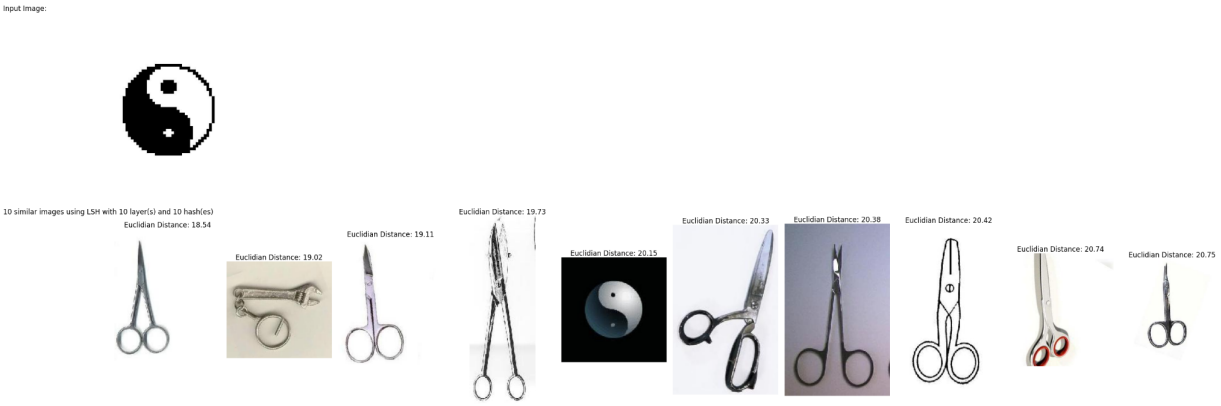
j. Test Case 10

i. Inputs:

- **L=10**
- **h=10**
- **t=10**
- **Query images: 8675**

ii. Output:

Numbers of unique images considered during the process: 143
Overall number of images considered during the process: 150



iii. Explanation:

In our Locality-Sensitive Hashing (LSH) implementation, we've opted for a random hyperplane as our hashing function. This hyperplane discretely assigns vectors to either 0 or 1 based on the result of their dot product with the hyperplane. The entire hash code is then derived by employing h such hash functions, effectively creating 2^h buckets or index keys. With our choice of h being 3, we find ourselves with a total of 1024 plausible buckets, forming the basis for our result set.

Given that we've set h to 10, the number of plausible buckets is 1024, which makes the result set returned by each layer smaller. Similar to the previous case, the LSH process has effectively narrowed down the search space to 143 unique images that share similar hash codes with the query image (8675), out of a total of 150 images considered during the process.

The parameters L, h, and t, along with the specific hash functions used, contribute to the behavior of the LSH algorithm and the resulting set of similar images.

The fact that the number of unique images considered is smaller than the overall number of images considered suggests that there is some degree of redundancy or overlap in the hash codes of different images, leading to fewer unique images being considered during the process. This strategic decision allows us to achieve a high level of accuracy in our results in general **but** in this case input image is pixelated and pixelation involves reducing the resolution of an image, and in the process, some fine details are lost. If the pixelated version lacks crucial information that ResNet50 relies on for distinguishing images, the feature vectors may not capture the essence of similarity accurately due to which we are seeing inaccurate output. In conclusion it is due to the nature of the image and its Resnet50 FC feature vector rather than LSH indexing.

5. Analysis of Task 5

The code successfully implements a relevance feedback system where users can tag search results as "Very Relevant (R+)," "Relevant (R)," "Irrelevant (I)," or "Very Irrelevant (I-)." The feedback from users is then used to reorder the initial search results, reflecting the perceived relevance based on user input.

The code considers Euclidean distance as one of the factors in the result ranking process. The Euclidean distance is calculated for each result, and the results are initially ordered based on this distance. However, there's an interesting observation related to the influence of Euclidean distance on the reordering process. While user feedback might improve the score of a result, especially if identified as relevant, it might not necessarily lead to a significant change in the reordering of results if there is a large gap in Euclidean distances. The impact of Euclidean distance appears to be dominant in the initial ordering, and user feedback, although affecting the score, might not overcome this distance-based ordering unless the user tags significantly shift the relevance perception.

The code aims to ensure that if a user identifies a result as relevant, it will improve the score, potentially leading to a higher ranking. The reordering, however, is influenced not only by user feedback but also by the Euclidean distance, which might introduce

nuances in the final ranking. There seems to be a trade-off between the impact of user feedback and the influence of Euclidean distance on the reordering process. If there is a substantial gap in Euclidean distances, user feedback might not be sufficient to overcome the initial ordering based on distance alone.

Related Work

Numerous researchers in the field of computer vision and machine learning have delved into image classification techniques, each bringing unique contributions to the table. The exploration of Convolutional Neural Networks (CNNs) by pioneers like Yann LeCun and Geoffrey Hinton revolutionized image classification, achieving remarkable accuracy on various datasets. Additionally, decision trees and ensemble methods, advocated by Leo Breiman and others, have shown promising results, offering interpretability and robustness in classification tasks. k-Nearest Neighbors (k-NN), introduced in pattern recognition by Evelyn Fix and Joseph Hodges, remains a fundamental algorithm for image similarity and classification due to its simplicity and effectiveness. Researchers have extensively evaluated classifiers using diverse evaluation metrics such as precision, recall, and F1-score, notably showcased in works by Gerard Salton and Chris Buckley. The application of these classifiers spans across industries: CNNs are deployed in medical imaging for disease diagnosis, decision trees are utilized in recommendation systems, while k-NN finds use in content-based image retrieval systems. Recent advancements have also explored personalized PageRank (PPR)-based algorithms for image classification, blending graph-based approaches with image databases for improved retrieval and classification accuracy. These approaches collectively contribute to advancing the understanding and practical implementation of image classification in various real-world applications.

References

B. Chaudhuri, B. Demir, S. Chaudhuri and L. Bruzzone, "Multilabel Remote Sensing Image Retrieval Using a Semisupervised Graph-Theoretic Method," in *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 2, pp. 1144-1158, Feb. 2018, doi: 10.1109/TGRS.2017.2760909.

Belhumeur, P. N., Hespanha, J. P., & Kriegman, D. J. (1997). Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(7), 711-720.

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

Li, W., Liu, A., Nie, W., Song, D., Li, Y., Wang, W., Xiang, S., Zhou, H., Bui, N.-M., Cen, Y., Chen, Z., Chung-Nguyen, H.-H., Diep, G.-H., Do, T.-L., Doubrovski, E. L., Duong, A.-D., Geraedts, J. M. P., Guo, H., Hoang, T.-H., … Zhao, S. (1970, January 1). *Monocular image based 3D model retrieval*. Eurographics DL Home. https://diglib.eg.org/handle/10.2312/3dor20191068

Lu Zhu, Jiang Zhu, Chongming Bao, Lihua Zhou, Chongyun Wang, and Bing Kong. 2018. Improvement of DBSCAN Algorithm Based on Adaptive Eps Parameter Estimation. In Proceedings of the 2018 International Conference on Algorithms, Computing and Artificial Intelligence (ACAI '18). Association for Computing Machinery, New York, NY, USA, Article 27, 1–7. https://doi.org/10.1145/3302425.3302493

Shanmukh, Vegi (2021). Image Classification Using Machine Learning-Support Vector Machine(SVM).https://medium.com/analytics-vidhya/image-classification-using-machine-learning-support-vector-machine-svm-dc7a0ec92e01

Suresh, S., Mohan, S. ROI-based feature learning for efficient true positive prediction using convolutional neural network for lung cancer diagnosis. *Neural Comput & Applic* 32, 15989–16009 (2020). https://doi.org/10.1007/s00521-020-04787-w

Vo, Tan & Tran, Dat & Ma, Wanli & Nguyen, Khoa. (2013). Improved HOG Descriptors in Image Classification with CP Decomposition. 8228. 384-391. 10.1007/978-3-642-42051-1_48.
”Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions” (by Alexandr Andoni and Piotr Indyk). Communications of the ACM, vol. 51, no. 1, 2008, pp. 117-122

Zhou, X.S., Huang, T.S.: Relevance feedback in image retrieval: A comprehensive review. Multimedia Systems 8 (2003) 536–544

Filipovska A, Rackham O. Pentatricopeptide repeats: modular blocks for building RNA-binding proteins. RNA Biol. 2013;10(9):1426-32. doi: 10.4161/rna.24769. Epub 2013 Apr 23. PMID: 23635770; PMCID: PMC3858425.

Appendix

- **Task 1: Implemented by Maryam Cheema**
- **Task 2: Implemented by Ariana Bui**
- **Task 3: Implemented by Yash Dhamecha**
- **Task 4: Implemented by Rajat Pawar**
- **Task 5: Implemented by Peter Kim**