**Handwriting Recognition using K-Means**

The [U.S. Postal Service](https://www.govexec.com/federal-news/1999/02/postal-service-tests-handwriting-recognition-system/1746/) has been using machine learning and scanning technologies since 1999. Because its postal offices have to look at roughly half a billion pieces of mail every day, they have done extensive research and developed very efficient algorithms for reading and understanding addresses. And not only the post office:

* ATMs can recognize handwritten bank checks
* Evernote can recognize handwritten task lists
* Expensify can recognize handwritten receipts

But how do they do it?

In this project, you will be using **K-means clustering** (the algorithm behind this magic) and scikit-learn to cluster images of handwritten digits.

Let's get started!

If you get stuck during this project, check out the **project walkthrough video** which can be found at the bottom of the page after the final step of the project.

Mark the tasks as complete by checking them off

**Getting Started with the Digits Dataset:**

1.

The sklearn library comes with a digits dataset for practice.

In **script.py**, we have already added three lines of code:

import codecademylib3\_seaborn

import numpy as np

from matplotlib import pyplot as plt

From sklearn library, import the datasets module.

Then, load in the digits data using .load\_digits() and print digits.

2.

When first starting out with a dataset, it’s always a good idea to go through the data description and see what you can already learn.

Instead of printing the digits, print digits.DESCR.

* What is the size of an image (in pixel)?
* Where is this dataset from?

3.

Let's see what the data looks like!

Print digits.data.

4.

Next, print out the target values in digits.target.

5.

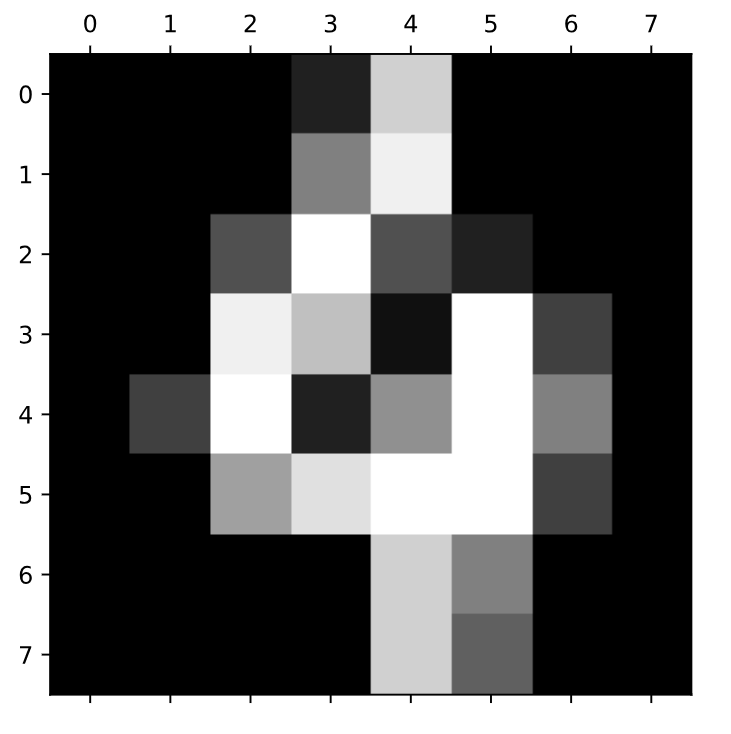
To visualize the data images, we need to use Matplotlib. Let's visualize the image at index 100:

plt.gray()

plt.matshow(digits.images[100])

plt.show()

The image should look like:



Is it a 4? Let's print out the target label at index 100 to find out!

print(digits.target[100])

Open the hint to see how you can visualize more than one image.

**K-Means Clustering:**

6.

Now we understand what we are working with. Let's cluster the 1797 different digit images into groups.

Import KMeans from sklearn.cluster.

7.

What should be the k, the number of clusters, here?

Use the KMeans() method to build a model that finds k clusters.

8.

Use the .fit() method to fit the digits.data to the model.

**Visualizing after K-Means:**

9.

Let's visualize all the centroids! Because data samples live in a 64-dimensional space, the centroids have values so they can be images!

First, add a figure of size 8x3 using .figure().

Then, add a title using .suptitle().

10.

Scikit-learn sometimes calls centroids "cluster centers".

Write a for loop to displays each of the cluster\_centers\_ like so:

for i in range(10):

# Initialize subplots in a grid of 2X5, at i+1th position

ax = fig.add\_subplot(2, 5, 1 + i)

# Display images

ax.imshow(model.cluster\_centers\_[i].reshape((8, 8)), cmap=plt.cm.binary)

The cluster centers should be a list with 64 values (0-16). Here, we are making each of the cluster centers into an 8x8 2D array.

11.

Outside of the for loop, use .show() to display the visualization.

It should look like:



These are the centroids of handwriting from *thirty* different people collected by Bogazici University (Istanbul, Turkey):

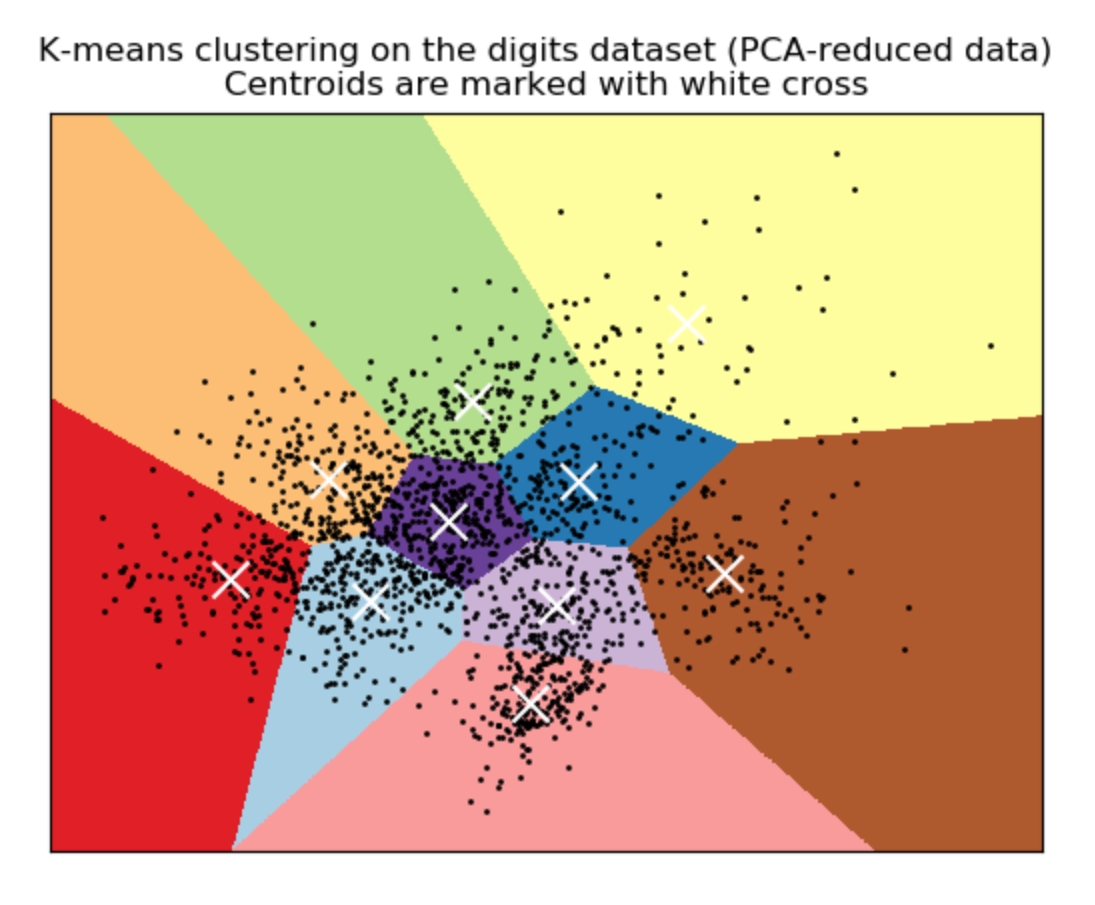
* Index 0 looks like 0
* Index 1 looks like 9
* Index 2 looks like 2
* Index 3 looks like 1
* Index 4 looks like 6
* Index 5 looks like 8
* Index 6 looks like 4
* Index 7 looks like 5
* Index 8 looks like 7
* Index 9 looks like 3

Notice how the centroids that look like 1 and 8 look very similar and 1 and 4 also look very similar.

12.

**Optional:**

If you want to see another example that visualizes the data clusters and their centers using K-means, check out the sklearn's [own example](http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html).



**Testing Your Model:**

13.

Instead of feeding new arrays into the model, let's do something cooler!

Inside the right panel, go to **test.html**.

14.

What year will robots take over the world?

Use your mouse to write a digit in each of the boxes and click Get Array.

15.

Back in **script.py**, create a new variable named new\_samples and copy and paste the 2D array into it.

new\_samples = np.array( )

16.

Use the .predict() function to predict new labels for these four new digits. Store those predictions in a variable named new\_labels.

17.

But wait, because this is a clustering algorithm, we don't know which label is which.

By looking at the cluster centers, let's map out each of the labels with the digits we think it represents:

for i in range(len(new\_labels)):

if new\_labels[i] == 0:

print(0, end='')

elif new\_labels[i] == 1:

print(9, end='')

elif new\_labels[i] == 2:

print(2, end='')

elif new\_labels[i] == 3:

print(1, end='')

elif new\_labels[i] == 4:

print(6, end='')

elif new\_labels[i] == 5:

print(8, end='')

elif new\_labels[i] == 6:

print(4, end='')

elif new\_labels[i] == 7:

print(5, end='')

elif new\_labels[i] == 8:

print(7, end='')

elif new\_labels[i] == 9:

print(3, end='')

18.

Is the model recognizing your handwriting?

Remember, this model is trained on handwritten digits of 30 Turkish people (from the 1990's).

Try writing your digits similar to these cluster centers:



19.

If you are stuck on the project or would like to see an experienced developer work through the project, watch the following project walkthrough video! <https://youtu.be/yrIcXqdY91U>

import codecademylib3\_seaborn

import numpy as np

from matplotlib import pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

digits = datasets.load\_digits()

#print(digits)

#print(digits.DESCR)

# indivisual images seem to be 4 blocks of 8 x 8 pixels so 32 pixels?

# data is from UCI

#print(digits.data)

#print(digits.target) # integers 0-9

plt.gray()

plt.matshow(digits.images[100])

plt.show()

print(digits.target[100]) # is a 4

## To see 64 sample images....

# Figure size (width, height)

fig = plt.figure(figsize=(6, 6))

# Adjust the subplots

fig.subplots\_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)

# For each of the 64 images

for i in range(64):

# Initialize the subplots: add a subplot in the grid of 8 by 8, at the i+1-th position

ax = fig.add\_subplot(8, 8, i+1, xticks=[], yticks=[])

# Display an image at the i-th position

ax.imshow(digits.images[i], cmap=plt.cm.binary, interpolation='nearest')

# Label the image with the target value

ax.text(0, 7, str(digits.target[i]))

plt.show()

## K Means Clustering

k = 10 # e.g. integers 0-9

model = KMeans(n\_clusters = k)

model.fit(digits.data)

## Visualizing after K Means

fig = plt.figure(figsize=(8, 3))

plt.suptitle('Visualizing After K Means')

for i in range(k):

# Initialize subplots in a grid of 2X5, at i+1th position

ax = fig.add\_subplot(2, 5, 1 + i)

# Display images

ax.imshow(model.cluster\_centers\_[i].reshape((8, 8)), cmap=plt.cm.binary)

plt.show()

## Testing You Model

new\_samples = np.array([

[0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,2.05,6.86,6.86,6.86,6.72,1.21,0.00,0.00,0.83,3.81,3.81,5.19,7.63,2.06,0.00,0.00,0.00,0.00,0.00,4.42,7.40,0.23,0.00,0.00,0.00,0.00,0.00,5.48,6.55,0.00,0.00,0.00,0.00,0.00,0.00,6.78,4.95,0.00,0.07,0.00,0.00,0.00,0.00,6.48,7.40,6.18,7.02,4.95,0.00,0.00,0.00,1.15,4.04,4.58,4.65,2.51],

[0.00,0.00,0.00,0.00,2.06,0.16,0.00,0.00,0.00,0.00,0.69,5.50,7.63,2.20,0.00,0.00,0.00,2.51,6.87,7.55,7.63,2.97,0.00,0.00,0.00,3.58,5.87,2.37,7.63,3.05,0.00,0.00,0.00,0.00,0.00,0.83,7.63,3.05,0.00,0.00,0.00,0.00,0.00,1.52,7.63,2.66,0.00,0.00,0.00,0.00,0.00,1.89,7.63,2.28,0.00,0.00,0.00,0.00,0.00,1.75,6.02,1.21,0.00,0.00],

[0.00,0.00,0.46,0.76,0.61,0.00,0.00,0.00,0.00,0.46,7.03,7.63,7.55,3.58,0.00,0.00,0.00,1.90,7.63,5.56,7.63,4.57,0.00,0.00,0.00,2.28,7.63,7.25,6.85,0.61,0.00,0.00,0.00,0.53,7.25,7.63,6.10,0.23,0.00,0.00,0.29,6.19,7.63,6.10,7.63,3.66,0.00,0.00,1.22,7.63,7.09,4.42,7.41,4.49,0.00,0.00,0.00,2.44,5.72,6.94,6.56,1.68,0.00,0.00],

[0.00,0.00,0.00,0.07,0.69,0.15,0.00,0.00,0.00,0.00,3.97,7.17,7.63,4.95,0.00,0.00,0.00,0.00,6.85,6.63,5.42,7.63,5.41,0.00,0.00,0.00,4.58,7.63,5.72,7.63,6.10,0.00,0.00,0.00,0.31,4.80,7.17,7.63,5.11,0.00,0.00,0.00,1.91,0.99,6.79,7.40,1.14,0.00,0.00,0.67,7.63,4.02,7.39,4.87,0.00,0.00,0.00,0.22,6.71,7.63,7.62,5.11,0.00,0.00]

])

new\_labels = model.predict(new\_samples)

'''But wait, because this is a clustering algorithm, we don't know which label is which.

By looking at the cluster centers, let's map out each of the labels with the digits we think it represents:'''

for i in range(len(new\_labels)):

if new\_labels[i] == 0:

print(0, end='')

elif new\_labels[i] == 1:

print(9, end='')

elif new\_labels[i] == 2:

print(2, end='')

elif new\_labels[i] == 3:

print(1, end='')

elif new\_labels[i] == 4:

print(6, end='')

elif new\_labels[i] == 5:

print(8, end='')

elif new\_labels[i] == 6:

print(4, end='')

elif new\_labels[i] == 7:

print(5, end='')

elif new\_labels[i] == 8:

print(7, end='')

elif new\_labels[i] == 9:

print(3, end='')

#This thing has zero hope of recognizing my handwriting haha