INFO 371 AUTUMN 2023
PROBLEM SET 4

Due 12/3/2023 11:59pm

This problem set is on principal component analysis (PCA) and clustering. Please turn in your jupyter notebook that contains the code and answers and its html file. Name your files LASTNAME-FIRSTNAME-PS4.[filetype]. This problem set will make up 15% of your overall grade.

1. Principal Component Analysis (5 points total)

Just when you thought you were *finally* done with the Boston Housing Data Set, it has come back for a few more minutes.

```
df_data  = pd.read_csv('./housing_data.csv')
all_features = ['crim', 'zn', 'indus',  'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax', 'ptratio', 'b', 'lstat']
X = df_data[all_features]
y = df_data["medv"]
```

1.1 (1 point) Scatterplot

Use matplotlib to create a scatter plot that shows the relationship between the median value of the home (y-axis) and the per-capita crime rate (x-axis). Properly label your axes, and make sure that your graphic looks polished and professional. (0.5 point)

Use the Linear Regression class from sklearn or the OLS class from SciPy to regress median housing price on per-capita crime rate by town. Use a training set consisting of 66% of your instances to fit your regression model. Report the RMSE on the training and test set. (0.5 point)

1.2 (1 point) Features and plots

Choose 8 out of the 14 features of the housing data that you think are most interesting to study. Create an 8x8 grid of scatterplots showing how each of these variables relate to each other.

```
plt.figure(figsize=(16,16))
for i in range(8):
   for j in range(8):
      plt.subplot(8, 8, i * 8 + j + 1)
      # insert your code

plt.tight_layout()
```

1.3 (1 point) PCA to the rescue

Run principal component analysis using PCA from sklearn to find the first 12 principal components of your data. Use only your training set (66% of the data), and use all features *except* the median housing price. Create a figure showing how the amount of variance explained by your components increases as you increase the number of components from 1 to 12.

1.4 (1 point) Visualizing the components

Create a 1X3 grid containing three scatter plots of your training data:

1.4.a PCA 1 vs. PCA 2 (you will need to use fit_transform to project your training data onto these two components) (0.33 point)

1.4.b PCA 1 (x-axis) vs. median housing value (0.33 point)

1.4.c PCA 2 vs. median housing value (0.33 point)

____

pca = PCA(n_components = 2)

# Enter your code

____

1.5 (1 point) Regression and PCA

Using just the first Principal Component learned from the training data, project all of your data (including the test data) onto this 1-dimensional subspace using the fit_transform function. Now, using your training data, fit a regression of median housing price on this first principal component of your data. Report the RMSE of this regression for both the training and test set. How do these results compare to your results from 1.1? Interpret your results!

pca = PCA(n_components = 1)

# Enter your code

Enter your observations

2 Cluster images (5 points total)

Your next task is to cluster visualization. Download the *visualizations.zip* file from canvas and decompress it on your computer. Note that it is a compressed archive and cannot be read on the fly, unlike compressed csv-s. It contains ~1000 visualizations scraped from the web and converted to 711×399 pixels by Prof. Leilani Battle (the full dataset can be downloaded from Prof. Battle's webpage). The archive contains ~25MB of images when decompressed.

Your task is to load the images and partition these into different clusters by k-means clustering. You will use different values of k and comment how the resulting clusters look like. You may code your k-means algorithm or use sklearn's implementation.

2.1 Decompress and explore (1 point)

a) (0.5 point) Decompress and inspect the images to get an idea about what we are analyzing. Use whatever image viewer you like. What do you see?
b) (0.5 point) First read all the file names (these are hashcodes) into your code. You can get a list of files with

```
import os

os.listdir(folder)
```

How many images are there? Compute the answer by coding and provide the script.

2.2 Load and cluster (1 points)

2.2.a (0.5 points) Load images. We strongly recommend you start with a small sample of images (e.g., 100 images). After everything runs properly, then you can increase the sample size –note that clustering all the data points might be slow (5-10min).

You can read images with code that is similar to the following snippet:

```
from matplotlib.image import imread

# images = np.random.choice(images, 100) # smaller sample for speed
im1 = imread(os.path.join(imagedir, images[0]))
imshape = im1.shape
X = np.empty(shape=(len(images),
imshape[0]*imshape[1]*imshape[2]))
for i, img in enumerate(images):
    pixels = imread(os.path.join(imagedir, img))
    X[i,:] = pixels.ravel()
```

You can increase the number of samples if the code is running fast enough.

2.2.b (0.5 point) select k and run clustering. This may be slow so don’t start with the full set. Using a subset of 100 images takes 12 seconds on an i7-3770 desktop.

Warning: scaling may not be linear, in particular when you run out of memory!

2.3 Analyze the clusters (3 points)

2.3.1 (1 points) group the clusters: this will create an array of cluster labels, one label for each cluster.

2.3.2 (1 points) inspect the images in each cluster. Describe what you see and how clusters differ from each other.

You can plot a few images using this function:

```
## plot images from cluster `cl’
## use this function to inspect what kind of images
## there are in each cluster
 def plotcluster(cl, c):
       """
       plot 6 random images from given cluster ’c’
inputs:
cl: list of clusters for images (Nx1 vector) c: images from
which cluster to plot (0..k-1)
"""
          plt.suptitle(’cluster ’ + str(c))
          Xcl = X[cl == c]
          inds = np.random.choice(len(Xcl), 6)
          for i, ind in enumerate(inds):
                 ax = plt.subplot(1,2, i+2)
```

.

```
                array =
                Xcl[ind,:].reshape(imshape)
                ax.imshow(array)
        plt.suptitle("Cluster " + str(c))
```

Please revise the code to make the plot better.

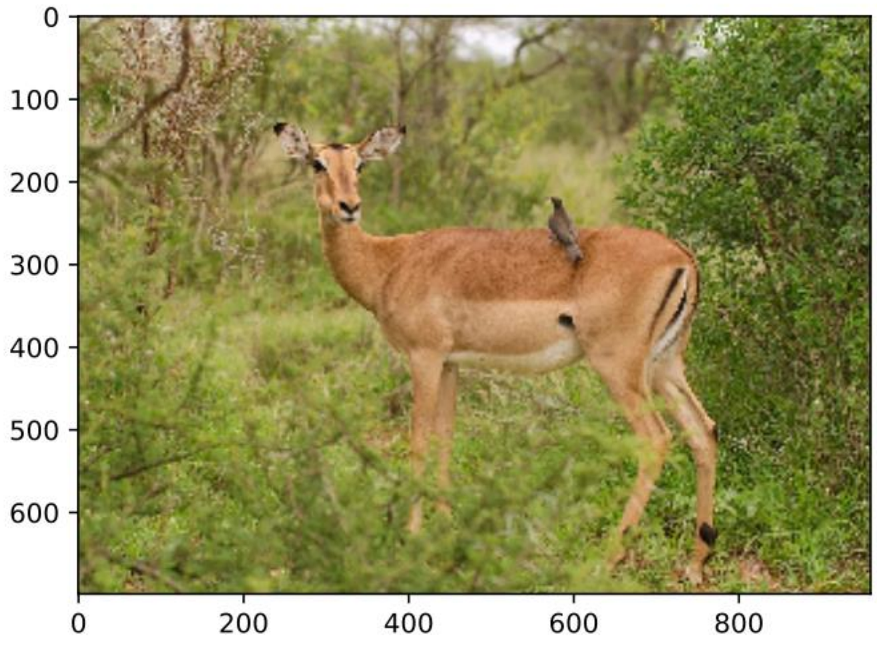
2.3.3 (1 point) Try again with a few different ks and increase sample size. What are the outputs obtained from using different ks? What do the clusters contain? Explain.

3 Images: Reduce the number of colors (5 points)

Finally, let's play a bit more with clustering and images. This time instead of clustering images, we will cluster the colors *within* images. Clustering can be used to reduce the number of colors, e.g. convert a color image to a two-color image.

3.1. (1 point) Get a color image that you choose, load it as a matrix, and display it using matplotlib. You may use the code along these lines:

```python
import matplotlib.pyplot as plt
from matplotlib.image import imread

pixels = imread("img/impala.jpg")
ax = plt.axes()
ax.set_aspect("equal")
_ = plt.imshow(pixels)
_ = plt.show()
```



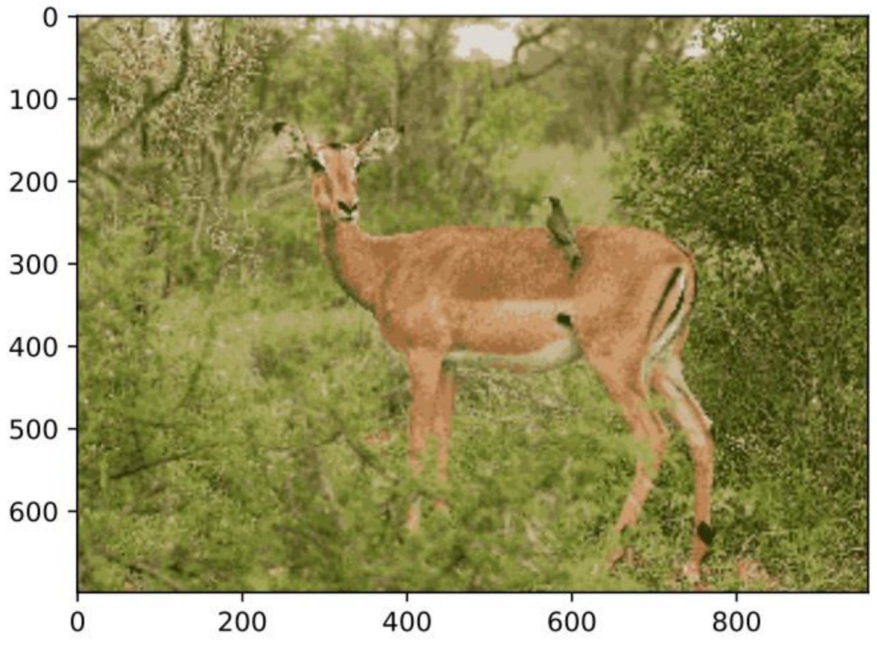
Note: imread works with *.jpg* images, it may not work with certain other formats. Select a small image to speed up the computations (e.g. 200 × 300). You may replace it with a large version later when everything works, if you wish (not needed!).

Next, let's convert your image down to 16 colors only. We use clustering to group the colors and convert each pixel to its cluster center, e.g., one of the 16 colors.

See the code snippet here:

```
pixels.shape
# convert the 'pixels' matrix into Nx3 (or Nx4) matrix
# where N is the number of pixels, and 3 or 4 is the number
# of color channels
# so each row is a pixel, and the row data is its color value

## (699, 960, 3)

M = pixels.reshape((-1,pixels.shape[2]))
print(M.shape)
# 3 (or 4) columns, one for each color channel

## (671040, 3)

from sklearn.cluster import KMeans

m = KMeans(16).fit(M)  # cluster the color values
cl = m.predict(M)  # find the cluster id for each pixel
centers = m.cluster_centers_  # find the cluster centers (i.e. colors)
print("center color values:\n", centers)

## center color values:
##  [[ 44.86871176  49.1297038    7.01861714]
##  [143.11847223 148.24711086  66.22198975]
```

```
##  [176.68585671 167.12100626 113.1931834 ]
##  [102.65626718 114.96031964  35.89857064]
##  [114.97161389 107.41662542  55.31214475]
##  [185.96837837 133.06239598  85.79846885]
##  [223.93323061 218.44821092 210.39565143]
##  [163.1657612  163.08682442  84.19648678]
##  [200.90147733 182.58038716 149.15287825]
##  [ 90.44833374  91.74528035  35.02174528]
##  [162.76780704 110.53807531  62.83140537]
##  [ 68.62599018  77.72338047  17.64156583]
##  [152.09907061 142.09930726  91.77580569]
##  [132.73763819 125.67639866  72.80842546]
##  [203.01960867 156.21095062 109.63520322]
##  [123.35610654 132.54940001  51.10778944]]

compressed = centers[cl]/255  # normalize to between 0 and 1
compressed = compressed.reshape(pixels.shape)
# make it back to NxM array
ax = plt.axes()
ax.set_aspect("equal")
_ = plt.imshow(compressed)
_ = plt.show()
```



3.2. (2 points) Get this code to work with your image and understand what it does. Explain:

(a) (0.4 point) What does the shape of *pixels* mean?

(b) (0.4 point) What is matrix M? Why does it have 671,040 rows and 3 columns in this example?

(c) (0.4 point) What do *cl* values tell us? E.g. if cl[12345] = 2, what does it mean?

(d) (0.4 point) What are the columns of “center color values” printed above? E.g. the first row, second column is value 163.37. What does that number mean?

(e) (0.4 point) How many “center color values” are there? Why do we have this number?

3.3. (2 points) Next repeat the above by reducing the number of colors to 2, and 4. Show the pictures!

Hint: make a function that takes the number of colors as an argument, then you don’t have to copy-paste your code.

Finally: How much time did you spend on this problem set?