Assignment Project Report

Hash: Querying in Face Datasets

Name: Arun Govind
Course: AI and ML
(Batch 4)

- **Problem Statement**

  Perform a hashing algorithm on any dataset of few images.

- **Prerequisites**

  1. Software:
     - Python 3 (Use anaconda as your python distributor as well)

  2. Tools:
     - Numpy
     - ImageHash
     - Matplot

  3. Dataset used: Yale face dataset

- **Method Used**

  Hashing is a function that applies to an arbitrary data and produces the data of a fixed size (mostly a very small size). There are many different types of hashes, but if we are talking about image hashing, it is used either to:
  - Find duplicates very fast. Almost any hash function will work. Instead of searching for the whole image, you will look for the hash of the image.
  - Finding similar images

  A hash function takes an input as a key, which is associated with a datum or record and used to identify it to the data storage and retrieval application.

- **Implementation:**

  1. Code for Importing all Libraries to fetch and arrange data:

```
import os
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
```

```
folder = "./YALE"
images = []
for file in os.listdir(folder):
    img = mpimg.imread(os.path.join(folder, file))
    if img is not None:
        images.append(img)
```

  2. Convert images to Vector format and then use hashing technique:

```
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)
```

```
def genRandomHashVectors(m, length):
    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
```

```
def localSensitiveHashing(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)
```

3. Defining dictionary keys as hash keys and assigning values to each photo:

In [10]:
```
image_dict = {}
for i in range(len(image_vector)):
    hash_code = localSensitiveHashing(hash_vector,image_vector[i])
    if hash_code not in image_dict.keys():
        image_dict[hash_code] = [i]
    else:
        image_dict[hash_code].append(i)
```

In [11]:
```
print(image_dict)
```

```
{'10100111001101010100': [0], '10100010101101010100': [1, 27, 92], '10100011101101010100': [2, 11, 13, 32, 67, 77,
84, 101, 104, 132, 141, 158, 162], '10100010100011010100': [3, 18, 24, 159], '10100011100001010100': [4, 15, 48, 5
4, 91, 93], '10100011101001010100': [5, 10, 31, 33, 44, 46, 63, 88, 109, 124, 136, 137, 144, 155], '101001111011010
10100': [6, 8], '10100011101011010100': [7, 86, 133], '10110111101001000100': [9], '10100010101001010100': [12, 34,
35, 53, 65, 94, 98, 135, 143], '10000010101111010100': [14], '10000111101001010100': [16, 50], '10000010001001011110
0': [17], '10100011100011010100': [19, 36, 40, 42, 74, 116, 118, 123, 146], '10100010101011010100': [20], '10000011
10110101010100': [21, 76, 80, 96, 120], '10100110101001010100': [22, 69], '10100011100101010100': [23, 70, 73, 157],
'10110111001001010100': [25], '10100111011001010100': [26], '00100111101101010100': [28], '10100111101011010100':
[29], '10100010001001010100': [30, 138], '00100111100001010100': [37], '00100111111001010100': [38], '1010011100100
1010100': [39, 64], '10000011101001010100': [41, 85, 99, 103, 111, 121, 122, 145], '10100010001011010100': [43], '1
0100111101001010100': [45, 66, 71, 83, 114, 130, 131, 142, 150, 160], '00100110100001010100': [47], '10100011000001
010100': [49, 72, 110], '10010010001001011110': [51], '10100111101001000100': [52], '10110110101101010100': [55], '
10000111100001010100': [56], '10100010001011011100': [57], '10000010100001010100': [58, 140], '10110011101101010100
': [59, 125, 148], '10110111101000010100': [60], '00100111100101010100': [61], '10000010001011010100': [62], '10100
111100001010100': [68, 87, 164], '10000010101001010100': [75, 163], '00100111111001001100': [78], '1011011000101101
0100': [79], '10100010100111010100': [81], '10100110101011011100': [82], '00010111011001010101': [89], '10000011100
001010100': [90], '00101010001011010100': [95], '10100111101001010101': [97], '10100111110001010100': [100], '10100
110100001010100': [102, 115], '10110010101100010100': [105], '10110111101101010100': [106], '10100110001001010100':
[107], '10100011111001010100': [108, 113], '10110110101100010100': [112, 161], '10011010001011011110': [117], '0000
0011111001010100': [119], '10100111000001010100': [126], '10101010100111010100': [127], '10100011000111010100': [12
8], '00100011101001010100': [129], '00000110101001010100': [134], '10100010101101000100': [139], '10100010110001010
100': [147], '10101011001011010100': [149], '10111111101011010110': [151], '00100011101101010100': [152], '10110011
101001010100': [153], '00100111101001010100': [154], '10100010001100010100': [156]}
```

4. Function to plot images based on hash key:

In [55]:
```
#Fitting above to KNN to determine accuracy
knn = neighbors.KNeighborsClassifier(n_neighbors=5)
knn.fit(Xtrain, y_train)
y_pred = knn.predict(Xtest)
y_pred
```

Out[55]:
```
array([2, 2, 2, 0, 1, 1, 2, 0, 2, 2, 2, 2, 1, 0, 1, 2, 1, 1, 1, 2, 2, 2,
       2, 0, 0, 2, 0, 1, 1, 2, 2, 1, 2, 1, 2, 2, 0, 1])
```

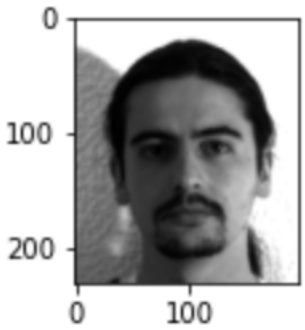
In [56]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

print('Accuracy Score:', accuracy_score(y_test, y_pred))
print('Confusion matrix \n',  confusion_matrix(y_test, y_pred))
print('Classification \n', classification_report(y_test, y_pred))
```

- **Results:**

  Below are the plots based on their respective hash keys:

```
plotImages(images, values[0])
```




```
In [16]: plotImages(images, values[2])
```