

# Project Report 4

## Project Based On Hashing: Querying in Face Datasets

Name : Dhivakar .R

Course : AI and ML

### Problem Statement

Perform activity recognition on the dataset using a hidden markov model. Then perform the same task using a different classification algorithm (logistic regression/decision tree) of your choice and compare the performance of the two algorithms

### Prerequisites

What things you need to install the software and how to install them:

Python 3.6 This setup requires that your machine has latest version of python. The following url <https://www.python.org/downloads/> can be referred to download python. Once you have python downloaded and installed, you will need to setup PATH variables (if you want to run python program directly, detail instructions are below in how to run software section). To do that check this: <https://www.pythoncentral.io/add-python-to-path-python-is-not-recognized-as-an-internal-or-external-command/> . Setting up PATH variable is optional as you can also run program without it and more instruction are given below on this topic. Second and easier option is to download anaconda and use its anaconda prompt to run the commands

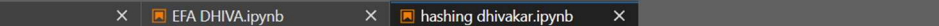
To install anaconda check this url <https://www.anaconda.com/download/> You will also need to download and install below 3 packages after you install either python or anaconda from the steps above Sklearn (scikit-learn) numpy scipy if you have chosen to install python 3.6

Tools: Pandas, Numpy , Matplotlib , Sklearn , Seaborn

### Method used :

Implement a basic hashing model from scratch that hashes the images. You can use any dataset of few images and can implement a-hash or any other hashing algorithm of your choice. For a-hash, given any images, first resize the image to a suitable size, followed by grayscale conversion of the image. Then mean normalize the image to obtain a binary image, whose sum can be used as a hash value. Using the hash model, encode all the images present inside your directory and then search for images similar to the query image

## Load all required libraries and data



The screenshot shows a JupyterLab window with three tabs: 'Launcher', 'EFA DHIVA.ipynb', and 'hashing dhivakar.ipynb'. The 'hashing dhivakar.ipynb' tab is active, displaying a code cell with the following Python code:

```
[24]: import cv2
import os
import pandas as pd
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
```

## Reading multiple images from a folder and storing it in a list.

```
[25]: # loading multiple images from a folder and storing it in a list.
folder = "/training_real"
images = []
for file in os.listdir(folder):
    img = mpimg.imread(os.path.join(folder, file))
    if img is not None:
        images.append(img)
print(images)

[array([[ 46, 34, 26, ..., 19, 13, 22],
       [ 50, 28, 21, ..., 11, 10, 21],
       [ 50, 27, 15, ..., 22, 24, 22],
       ...,
       [208, 179, 108, ..., 191, 205, 204],
       [212, 198, 169, ..., 181, 200, 204],
       [208, 203, 171, ..., 165, 188, 189]], dtype=uint8), array([[ 55, 33, 54, ..., 40, 42, 52],
       [ 43, 42, 36, ..., 62, 75, 85],
       [ 45, 54, 18, ..., 84, 102, 111],
       ...,
       [242, 240, 242, ..., 152, 177, 186],
       [244, 242, 242, ..., 117, 183, 193],
       [245, 242, 245, ..., 134, 163, 167]], dtype=uint8), array([[255, 255, 255, ..., 76, 90, 88],
       [255, 255, 255, ..., 80, 92, 85],
       [255, 255, 255, ..., 90, 93, 92],
       ...,
       [ 28, 24, 31, ..., 39, 40, 30],
       [ 27, 21, 46, ..., 39, 35, 37],
       [ 39, 28, 50, ..., 37, 37]], dtype=uint8), array([[248, 243, 254, ..., 162, 166, 157],
```

### Vectorizing the images and storing it in a list & converting the image vector to a unit vector

```
[26]: # Vectorizing the images and storing it in a list
image_vector = []
for image in images:
    row,col = image.shape
    img_vec = image.reshape(row*col)
    img_vec_norm = img_vec / np.linalg.norm(img_vec) # Converting the image vector to a unit vector
    image_vector.append(img_vec_norm)
print(img_vec.shape)
print(len(image_vector))

(2304,)
383
```

### Generate random unit vectors for Hashing

```
[29]: def genRandomHashVectors(m, length): # Generate random unit vectors for Hashing
      hash_vector = []
      for i in range(m):
          v = np.random.uniform(-1,1,length)
          vcap = v / np.linalg.norm(v)
          hash_vector.append(vcap)
      return hash_vector

[30]: def ahash(hash_vector ,data):
      hash_code = []
      for i in range(len(hash_vector)):
          if np.dot(data,hash_vector[i]) > 0:
              hash_code.append('1')
          else:
              hash_code.append('0')
      return ''.join(hash_code)

[31]: hash_vector = genRandomHashVectors(20,len(image_vector[0]))
      print(ahash(hash_vector,image_vector[0]))

11000100010000001110
```

## Generate random unit vectors for Hashing

```
[29]: def genRandomHashVectors(m, length): # Generate random unit vectors for hashing
      hash_vector = []
      for i in range(m):
          v = np.random.uniform(-1,1,length)
          vcap = v / np.linalg.norm(v)
          hash_vector.append(vcap)
      return hash_vector

[30]: def ahash(hash_vector, data):
      hash_code = []
      for i in range(len(hash_vector)):
          if np.dot(data, hash_vector[i]) > 0:
              hash_code.append('1')
          else:
              hash_code.append('0')
      return ''.join(hash_code)

[31]: hash_vector = genRandomHashVectors(20, len(image_vector[0]))
      print(ahash(hash_vector, image_vector[0]))
      11000100010000001110
```

## Creating a Image Dictionary using the hash as the keys

[illegible]

## Output

### Plotting images with same hash code

```
[9]: # shuffling images with same basic code
def plot_images(images, img_indices):
    imgs = [images[i] for i in range(len(images)) if i in img_indices]
    fig = plt.figure()
    cols = 2
    n_images = len(imgs)
    for n_image in range(n_images):
        ax = fig.add_subplot(cols, n_images//cols, n_image + 1)
        plt.imshow(imgs[n_image])
        plt.title(imgs[n_image].get_size_inches()[0] * n_images)
    fig.savefig('p_array(fig.get_size_inches()) * n_images')
    plt.show()
```

[10]: plot\_images(images, values[1])