Capstone Project Report

Name: Sumit Patel

Course: AI and ML (Batch – AUG 2020)

Duration: 10 months

K-Means Clustering: Image Segmentation

Problem Statement:

Factor analysis is a useful technique to find latent factors that can potentially describe multiple attributes, which is sometimes very useful for dimensionality reduction. Use the Airline Passenger Satisfaction dataset to perform factor analysis. (Use only the columns that represent the ratings given by the passengers, only 14 columns). Choose the best features possible that helps in dimensionality reduction, without much loss in information.

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-externalcommand/. 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 then run below commands in command prompt/terminal to install these packages:

pip install numpy

pip install matplotlib

If you have chosen to install anaconda then run below commands in anaconda prompt to install these packages:

conda install -c anaconda numpy

conda install -c anaconda matplotlib

Dataset used:

Any bright coloured image

Importing the libraries and loading dataset.

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

Assign random labels to individual pixels ¶

```
K = 5 ## K is the number of clusters that we want to create

for i in range(label_arr.shape[0]):
    for j in range(label_arr.shape[1]):
        label_arr[i,j] = np.random.choice(K)
print(label_arr)
print(label_arr.shape)
```

```
[[2. 3. 3. ... 3. 4. 2.]
 [4. 0. 3. ... 2. 2. 0.]
 [4. 3. 1. ... 2. 1. 0.]
 ...
 [0. 4. 4. ... 2. 3. 0.]
 [1. 3. 0. ... 3. 4. 4.]
 [0. 1. 4. ... 4. 4. 0.]]
(800, 800)
```

```
def dist_p(vec1,vec2,p):  # Generalised Distance Formula
    L = len(vec1)
    s1 = 0
    for l in range(L):
        diff = np.abs(vec2[l]-vec1[l])
        s1 = s1 + diff**p
    distance = s1**(1/p)
    return(distance)
```

Define function to generate the initial mean values from initial labels

```
def init_mean(K,img_arr,label_arr):
    mean_ls = [] ## List containing mean values of the clusters
    pixel_ls = [[] for k in range(K)] ## Create list of empty lists to store pixels belonging to a certain cluster

    for i in range(label_arr.shape[0]):
        for j in range(label_arr.shape[1]):
            for k in range(K):
                if label_arr[i,j] == k:  ## if the label of the pixel at location [i,j] is 'k'
                    pixel_ls[k].append(np.ravel(img_arr[i,j,:])) ## Fill the kth empty list with this pixel value

    for k in range(K):
        pixel_mat = np.matrix(pixel_ls[k])
        mean_k = np.mean(pixel_mat,axis=0)
        mean_ls.append(np.ravel(mean_k))
    return(mean_ls)
```

Update labels by comparing distances with previous mean values and generate new labels

```
def label_update(prev_mean,img_arr,label_arr,p):
    for i in range(img_arr.shape[0]):
        for j in range(img_arr.shape[1]):
            dist_ls = []
            for k in range(len(prev_mean)):
                dist = dist_p(img_arr[i,j,:],prev_mean[k],p) ## Calculate the distance of the pixel at [i,j] with the kth mean
                dist_ls.append(dist) ## Put the distance values in a list
            dist_arr = np.array(dist_ls) ## Convert it to a NumPy array
            new_label = np.argmin(dist_arr) ##The new_label of the point is the one which is closest to the pixel at [i,j]
            label_arr[i,j] = new_label ## Set the new label
    return(label_arr)
```

Generate new mean values from the updated labels ¶

```
def mean_from_label(K,prev_mean,img_arr,label_arr):
    pixel_ls = [[] for k in range(K)]  ## Create list of empty lists to store pixels belonging to a certain cluster

    for i in range(label_arr.shape[0]):
        for j in range(label_arr.shape[1]):
            for k in range(K):
                if label_arr[i,j] == k: ## if the label of the pixel at location [i,j] is 'k'
                    pixel_ls[k].append(np.ravel(img_arr[i,j,:])) ## Fill the kth empty list with this pixel value

    for k in range(K):
        if len(pixel_ls[k]) != 0:  ## Only update the means of those clusters which has received at least one new point, else ret
            pixel_mat = np.matrix(pixel_ls[k])
            mean_k = np.mean(pixel_mat,axis=0)
            prev_mean[k] = np.ravel(mean_k)
    new_mean = prev_mean
    return(new_mean)
```

Run the K-Means Algorithm and obtain the final labels and means

```
def KMeans(img_arr,label_arr,K,p,maxIter):
    mean_old = init_mean(K,img_arr,label_arr)
    for t in range(maxIter):
        new_label_arr = label_update(mean_old,img_arr,label_arr,p)
        mean_new = mean_from_label(K,mean_old,img_arr,new_label_arr)
        print("The mean obtained at {}th iteration is {}\n".format(t,mean_new))
        label_arr = new_label_arr ## Update the label array
        mean_old = mean_new ## Update the mean values
    return(mean_new,label_arr)
```

Use the finally obtained mean and labels to segment the image

```
def segmentImage(image_arr,label_arr,mean_ls):
    seg_image = np.zeros((image_arr.shape[0],image_arr.shape[1],image_arr.shape[2]))
    for i in range(seg_image.shape[0]):
        for j in range(seg_image.shape[1]):
            k = label_arr[i,j]
            seg_image[i,j,:] = mean_ls[int(k)]
    seg_image = seg_image.astype(np.uint8)
    plt.imshow(seg_image)
```

```
segmentImage(image_arr,label_final,mean_final)
```