**PCA Based Animal Recognition**

**Dataset Investigation:**

Image Details:

Used image resizing to resize the datasets to 25 x 25-pixel images. All the resized images were saved as JPEG extension.

Code:

from PIL import Image

import os, sys

path = "Grape/" # specify folder path

dirs = os.listdir(path)

def resize():

count = 0

for item in dirs:

if os.path.isfile(path+item):

# if imagePath == directory + '.DS\_Store':

# continue

im = Image.open(path+item)

if im.mode in ("RGBA", "P"):

im = im.convert("RGB")

f, e = os.path.splitext(path+item)

imResize = im.resize((25,25), Image.ANTIALIAS)

imResize.save( 'resized\_fruits/' + f + 'resized.jpg', 'JPEG', quality=100)

count+=1

resize()

Used two different datasets to establish a solid investigation of the images. 400 images of each category (for both animal and fruit datasets) are used to extract the principal components. The data is randomized and split in 60-40 ratio for training and testing the classifier (K- Nearest Neighbor)

Fruit Dataset (<https://www.kaggle.com/moltean/fruits>)

Using the above dataset to classify four fruit categories. Namely, lemon, apple, grape and pineapple.

After extracting the Principal Components (3PCs). A 3D scatter plot was used to visualize the fruit images data. (Explanation of the code for principal components is given at the end of this paper)

Grape Data Components in a 3D scatter plot

**A close up of text on a white background

Description automatically generated**

Pineapple Data Components in a 3D scatter plot**A picture containing text

Description automatically generated**

Lemon Data Components in a 3D scatter plot**A close up of a map

Description automatically generated**

Apple Data Components in a 3D scatter plot**A picture containing text

Description automatically generated**

A combined representation of the four categories**A close up of text on a white background

Description automatically generated**

After using KNN classifier (K=3 nearest neighbors), the accuracy with the testing data was found to be **99.9956%**

Animal Dataset (<https://www.kaggle.com/alessiocorrado99/animals10>)

Using the above dataset to classify four animal categories. Namely, dog, elephant, horse and butterfly.

After extracting the Principal Components (3PCs). A 3D scatter plot was used to visualize the animal images data.

Dog Data Components in a 3D scatter plot

**A close up of a piece of paper

Description automatically generated**

Elephant Data Components in a 3D scatter plot

**A close up of text on a white surface

Description automatically generated**

Butterfly Data Components in a 3D scatter plot

**A picture containing text

Description automatically generated**

Horse Data Components in a 3D scatter plotA picture containing text

Description automatically generated

A combined representation of the four categories

**A picture containing text

Description automatically generated**

After using KNN classifier (K=3 nearest neighbors), the accuracy with the testing data was found to be **89.9408%**

**Conclusion:**

As we can clearly see from the scatter plots, the fruit dataset principal components are clearly distinguishable compared to the animal data, which is highly overlapping. The KNN classifier performs better on the fruit dataset.

PCA Implementation (Dimension Reduction)

Image Size: Each image is resized to 25 x 25 pixels. And each image has a RBG value assigned to it. Therefore, an image flattening library is used. First 3 matrices of dimension 25 x 25 are constructed and then each cell of three matrix are added row wise. The resulting matrix dimension is 400 (number of images) x (3 (RBG) \* 25 \* 25).

Since we are working with a very high dimension dataset, PCA is applied to reduce the dimension and work with a set of column vector that can be visualized in a 3D scatter plot. PCA is designed to extract the principal components in a such a way that there is not much loss of information.

Code:

def open\_imgs\_set(path, matrix):

count = 0

for filename in glob.glob(path):

# 3-D array from an image

img\_data\_3d = np.asarray(Image.open(filename))

if img\_data\_3d.shape == (25,25,3):

# From 3-D array to 1-D array: converting single image into 1-D array of size n (int this case n=154587)

img\_data\_1d = img\_data\_3d.ravel()

# Preparing matrix n\_photo\*154587

matrix.append(img\_data\_1d)

if count == 400:

break

count+=1

return matrix

Then PCA is applied to the above matrix with 400 rows and 1875 columns. We want the final column size to be 3.

Code:

def get\_X\_Y\_values(filename,label): # PCA Dimesnsion Reduction for each dataset

pca\_data = []

x = []

x = open\_imgs\_set(filename, x)

pca\_data.append(x)

pca\_data = np.array(pca\_data)

#print(len(pca\_data[0][0,:])) # Extracting a row (20 X 20 X 3)

#print(len(pca\_data[0][:,0])) # Extracting a col (number of images)

row\_length = len(pca\_data[0][0,:]) # 1200

col\_length = len(pca\_data[0][:,0]) # images = 3

# Computing the d dimensional mean vector

mean\_vector = []

for y in range(len(pca\_data[0][:,0])):

mean\_vector.append(np.mean(pca\_data[0][y,:]))

# Computing the scatter matrix

scatter\_matrix = np.zeros((row\_length,row\_length))

for i in range(pca\_data.shape[1]):

scatter\_matrix += (pca\_data[:,i].reshape(row\_length,1) - mean\_vector).dot((pca\_data[:,i].reshape(row\_length,1) - mean\_vector).T)

#print('Scatter Matrix:\n', scatter\_matrix)

# Computing the eigenvectors and eigenvalues from the scatter matrix

eig\_val\_sc, eig\_vec\_sc = np.linalg.eigh(scatter\_matrix)

# Getting eigen pairs and sorting them

eig\_pairs = [(np.abs(eig\_val\_sc[i]), eig\_vec\_sc[:,i]) for i in range(len(eig\_val\_sc))]

eig\_pairs.sort(key=lambda x: x[0], reverse=True)

# Choosing K eigen vectors with largest values (3PC's)

matrix\_w = np.hstack((eig\_pairs[0][1].reshape(row\_length,1), eig\_pairs[1][1].reshape(row\_length,1),eig\_pairs[2][1].reshape(row\_length,1)))

# Transforming the samples onto a new subspace

#print(matrix\_w.shape)

transformed = matrix\_w.T.dot(pca\_data[0].transpose())

data = transformed.transpose()

dataset = pd.DataFrame({'X':data[:,0],'Y':data[:,1],'Z':data[:,2],'Class':label})

return dataset

The steps for PCA are explained in the code comments.

Classification Implementation (KNN)

After a dimension reduction step, the test images need to be classified.

Code:

# Using KNN for clasification

import math

# Cartesian distance calculator function

def euclideanDistance(item1, item2, length):

cal\_dist = 0

for x in range(length):

cal\_dist += pow((item1[x] - item2[x]), 2)

return math.sqrt(cal\_dist)

# Function to check the nearest points from the given input

def checkNeighbors(trainingData, test, k):

distance\_measure =[]

length = len(test)

# calling the distance calculator function to measure each distance and add to a list

for x in range(len(trainingData)):

dist = euclideanDistance(test, trainingData[x], length)

distance\_measure.append(( dist, trainingData[x]))

sort\_arr = []

for x in range(len(distance\_measure)):

sort\_arr.append(distance\_measure[x][0])

sort\_arr.sort()

sort\_arr = (sort\_arr[::-1][:k])

distance\_measure = sorted(distance\_measure, key=lambda x: x[0])

neighbors =[]

for x in range(k):

neighbors.append(distance\_measure[x][1])

return neighbors

# Function to see the majority class near the point and predict the class

def determineClass(neighbors):

classMajority = {}

# Determining the classes majority nearest to the given data point

for x in range(len(neighbors)):

classification = neighbors[x][-1]

if classification in classMajority:

classMajority[classification] +=1

else:

classMajority[classification] = 1

lis = []

for x in classMajority:

lis.append(x)

xis= []

for x in classMajority.values():

xis.append(x)

max = -1

counter = 0

max\_counter=0

for y in xis:

counter +=1

if(y>max):

max =y

max\_counter = counter

num = ''

for x in range(len(lis)):

if x == max\_counter-1:

num = lis[x]

return num

KNN\_arr = [] # Classification array for KNN

for x in final\_df\_train.values:

KNN\_arr.append(x)

The classifier is Validated on the testing data.

Code:

# Testing the classifier

correct = 0

total = 0

for x in final\_df\_test.values:

result = checkNeighbors(KNN\_arr,x[0:3],3)

result\_final = determineClass(result)

if result\_final == x[3]:

correct +=1

total+=1

print(correct/total)

Experience:

Working with an image classification is a different game ball, compared to a clean and structured data. The information is pixels are extracted and flattened. The flattened pixel set is used as information about each pixel. Image quality and pixel density matrix plays a huge role in determining the accuracy of the classifier. Since, the animal data used was very ambiguous. The animal images also contain backgrounds. The aforementioned also adds to the discrepancy in the classification and undermines the performance of the classifier used.