10/13/2022

K Nearest Neighbors

Ali Hassan

Contents

K-Nearest Neighbours

KNN is one of the simplest forms of machine learning algorithms mostly used for classification. It classifies the data point on how its neighbour is classified. KNN classifies the new data points based on the similarity measure of the earlier stored data points.

1. KNN Model Representation

The model representation for KNN is the entire training dataset. It is as simple as that. KNN has no model other than storing the entire dataset, so there is no learning required. Efficient implementations can store the data using complex data structures like k-d trees to make look-up and matching of new patterns during prediction efficient. Because the entire training dataset is stored, you may want to think carefully about the consistency of your training data. It might be a good idea to curate it, update it often as new data becomes available and remove erroneous and outlier data.

With the help of KNN algorithms, we can classify a potential voter into various classes like “Will Vote”, “Will not Vote”, “Will Vote to Party 'Congress', “Will Vote to Party 'BJP'.

Other areas in which KNN algorithm can be used include

- Speech Recognition
- Handwriting Detection
- Image Recognition
- Video Recognition.

2. Making Predictions with KNN Algorithm

KNN makes predictions using the training dataset directly. Predictions are made for a new data point by searching through the entire training set for the K most similar instances (the neighbours) and summarizing the output variable for those K instances.

To determine which of the K instances in the training dataset are most similar to a new input a distance measure is used. For real-valued input variables, the most popular distance measure is Euclidean distance. Euclidean distance is calculated as the square root of the sum of the squared differences between a point “a” and point “b” across all input attributes “I”.

$$EuclideanDistance(a, b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

Other popular distance measures include:

- **Hamming Distance**: Calculate the distance between binary vectors.

- **Manhattan Distance**: Calculate the distance between real vectors using the sum of their absolute difference. Also called City Block Distance.
- **Minkowski** Distance: Generalization of Euclidean and Manhattan distance.

There are many other distance measures that can be used, such as Tanimoto, Jaccard, Mahalanobis and cosine distance. We can choose the best distance metric based on the properties of your data.

2.1 KNN for Classification

When KNN is used for classification, the output can be calculated as the class with the highest frequency from the K-most similar instances. Each instance in essence votes for their class and the class with the most votes is taken as the prediction. Class probabilities can be calculated as the normalized frequency of samples that belong to each class in the set of K most similar instances for a new data instance.

KNN works well with a small number of input variables (p), but struggles when the number of inputs is very large. Each input variable can be considered a dimension of a p-dimensional input space. For example, if you had two input variables X1 and X2, the input space would be 2-dimensional. As the number of dimensions increases the volume of the input space increases at an exponential rate. In high dimensions, points that may be similar may have very large distances. All points will be far away from each other and our intuition for distances in simple 2 and 3-dimensional spaces breaks down. This might feel unintuitive at first, but this general problem is called the **Curse of Dimensionality**.

In our case, we will be using Euclidean Distance to find the distance between the corresponding pixels of the test image and the images available in the dataset i.e. **data.mat.**

3. Preparing Data for KNN

Preparing data to find the Euclidean distance between the images may involve following steps.

3.1 Loading Test and Training Data

First step is to load the training data, which In our case is Data.mat file, and Test Image in the program.

Note: If you are using google colab you might need to your drive with Google colab to get rid of runtime data loss problems. To open .mat files in Google Colab one of the following libraries is needed.

- scipy.io
- NumPy
- mat4py

Now lets connect the drive with google collab and link the training data first.

```
import scipy.io as sio
GOOGLE_COLAB = True
```

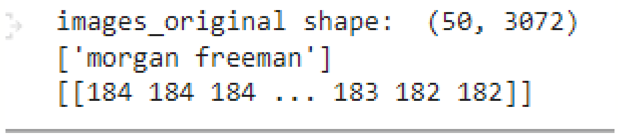
```
path = ""
if GOOGLE_COLAB:
  from google.colab import drive, files
  drive.mount('/content/drive')
  path = "/content/drive/MyDrive/ML"

dataset= path+"/data.mat"
```

Please note that scipy.io is used to load .mat file in python. As the file was available in the folder named ML so exact path is given and it is stored in a variable named “dataset”.

The mat file has 3 matrices. First Matrix named “images” which is of dimension (50,3072) and contains 50 images in the form of 50 rows, each row containing the detail of every pixel of RGB i.e. 32x32x3. Second matric named “C” contains the label i.e. names of the actors or actresses. So next task is to access these two matrices to check if the data.mat file is correctly loaded or not.

For this we will use sio.loadmat function, and will access images and its labels one by one.

```
contents = sio.loadmat(dataset)
originalimages = contents['images']
label = contents ['C']
print ("images_original shape: ", originalimages.shape)
print(label[1,0])
print(originalimages[0:1])
```



```
images_original shape:  (50, 3072)
['morgan freeman']
[[184 184 184 ... 183 182 182]]
```

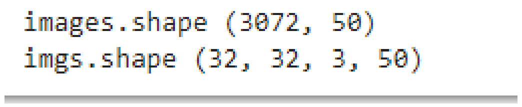
Figure 1 Output Training Data detail

As from the output, its clear that data has been correctly loaded in Google Colab.

3.2 Rescaling and Reshaping Data

Now to display the image we need to take the transpose of the matrix and convert the image into its normal dimension i.e. 32x32. For this purpose we will use np.transpose and np.reshape function of numpy, respectively.

```
import numpy as np
images = np.transpose(originalimages)
print("images.shape", images.shape)
imgs = np.reshape(images, [32,32,3,-1],order="F")
print("imgs.shape", imgs.shape)
```



```
images.shape (3072, 50)
imgs.shape (32, 32, 3, 50)
```

Figure 2 Output, Transposing and Reshaping

From the output its clear that we have done what we wanted to do. Now to display the image we will use pyplot from matplotlib library. Lets display picture number 40 with its label.

```
from matplotlib import pyplot as plt
plt.imshow(imgs[:,:,:,40])
print("imgs[:,:,:,40].shape", imgs[:,:,:,40].shape)
print(label[40,0])
```

```
imgs[:,:,:,40].shape (32, 32, 3)
['fawad khan']
```

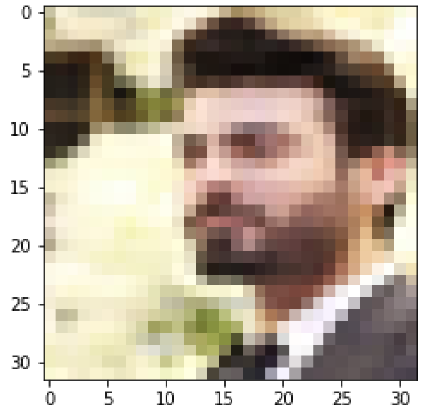


Figure 3 Output Picture No.40 from Data.mat

Now our training data is ready. We will now link and display the test image. Then we will resize it to 32x32 and will make he same (3072x1) matrix out of it to compare it with the training data.

```
import cv2
from math import sqrt
test_image = path + "/pic.jpeg"

img0 = cv2.imread(test_image)
plt.figure()
plt.imshow(img0)

img1=cv2.cvtColor(img0, cv2.COLOR_BGR2RGB)
plt.figure()
plt.imshow(img1)

img2=cv2.resize(img1, (32,32))
plt.figure()
plt.imshow(img2)

img3 = np.reshape(img2, [3072,1], order="F")
print("img3.shape", img3.shape)
```

```
img3.shape (3072, 1)
```

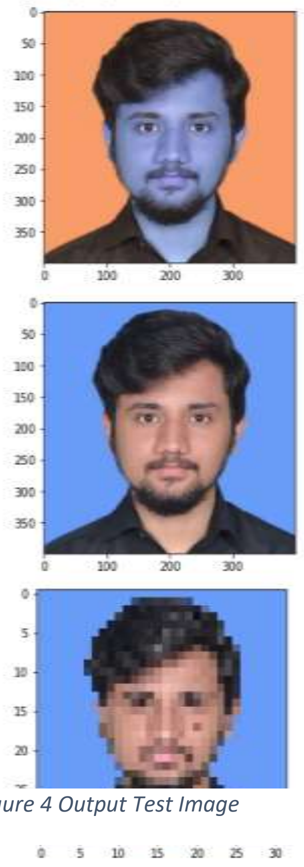


Figure 4 Output Test Image

By default, cv2.imread command read image in BGR so it is converted to RBG via cv2.cvt.Color() function by using cv2.COLOR_BGR2RGB in arguments, to make it compatible with the test data. Its size is reduce to 32x32 and its reshaped to 3072x1 i.e. same as the training data.

3.3 Lower Dimensionality

KNN is suited for lower dimensional data. You can try it on high dimensional data (hundreds or thousands of input variables) but be aware that it may not perform as well as other techniques. KNN can benefit from feature selection that reduces the dimensionality of the input feature space. Therefore in or case data is either given in 32x32 format or it is reduced and rescaled to it.

4. Euclidean Distance

In mathematics, the Euclidean distance between two points in Euclidean space is the length of a line segment between the two points. It can be calculated from the Cartesian coordinates of the points using the Pythagorean theorem, therefore occasionally being called the Pythagorean distance.

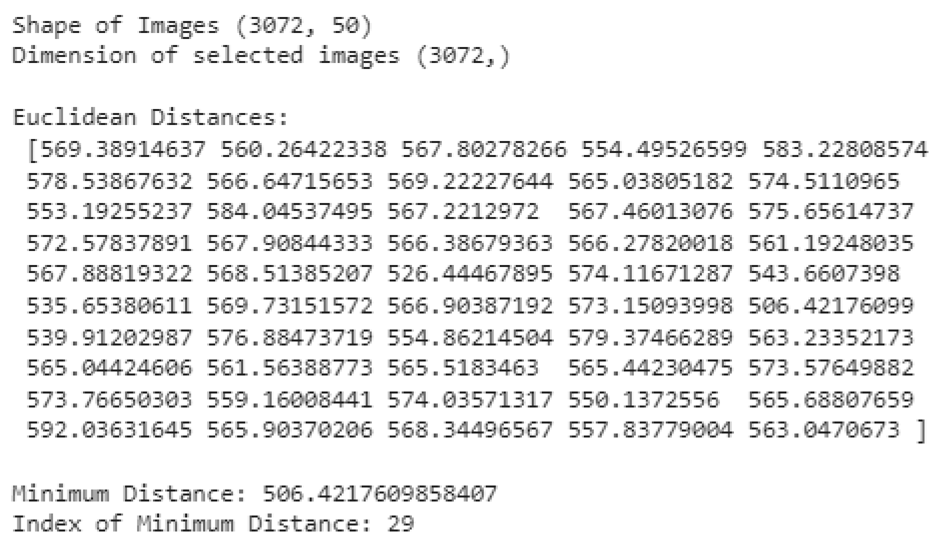
Now we will define a function that can take two arrays or vectors and return the Euclidean distance between them.

```
def euclidean(row1, row2):
  distance = 0.00
  for i in range(len(row1)):
    distance = distance + (row1[i]-row2[i])**2
  return sqrt(distance)
```

```
As the function is now defined. Now lets call this function to
calculate the Euclidean distance between our test image and all the
images in the data set.
```

```
dist = np.zeros(50)
print("Shape of Images",images.shape)
for i in range(50):
  selectedimg = images[0:3072,i]
  dist[i] = euclidean(img3, selectedimg)
print("Dimension of selected images",selectedimg.shape)
print("\nEuclidean Distances:\n", dist)
print("\nMinimum Distance:", min(dist))
mindist = np.argmin(dist)
print("Index of Minimum Distance:", mindist)
```

in the above script min() is used to find the minimum number is dist and np.argmin function is used to give index of the position at which minimum number is present. In the example case the minimum distance is present in index number 29.



```
Shape of Images (3072, 50)
Dimension of selected images (3072,)

Euclidean Distances:
 [569.38914637 560.26422338 567.80278266 554.49526599 583.22808574
 578.53867632 566.64715653 569.22227644 565.03805182 574.5110965
 553.19255237 584.04537495 567.2212972  567.46013076 575.65614737
 572.57837891 567.90844333 566.38679363 566.27820018 561.19248035
 567.88819322 568.51385207 526.44467895 574.11671287 543.6607398
 535.65380611 569.73151572 566.90387192 573.15093998 506.42176099
 539.91202987 576.88473719 554.86214504 579.37466289 563.23352173
 565.04424606 561.56388773 565.5183463  565.44230475 573.57649882
 573.76650303 559.16008441 574.03571317 550.1372556  565.68807659
 592.03631645 565.90370206 568.34496567 557.83779004 563.0470673 ]

Minimum Distance: 506.4217609858407
Index of Minimum Distance: 29
```

Figure 5 Euclidean Distances

Now let's arrange the distances in ascending order by keeping the track of the indices with respect to original distance marix.

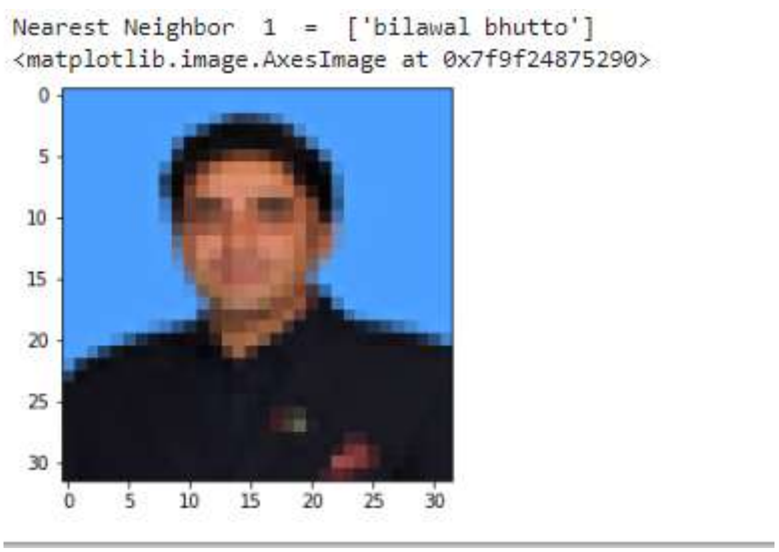
```
dist2 = [0.0]*50

for k in range (0,len(dist)):
  dist2[k]=dist[k]

dist2.sort()
print("dist2",dist2)

indices = []
ind = 0
for x in range (50):
  for y in range(50):
    if dist2[x] == dist[y]:
      ind = y
      indices.append(ind)
      break
print("indices in main distance vector",indices)
```

```
dist2 [506.4217609858407, 526.4446789549686, 535.6538061098792, 539.9120298715338,
543.6607398001073, 550.1372556008182, 553.1925523721375, 554.4952659852021,
554.8621450414508, 557.8377900429479, 559.1600844123265, 560.2642233803618,
561.1924803487659, 561.5638877278345, 563.0470673043241, 563.2335217296641,
565.0380518159817, 565.0442460551209, 565.442304749123, 565.5183462983318,
565.6880765934527, 565.9037020553939, 566.2782001807946, 566.3867936313487,
566.6471565268814, 566.9038719218629, 567.2212972024234, 567.4601307581,
567.8027826631356, 567.8881932211657, 567.9084433251543, 568.3449656678591,
568.5138520739843, 569.2222764439214, 569.389146366525, 569.7315157159554,
572.5783789141885, 573.150939980037, 573.5764988212121, 573.7665030306318,
574.0357131747118, 574.116712872914, 574.5110964985794, 575.656147365769,
576.8847371875945, 578.5386763216441, 579.3746628909483, 583.2280857434765,
584.0453749495839, 592.0363164536446]
indices in main distance vector [29, 22, 25, 30, 24, 43, 10, 3, 32, 48, 41, 1, 19, 36, 49, 34, 8, 35, 38, 37, 44,
46, 18, 17, 6, 27, 12, 13, 2, 20, 16, 47, 21, 7, 0, 26, 15, 28, 39, 40, 42, 23, 9, 14, 31, 5, 33, 4, 11, 45]
```

5. 1NN

By 1NN we mean that, we have to find the first neighbour that is the closest match t our test image. Obviously, this will be the image with the least distance from the test image. We have already found the 1NN, when we found the minimum distance between our test image and the images in the data set. Now we will simply display that image.

```
b=0
print("Nearest Neighbor ",b+1," = ", label[indices[0],0])

plt.figure()
plt.imshow(imgs[:,:,:,indices[0]])
```



```
Nearest Neighbor  1  =  ['bilawal bhutto']
<matplotlib.image.AxesImage at 0x7f9f24875290>
```

Figure 6 Best Match 1NN

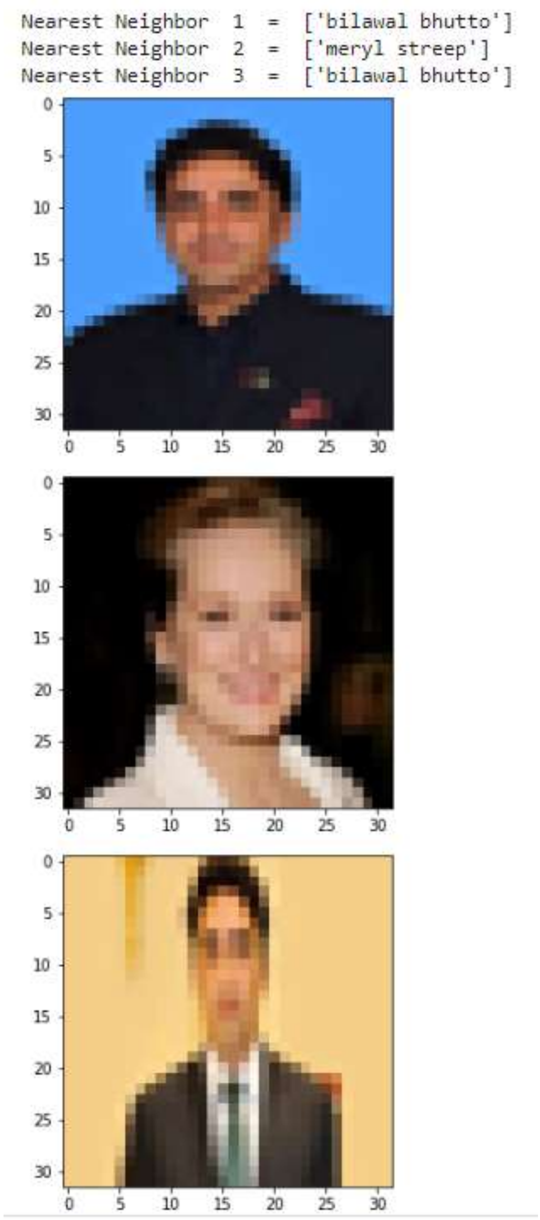
Indices vector already had the index of the least distant image so we just used the number at indices[0] to display the image as well as its label.

6. 3NN and 5NN

To find the 3 and 5 nearest neighbours we will follow the same scheme. As the data is already sorted in ascending order so we just need to display the first 3 and first 5 nearest images respectively.

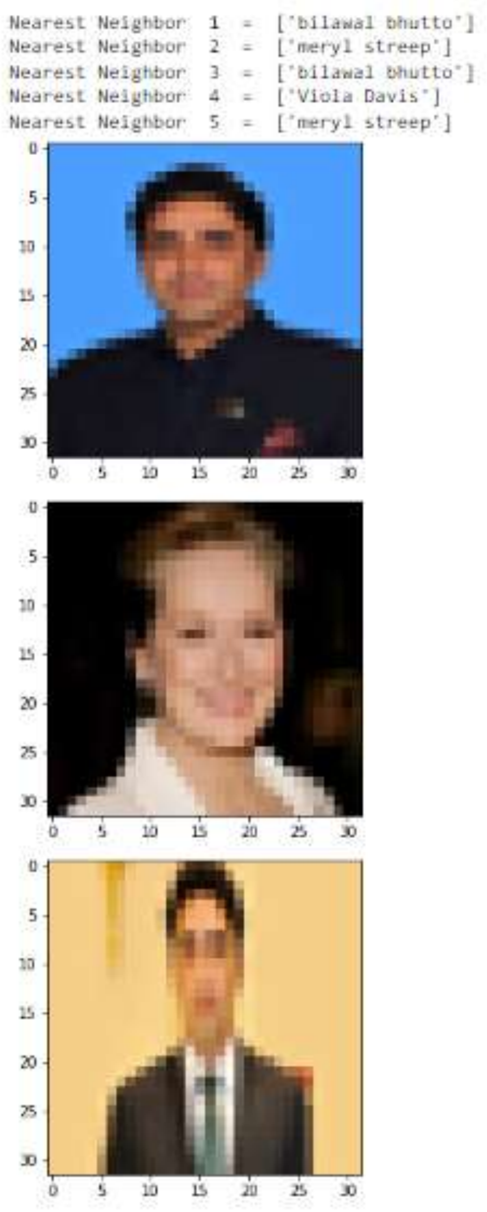
Lets begin with 3NN:

```
for b in range(3):
 print("Nearest Neighbor ",b+1," = ", label[indices[b],0])
 plt.figure()
 plt.imshow(imgs[:,:,:,indices[b]])

now for 5NN
b=0
for b in range(5):
 print("Nearest Neighbor ",b+1," = ", label[indices[b],0])
 plt.figure()
 plt.imshow(imgs[:,:,:,indices[b]])
```



```
Nearest Neighbor  1  =  ['bilawal bhutto']
Nearest Neighbor  2  =  ['meryl streep']
Nearest Neighbor  3  =  ['bilawal bhutto']
```

Figure 7 3NN



```
Nearest Neighbor  1  =  ['bilawal bhutto']
Nearest Neighbor  2  =  ['meryl streep']
Nearest Neighbor  3  =  ['bilawal bhutto']
Nearest Neighbor  4  =  ['Viola Davis']
Nearest Neighbor  5  =  ['meryl streep']
```

Figure 8 5NN

7. Majority Voting and Generic Function

In case of 3 or more nearest neighbours, good practice is to do majority voting to get a covering response. In the previous example of 3NN Bilawal Bhutto was appearing 2 times. So bilawal Bhutto is the best match in case of 3NN same in the case with 5NN. Lets see how we can do the majority voting in programming.

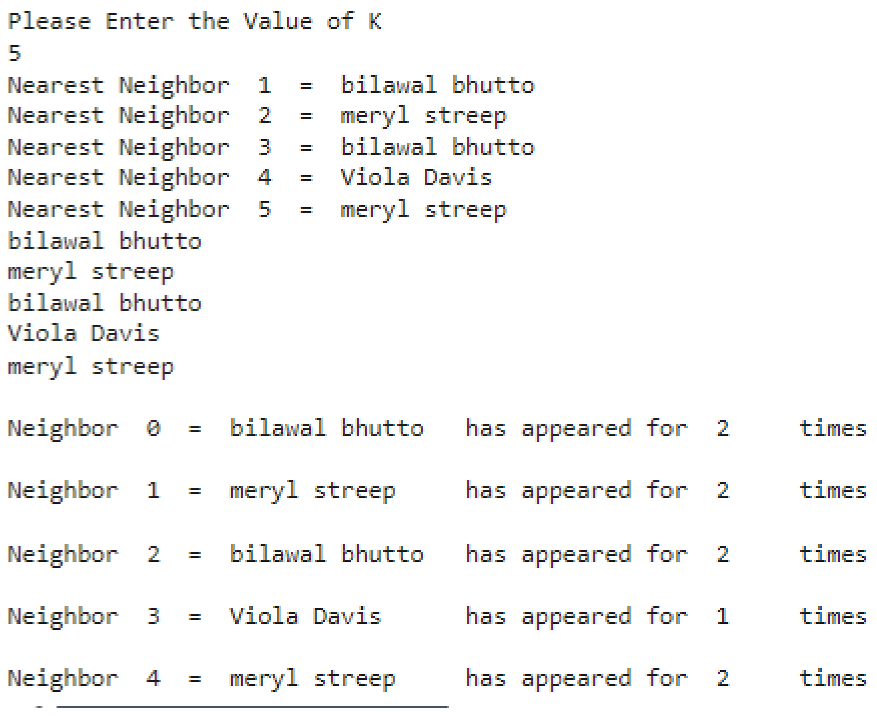
Lets begin by defining a generic function that will ask the user the value of k i.e. no. of nearest neighbours.

```
def user_input():
  v = int(input("Please Enter the Value of K \n"))
  z=[]
  index=[]
  label2=[]
  countt = 0
  for k in range(v):
    index.append(indices[k])
  for b in range (v):
    z.append(index[b])
    label2.append(label[z[b],0][0])
    print("Nearest Neighbor ",b+1," = ", label[z[b],0][0])
    plt.figure()
    plt.imshow(imgs[:,:,:,z[b]])

## M A J O R I T Y  V O T I N G ##
  m_v = []
  value = 0
  for h in range (v):
    print(label2[h])
    for p in range(v):
      if label2[h]==label2[p]:
        countt=countt+1
    m_v.append(countt)
    countt=0

  for h in range (v):
    print("\nNeighbor ",h," = ",label[z[h],0][0], "\t has appeared for
", m_v[h], "\t times")
```

in majority voting section of the above function, labels are matched in the nearest neighbour array and if the match is found “count” variable is unit incremented.

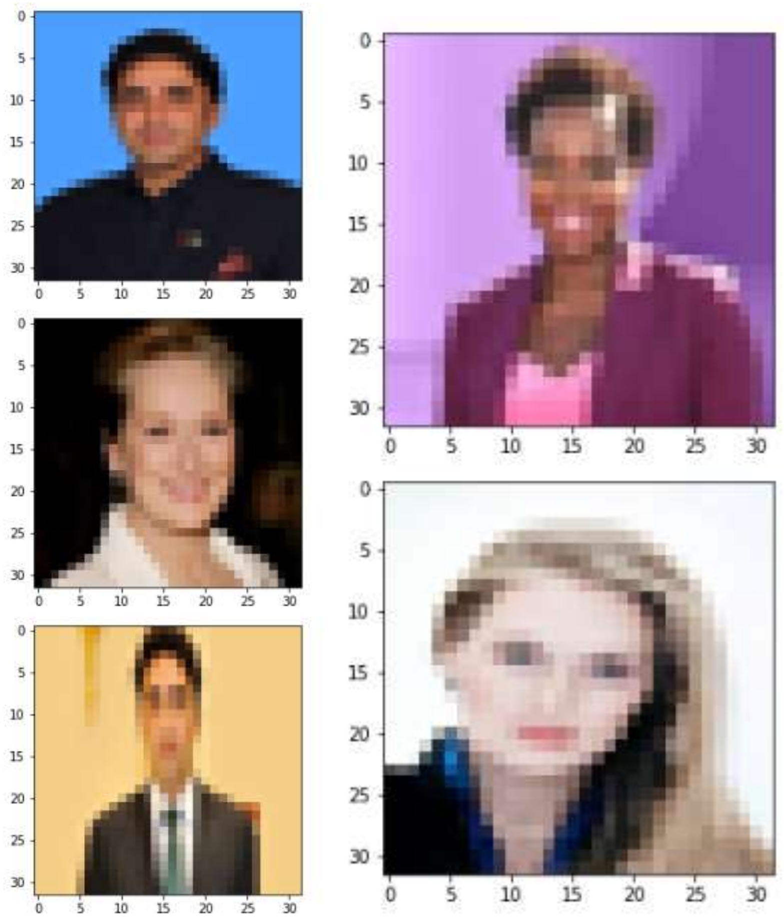
Now lets call this function are check the output for 5NN.

```
user_input()
```



```
Please Enter the Value of K
5
Nearest Neighbor  1  =  bilawal bhutto
Nearest Neighbor  2  =  meryl streep
Nearest Neighbor  3  =  bilawal bhutto
Nearest Neighbor  4  =  Viola Davis
Nearest Neighbor  5  =  meryl streep
bilawal bhutto
meryl streep
bilawal bhutto
Viola Davis
meryl streep

Neighbor  0  =  bilawal bhutto   has appeared for  2     times

Neighbor  1  =  meryl streep     has appeared for  2     times

Neighbor  2  =  bilawal bhutto   has appeared for  2     times

Neighbor  3  =  Viola Davis      has appeared for  1     times

Neighbor  4  =  meryl streep     has appeared for  2     times
```

Figure 9 Majority Voting



8. Pseudocode

Pseudocode has no defined standard, as it is simply a method of writing a human-readable representation of program code. There are many ways to write the pseudocode of the python code. I will follow the following rules.

1. Importing the libraries and its functions will be used in the same way as in actual code
2. Variable and identifiers names will be capitalized
3. Data type of the variables will be ignored as the data itself justifies the type of the variable used
4. Useless and repetitive statements will be merged e.g. if I have plotted 3 picture continuously using the same commands, I will write it in one statement only in pseudocode
5. Standard style will be followed in case of loops.
6. To distinguish it from code it will be typed in italics.

Lets begin writing the pseudocode now

Mounting google drive with google colab.

```
MOUNT drive with colab
IMPORT scipy.io as SIO
SET DATASET to path of data.mat file
```

Importing training data file i.e. data.mat.

```
IIMPORT images in ORIGINALIMAGES
IMPORT c in LABEL
PRINT shape of ORIGINALIMAGES
PRINT shape of LABEL
PRINT first member of LABEL
PRINT first member of ORIGINALIMAGES
IMPORT numpy as NP
SET IMAGES to transpose of ORIGINALIMAGES
SET IMGS to reshaped IMAGES
IMPORT pyplot as PLT from matplotlib
DISPLAY 40th image in IMGS
```

Importing rescaling and reshaping the test image.

```
SET TEST_IMAGE to path of test image
IMPORT cv2
Display TEST_IMAGE
SET IMG1 to rgb converted TEST_IMAGE
Display IMG1
SET IMG2 to resized IMG1
SET IMG3 to reshaped IMG2
```

Defining function to calculate the Euclidean distance.

```
def EUCLIDEAN(ROW1,ROW2)
  SET DISTANCE to 0
  FOR I in range( length of ROW1)
   DISTANCE=DISTANCE+(ROW1[I]-ROW2[I])**2
  RETURN square root of DISTANCE
```

Using the defined function to measure the distances from the test image and all the images in the dataset.

```
FOR I in range(5)
  SET SELECTEDIMG to IMAGES[0:3072,I]
  SET DIST[I] to EUCLIDEAN(IMG3,SELECTEDIMG)
FIND minimum number in DIST
SET MINDIST to arguments of DIST
FOR K in range (0,len(DIST)):
  DIST2[K]=DIST[K]
SORT DIST2
FOR X in range (50):
 FOR Y in range(50):
  IF DIST2[X] == DIST[Y]:
  IND = Y
  APPEND INDECES to IND
  BREAK
```

Display the label and image of first nearest neighbour.

```
DISPLAY image at INDECES [0,0]
PRINT LABEL [INDECES [0,0]]
```

Display the labels and images in case of *3NN and 5NN*

```
 FOR B in range(3):
  PRINT LABEL[INDICES[B],0])
  DISPLAY image at INDICES[B]
```

9. Result and Analysis

As already discussed in section 2.1 KNN works well with a small number of input variables (p), but struggles when the number of inputs is very large. Therefore, the images in our case were either used of a lower dimension or are scaled down. If the results of 1NN analyzed from figure 6, we can see the resulting image actually looks very similar to the test image in figure 4. The good part is, the program manages to find the exact background of the images because the details in the background were not much lost in reshaping and rescaling the test image.

Similarly, in case of 3NN and 5NN, with the help of majority voting, the program was able to find the exact match of the test image, because the result of 1NN was appearing multiple times as scene in figure 9.

The results can be further improved by selecting and training the program on the test data of higher dimension, so the maximum details in the images are kept intact. In high dimensions, points that may be similar may have very large distances. All points will be far away from each other and our intuition for distances in simple 2 and 3-dimensional spaces breaks down.

10. Conclusion

K Nearest Neighbour is a simple algorithm that stores all the available cases and classifies the new data or case based on a similarity measure. It is mostly used to classifies a data point based on how its neighbours are classified. 1NN gives the closest match in the available test data. 3NN and 5NN gives the 3 and 5 nearest neighbours I the test data respectively. Euclidean distance was found between the test images and the images available in the dataset. Results were found satisfactory. One of the possible reasons for the anomaly in the results could be that the dataset has the images of very low dimensions, because of which the program was not able to incorporate the exact details in the images. Results can be improved further if the dataset is increased in quantity or if the individual images in the dataset could have more details and features.

11. References

[1]. https://stackoverflow.com/questions/48570343/how-do-you-make-a-function-in-pseudocode#:~:text=Pseudocode%20has%20no%20defined%20standard,be%20defined%20however%20you%20wish.
[2]. https://student.cs.uwaterloo.ca/~cs231/resources/pseudocode.pdf
[3]. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761?gi=b20e73ca1692
[4]. Delores M. Etter, Jeanine A. Ingber. Engineering Problem Solving With C++
[5]. https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning