PCA-KNN based fruit image recognition

Name: Ashwin Babu

Dataset: https://www.kaggle.com/moltean/fruits/version/44?almost-there=true&token=CfDJ8LdUzqlsSWBPr4Ce3rb9VL94NDb4cA1DaPdZIO02viMgZayoqaDonA1njtJgjTUWCIBe5VErc_Gk6qEfOsCtUoelQgpmoxhlA3_dUz338NuQTnhcwylzBoMgDEbIZr1uAyA8RTH7Ifd2p-HJB42HcnNx6KsqxgwWyAjf3qC3H-Kssn06d_fio4J8ch4Z_K4rA19gzoMEt7X0D03ZqbSOrwAbZWILYVpXuWntyEGYxEiXh4v-6cpOhtfeEKWhlHNjmgTqEcnYgQyHAhONBGqfZUmVOJlnWPRx5YQDwSO6IRgWGYU1WGaezKcg89VWRa2qUmXtC7-lw1vEHhbmSsG0xvE9aIr4WyBAkaAU8M1K5Ejuiygnhj9j1nf8ipwAG4_f50t5OB2hT0uNs2TjmXFhsuWIB1Ar1cP0QQd5j_fB57FUt33AqnHyS0aPnSOwjRE9OTIfRH_rC62G92Qo_RjeECAEu7qU2eJX27L6OrCVkj5lidik5AlIZXt-fqW3wY7gPvYglMxY55ie0Y2IQgs6QfIgY5TV-xem2q02DelcqVWzlspFGsm2OD824fI_qXxWP0U1kcfCi-5-9xNBGwWnsjA#

Used Dataset of four different fruits namely: `{0: 'Apple Braeburn', 1: 'Avocado', 2: 'Banana', 3: 'Dates'}`

Training Data consists of total 1899 images, whereas testing data consists of total 639 images.

Step-1 Loading the Training Data and extracting the labels of each:

Used open CV (cv2) to load the image data and extract the labels of each image, resized the image to 28*28. Here is the code.

In [16]:
```
# Training Data set
fruit_images = []
labels = []
for fruit_dir_path in glob.glob("/Users/ashwinbabu/Downloads/fruits-proj/Training/*"):
    # Extracting the labels from the folder name
    fruit_label = fruit_dir_path.split("/")[-1]
    print(fruit_label)
    for image_path in glob.glob(os.path.join(fruit_dir_path, "*.jpg")):
        # Loads a color image
        image = cv2.imread(image_path, cv2.IMREAD_COLOR)
        # Resize the image
        image = cv2.resize(image, (28, 28))
        # Changing the color space
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        fruit_images.append(image)
        labels.append(fruit_label)
fruit_images = np.array(fruit_images)
labels = np.array(labels)
print('Total Training data images: ',len(fruit_images))
```

```
Avocado
Dates
Apple Braeburn
Banana
Total Training data images:  1899
```

In [3]:
```
label_to_id_dict = {v:i for i,v in enumerate(np.unique(labels))}
id_to_label_dict = {v: k for k, v in label_to_id_dict.items()}
print(id_to_label_dict)
label_ids = np.array([label_to_id_dict[x] for x in labels])
```

```
{0: 'Apple Braeburn', 1: 'Avocado', 2: 'Banana', 3: 'Dates'}
```

Step-2 Implementing PCA as a function: First Flatten the image data to 1899 by (28*28*3), then standardize the data, find co-variance matrix, find eigen value, eigen vector and lastly plotted the proportion of variance graph to decide how many Principal components to pick, in my case I pic ked 2 principal components, transform the data to 2-D using the eigen vector and eigen value. He ere is the code.

In [6]:

```python
# PCA implemtation
def pca(data):
    # First Flattening the image data to 1899 by 2352(28*28*3)
    image_flattened = ([i.flatten() for i in data])
    image_flattened = np.asarray(image_flattened,dtype=float)

    # Standardizing the data by calculating the mean and standard deviation
    # Subtracting with the mean to center the data and dividing by the standard deviation
    for x in range(len(image_flattened[0])):
        mean = np.mean(image_flattened[:,x])
        std = np.std(image_flattened[:,x])
        for y in range(len(image_flattened)):
            if std != 0.0:

                image_flattened[y][x] = (image_flattened[y][x] - mean)/std
            else:
                image_flattened[y][x] = (image_flattened[y][x]- mean)



    # Finding the co-variance matrix
    co_var = np.cov(image_flattened.T)


    # Computing the eigen-value and eigen-vector from co-variance matrix
    eig_val_cov, eig_vec_cov = np.linalg.eig(co_var)



    # Make a list of (eigenvalue, eigenvector) tuples
    eig_pairs = [(np.abs(eig_val_cov[i]), eig_vec_cov[:,i]) for i in range(len(eig_val_cov))]

    # Sort the (eigenvalue, eigenvector) tuples from high to low
    eig_pairs.sort(key=lambda x: x[0], reverse=True)


    # Taking the top 2 features (components)
    matrix_w = np.hstack((eig_pairs[0][1].reshape(2352,1), eig_pairs[1][1].reshape(2352,1)))

    # Training data converted to 2-D space
    transformed = matrix_w.T.dot(image_flattened.T)

    # print(len(transformed[0]))
    # print(transformed.T)

    final_data = transformed.T
    final_data = final_data.real
    final_data[:,1] = np.multiply(final_data[:,1],-1)

    """
     Plottling the propotion of variance graph and keeping the threshold 40% and adding features until 40% of total
    variability which is PC1 and PC2 together in this case (45%)

    """
    eig_val_cov = np.real(eig_val_cov)
    eig_val_sorted = np.sort(eig_val_cov)[::-1]
    eig_val = []
    total = np.sum(eig_val_sorted)

    for x in range(len(eig_val_sorted)):
        temp = []
        if eig_val_cov[x] !=0:
            temp.append(eig_val_cov[x]/(total))
            temp.append(x+1)
            eig_val.append(temp)

    eig_val = np.asarray(eig_val)

    for x in range(len(eig_val)):
        plt.plot(eig_val[x][1],eig_val[x][0],'ro-',linewidth=2)
    plt.title("The proportion of variance plot")
    plt.xlabel('Principal Component')
    plt.ylabel('Eigenvalue')
    plt.show()


    return final_data
```

I performed PCA using the sklearn library on my data to check if gives the same answer like my own function implementation of PCA and here are the results:

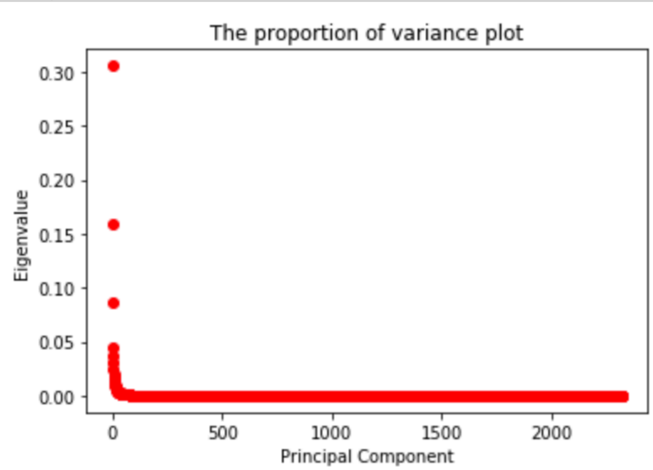
Using my function:

```
[[-13.79713764  -5.44450968]
 [-18.24821148  -7.06634674]
 [-32.94622328  -5.69887025]
 ...
 [ 35.48930194 -11.50028708]
 [ 20.2922983  -18.09956562]
 [ 44.32998711  -9.79990508]]
```

Using the sklearn library:

```
[[-13.79713764  -5.44450968]
 [-18.24821148  -7.06634674]
 [-32.94622328  -5.69887025]
 ...
 [ 35.48930194 -11.50028708]
 [ 20.2922983  -18.09956562]
 [ 44.32998711  -9.79990508]]
```

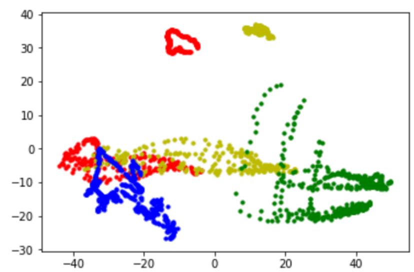
The scree plot (proportion of variance) looks like this:



The transformed Training data looks like this:

In [9]:

```
# graphing the new 2-d Training Data
for x in range((len(train))):
    if train[x][-1] == 0.0:
        plt.scatter(train[x][0], train[x][1], s=10, c='b', marker="o", label='first')
    elif train[x][-1] == 1.0:
        plt.scatter(train[x][0], train[x][1], s=10, c='r', marker="o", label='second')
    elif train[x][-1] == 2.0:
        plt.scatter(train[x][0], train[x][1], s=10, c='g', marker="o", label='third')
    else:
        plt.scatter(train[x][0], train[x][1], s=10, c='y', marker="o", label='fourth')
plt.show()
```



Step-3 Load the Testing data, resize to 28* 28 and perform PCA on it as well. The code looks lik e this.

In [10]:

```
#Testing Data set
fruit_test_images = []
labels_test = []
for fruit_dir in glob.glob("/Users/ashwinbabu/Downloads/fruits-proj/Test/*"):
    fruit_label_test = fruit_dir.split("/")[-1]
    print(fruit_label_test)
    for image_path_t in glob.glob(os.path.join(fruit_dir, "*.jpg")):
        image = cv2.imread(image_path_t, cv2.IMREAD_COLOR)

        image = cv2.resize(image, (28, 28))
        image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

        fruit_test_images.append(image)
        labels_test.append(fruit_label_test)
fruit_test_images = np.array(fruit_test_images)
labels_test = np.array(labels_test)
print(len(fruit_test_images))

```

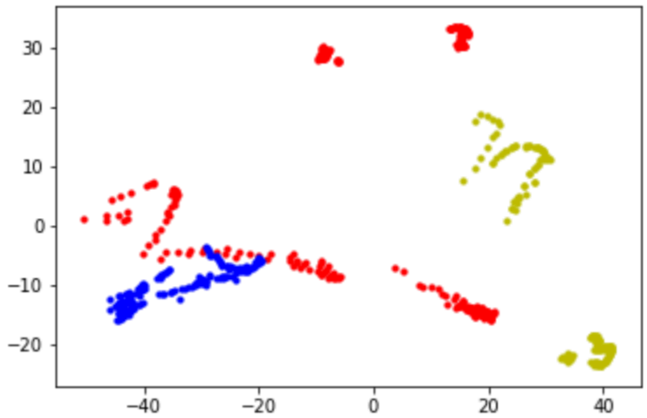
```
Avocado
Dates
Apple Braeburn
Banana
639
```

In [11]:

```
label_to_id_dict_test = {v:i for i,v in enumerate(np.unique(labels_test))}
id_to_label_dict_test = {v: k for k, v in label_to_id_dict_test.items()}
print(id_to_label_dict_test)
label_test_ids = np.array([label_to_id_dict_test[x] for x in labels_test])
```

```
{0: 'Apple Braeburn', 1: 'Avocado', 2: 'Banana', 3: 'Dates'}
```

The testing data after performing PCA looks like this in 2-D



Step-4 KNN implementation

Find K-nearest data points (using Euclidean distance) and take a majority vote to predict the labe

l.

```
# K-Nearest Neighbours Implementation
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)


def checkNeighbors(trainingData, test, k):
    distance_measure =[]
    length = len(test)-1
    for x in range(len(trainingData)):
        dist = euclideanDistance(test, trainingData[x], length)
        distance_measure.append(( dist, trainingData[x]))
    distance_measure.sort(key=operator.itemgetter(0))

    neighbors =[]
    for x in range(k):
        neighbors.append(distance_measure[x][1])

    return neighbors

def determineClass(neighbors):
    classMajority = {}
    for x in range(len(neighbors)):
        classification = str(neighbors[x][-1])

        if classification in classMajority:
            classMajority[classification] +=1
```

```
        else:
            classMajority[classification] = 1

    sortedMajority = max(classMajority.items(), key=operator.itemgetter(1))[0]
    return sortedMajority[0][0]


def accuracy(test, prediction):
    identified=0

    for x in range(len(test)):


        b = float(prediction[x])

        if test[x][-1] == b:
            identified +=1
#     print(identified)
    return (identified/float(len(test))) * 100.0
```

Main function to perform KNN and output the results:

K=3

In [15]:

```
# calling the functions of KNN --> predicting labels --> accuracy
def main():
    # New data that has been conerted to 2-D space
    trainingData = train
    testData = test
    print('Training Data: ' + repr(len(trainingData)))
    print('Test Data: ' + repr(len(testData)))
    # Predicting the class
    predictions = []
    k=input('Enter value for k: ')
    k = int(k)

    for x in range(len(testData)):
        neighbors = checkNeighbors(trainingData, testData[x], k)
        result = determineClass(neighbors)
        predictions.append(result)
#         print('--> predicted=' + repr(result) + '-->actual=' + repr(testData[x][-1]))
    accuracy_1 = accuracy(testData, predictions)
    print('Accuracy:' + repr(accuracy_1) + '%')

main()
```

```
Training Data: 1899
Test Data: 639
Enter value for k: 3
Accuracy:73.55242566510172%
```

k=5

In [17]:

```python
# calling the functions of KNN --> predicting labels --> accuracy
def main():
    # New data that has been conerted to 2-D space
    trainingData = train
    testData = test
    print('Training Data: ' + repr(len(trainingData)))
    print('Test Data: ' + repr(len(testData)))
    # Predicting the class
    predictions = []
    k=input('Enter value for k: ')
    k = int(k)

    for x in range(len(testData)):
        neighbors = checkNeighbors(trainingData, testData[x], k)
        result = determineClass(neighbors)
        predictions.append(result)
#         print('--> predicted=' + repr(result) + '-->actual=' + repr(testData[x][-1]))
    accuracy_1 = accuracy(testData, predictions)
    print('Accuracy:' + repr(accuracy_1) + '%')

main()
```

```
Training Data: 1899
Test Data: 639
Enter value for k: 5
Accuracy:74.80438184663537%
```

K=1

In [18]:

```python
# calling the functions of KNN --> predicting labels --> accuracy
def main():
    # New data that has been conerted to 2-D space
    trainingData = train
    testData = test
    print('Training Data: ' + repr(len(trainingData)))
    print('Test Data: ' + repr(len(testData)))
    # Predicting the class
    predictions = []
    k=input('Enter value for k: ')
    k = int(k)

    for x in range(len(testData)):
        neighbors = checkNeighbors(trainingData, testData[x], k)
        result = determineClass(neighbors)
        predictions.append(result)
#         print('--> predicted=' + repr(result) + '-->actual=' + repr(testData[x][-1]))
    accuracy_1 = accuracy(testData, predictions)
    print('Accuracy:' + repr(accuracy_1) + '%')

main()
```

```
Training Data: 1899
Test Data: 639
Enter value for k: 1
Accuracy:68.38810641627543%
```

Outcomes of this project:

This project helped me learn how to handle image data, I had no prior experience handling image data. I learned how to use cv2 to load the image data. Dimension reduction is a very important topic in machine learning (without much loss of information). Learned how to reduce dimension, how many components to choose, meaning of scree plot, importance of variance, co-variance matrix. The importance of eigen-value, eigen-vector and numpy operations.