DS-210: Programming for Data Science

Lecture 7:

- Clustering
- k–means with SciPy

Clustering

General idea

- **Input:** set of objects
- Some information about relationship between them
- **Goal:** partition the objects into groups of similar objects

Clearly: unsupervised learning

Clustering

General idea

- **Input:** set of objects
- Some information about relationship between them
- **Goal:** partition the objects into groups of similar objects

Clearly: unsupervised learning

Why clustering?

Clustering

General idea

- **Input:** set of objects
- Some information about relationship between them
- **Goal:** partition the objects into groups of similar objects

Clearly: unsupervised learning

Why clustering?

- Discover similar cases
- Make sense of data
- Reduce data size

Examples of popular types of clustering

- k–means
- correlation clustering
- (hierarchical) agglomerative clustering (HAC)

k–means

- k is the target number of clusters
- **Input:** set S of points in $\mathbb{R}^n$
- **Euclidean between points:**

$$\text{dist}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- **Ideal solution:** set $C \subseteq \mathbb{R}^n$ of k points that minimize

$$\sum_{x \in S} \min_{c \in C} (\text{dist}(x, c))^2$$

(points in C are *cluster centers*)

- **Clusters:**
  - Assign each point $x \in S$ to the closest $c \in C$
  - One cluster for each $c \in C$: the points assigned to it

k–means

- k is the target number of clusters
- **Input:** set S of points in $\mathbb{R}^n$
- **Euclidean between points:**

$$\text{dist}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- **Ideal solution:** set $C \subseteq \mathbb{R}^n$ of k points that minimize

$$\sum_{x \in S} \min_{c \in C}(\text{dist}(x, c))^2$$

(points in C are *cluster centers*)

- **Clusters:**
  - Assign each point $x \in S$ to the closest $c \in C$
  - One cluster for each $c \in C$: the points assigned to it

Reality

- NP–hard
- Likely exponential time needed

k–means

- k is the target number of clusters
- **Input:** set S of points in $\mathbb{R}^n$
- **Euclidean between points:**

$$\text{dist}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- **Ideal solution:** set $C \subseteq \mathbb{R}^n$ of k points that minimize

$$\sum_{x \in S} \min_{c \in C} (\text{dist}(x, c))^2$$

(points in C are *cluster centers*)

- **Clusters:**
  - Assign each point $x \in S$ to the closest $c \in C$
  - One cluster for each $c \in C$: the points assigned to it

Reality

- NP–hard
- Likely exponential time needed

Typical heuristic

1. *Seeding*: Start from some solution C
2. Keep improving C until satisfied

Part 1: Initial solution (seeding)

Example 1: random assignment

- Option 1: select k points from S
    - likely to focus on the more populous parts of the data set
- Option 2: select k points from the area to which points belong
    - points might end up outside of the area of interest
    - points may not be a minimum for any point in S

Part 1: Initial solution (seeding)

Example 1: random assignment

- Option 1: select k points from S
  - likely to focus on the more populous parts of the data set
- Option 2: select k points from the area to which points belong
  - points might end up outside of the area of interest
  - points may not be a minimum for any point in S

Example 2: k–means++

- very popular heuristic
- iterative (i.e., add points one by one):
  - given current C, assign weights to all points in S
  - $\text{weight(x)} = \min_{c \in C}(\text{dist}(x, c))^2$
  - draw next point with probabilities proportional to the weights
- relatively good approximation in expectation

Part 2: Iterative improvement

Typical iteration

- Assign each point in $x \in S$ to the closest center $c \in C$
- For each $c \in C$:
  - let S_c be points assigned to C
  - move c to
    $$\frac{1}{|S_c|} \sum_{y \in S_c} y$$
    if S_c is not empty
  - Note: the new location minimizes
    $$\sum_{x \in S_c} (\mathrm{dist}(x, c))^2$$

Part 2: Iterative improvement

Typical iteration

- Assign each point in $x \in S$ to the closest center $c \in C$
- For each $c \in C$:
    - let S_c be points assigned to C
    - move c to
      $$\frac{1}{|S_c|} \sum_{y \in S_c} y$$
      if S_c is not empty
    - Note: the new location minimizes
      $$\sum_{x \in S_c} (\text{dist}(x, c))^2$$

When to stop

- fixed number of steps?
- the solution has stopped improving?

Part 2: Iterative improvement

Typical iteration

- Assign each point in $x \in S$ to the closest center $c \in C$
- For each $c \in C$:
  - let S_c be points assigned to C
  - move c to
    $$\frac{1}{|S_c|} \sum_{y \in S_c} y$$
    if S_c is not empty
  - Note: the new location minimizes
    $$\sum_{x \in S_c} (\mathrm{dist}(x, c))^2$$

When to stop

- fixed number of steps?
- the solution has stopped improving?

General problems

- may get stuck in a local minimum
- may improve very slowly
- possibly good ideas:
  - try different seeding methods
  - run multiple times from different starting points

Example: Reduce number of colors in an image

In [1]:

```
# PIL usually distributed as "Pillow"
from PIL import Image
import numpy as np
image = Image.open("cds.png")
image
```

Out[1]:

Example: Reduce number of colors in an image

In [1]:
```python
# PIL usually distributed as "Pillow"
from PIL import Image
import numpy as np
image = Image.open("cds.png")
image
```

Out[1]:

Typical color representation: RGB

- (red, green, blue), each in $0 \ldots 255$
- `uint8` = 8 bits = 1 byte

Example: Reduce number of colors in an image

In [1]:
```
# PIL usually distributed as "Pillow"
from PIL import Image
import numpy as np
image = Image.open("cds.png")
image
```

Out[1]:

Typical color representation: RGB

- (red, green, blue), each in $0 \ldots 255$
- `uint8` $= 8$ bits $= 1$ byte

In [2]:
```
arr = np.asarray(image)
## drop additional transparency channel (alpha)
# arr = arr[:,:,:3]
print(arr.shape)
arr
```

```
(400, 711, 4)
```

Out[2]:
```
array([[[ 53,  82, 152, 255],
        [ 52,  82, 152, 255],
        [ 50,  80, 152, 255],
        ...,
        [ 13,  13,  11, 255],
        [ 12,  12,  10, 255],
        [ 18,  18,  16, 255]],

       [[ 56,  85, 156, 255],
        [ 52,  81, 152, 255],
        [ 51,  81, 153, 255],
        ...,
        [ 14,  14,  12, 255],
        [ 15,  15,  13, 255],
        [ 18,  18,  16, 255]],

       [[ 57,  85, 158, 255],
```

Example: Reduce number of colors in an image

In [1]:
```
# PIL usually distributed as "Pillow"
from PIL import Image
import numpy as np
image = Image.open("cds.png")
image
```

Out[1]:

Typical color representation: RGB

- (red, green, blue), each in $0 \ldots 255$
- `uint8` = 8 bits = 1 byte

In [3]:
```
arr = np.asarray(image)
## drop additional transparency channel (alpha)
arr = arr[:,:,:3]
print(arr.shape)
arr
```

```
(400, 711, 3)
```

Out[3]:
```
array([[[ 53,  82, 152],
        [ 52,  82, 152],
        [ 50,  80, 152],
        ...,
        [ 13,  13,  11],
        [ 12,  12,  10],
        [ 18,  18,  16]],

       [[ 56,  85, 156],
        [ 52,  81, 152],
        [ 51,  81, 153],
        ...,
        [ 14,  14,  12],
        [ 15,  15,  13],
        [ 18,  18,  16]],

       [[ 57,  85, 158],
```

Example: Reduce number of colors in an image

In [4]:
```
# save dimensions
height,width,color_dim = arr.shape
# turn into a "1D" array of pixels
arr = arr.reshape(-1,color_dim)
arr
```

Out[4]:
```
array([[ 53,  82, 152],
       [ 52,  82, 152],
       [ 50,  80, 152],
       ...,
       [ 42,  44,  59],
       [ 39,  43,  61],
       [ 43,  50,  69]], dtype=uint8)
```

Example: Reduce number of colors in an image

```
In [4]: # save dimensions
        height,width,color_dim = arr.shape
        # turn into a "1D" array of pixels
        arr = arr.reshape(-1,color_dim)
        arr

Out[4]: array([[ 53,  82, 152],
               [ 52,  82, 152],
               [ 50,  80, 152],
               ...,
               [ 42,  44,  59],
               [ 39,  43,  61],
               [ 43,  50,  69]], dtype=uint8)
```

```
In [5]: from scipy.cluster.vq import kmeans, kmeans2
        arr = arr.astype(np.float32)
        codebook,_ = kmeans(arr,2)
        # codebook,_ = kmeans2(arr,16,minit='++')
        codebook

Out[5]: array([[131.33862 , 153.9153  , 194.26178 ],
               [ 44.8325  ,  44.519474,  50.553364]], dtype=floa
        t32)
```

Example: Reduce number of colors in an image

In [6]:
```
# assign closest center to each pixel
from scipy.cluster.vq import vq
encoding,_ = vq(arr,codebook)
encoding
```

Out[6]:
```
array([1, 1, 1, ..., 1, 1, 1], dtype=int32)
```

Example: Reduce number of colors in an image

In [6]:
```
# assign closest center to each pixel
from scipy.cluster.vq import vq
encoding,_ = vq(arr,codebook)
encoding
```

Out[6]:
```
array([1, 1, 1, ..., 1, 1, 1], dtype=int32)
```

In [7]:
```
# make color coordinates small integers
codebook = codebook.astype(np.uint8)
codebook
```

Out[7]:
```
array([[131, 153, 194],
       [ 44,  44,  50]], dtype=uint8)
```

Example: Reduce number of colors in an image

In [6]:
```
# assign closest center to each pixel
from scipy.cluster.vq import vq
encoding,_ = vq(arr,codebook)
encoding
```

Out[6]:
```
array([1, 1, 1, ..., 1, 1, 1], dtype=int32)
```

In [7]:
```
# make color coordinates small integers
codebook = codebook.astype(np.uint8)
codebook
```

Out[7]:
```
array([[131, 153, 194],
       [ 44,  44,  50]], dtype=uint8)
```

Example: Reduce number of colors in an image

In [6]:
```
# assign closest center to each pixel
from scipy.cluster.vq import vq
encoding,_ = vq(arr,codebook)
encoding
```

Out[6]:
```
array([1, 1, 1, ..., 1, 1, 1], dtype=int32)
```

In [7]:
```
# make color coordinates small integers
codebook = codebook.astype(np.uint8)
codebook
```

Out[7]:
```
array([[131, 153, 194],
       [ 44,  44,  50]], dtype=uint8)
```

In [8]:
```
# map entries to closest colors
newarr = [codebook[entry] for entry in encoding]
newarr = np.array(newarr)
newarr
```

Out[8]:
```
array([[44, 44, 50],
       [44, 44, 50],
       [44, 44, 50],
       ...,
       [44, 44, 50],
       [44, 44, 50],
       [44, 44, 50]], dtype=uint8)
```

Example: Reduce number of colors in an image

In [6]:
```
# assign closest center to each pixel
from scipy.cluster.vq import vq
encoding,_ = vq(arr,codebook)
encoding
```

Out[6]:
```
array([1, 1, 1, ..., 1, 1, 1], dtype=int32)
```

In [7]:
```
# make color coordinates small integers
codebook = codebook.astype(np.uint8)
codebook
```

Out[7]:
```
array([[131, 153, 194],
       [ 44,  44,  50]], dtype=uint8)
```

In [8]:
```
# map entries to closest colors
newarr = [codebook[entry] for entry in encoding]
newarr = np.array(newarr)
newarr
```

Out[8]:
```
array([[44, 44, 50],
       [44, 44, 50],
       [44, 44, 50],
       ...,
       [44, 44, 50],
       [44, 44, 50],
       [44, 44, 50]], dtype=uint8)
```

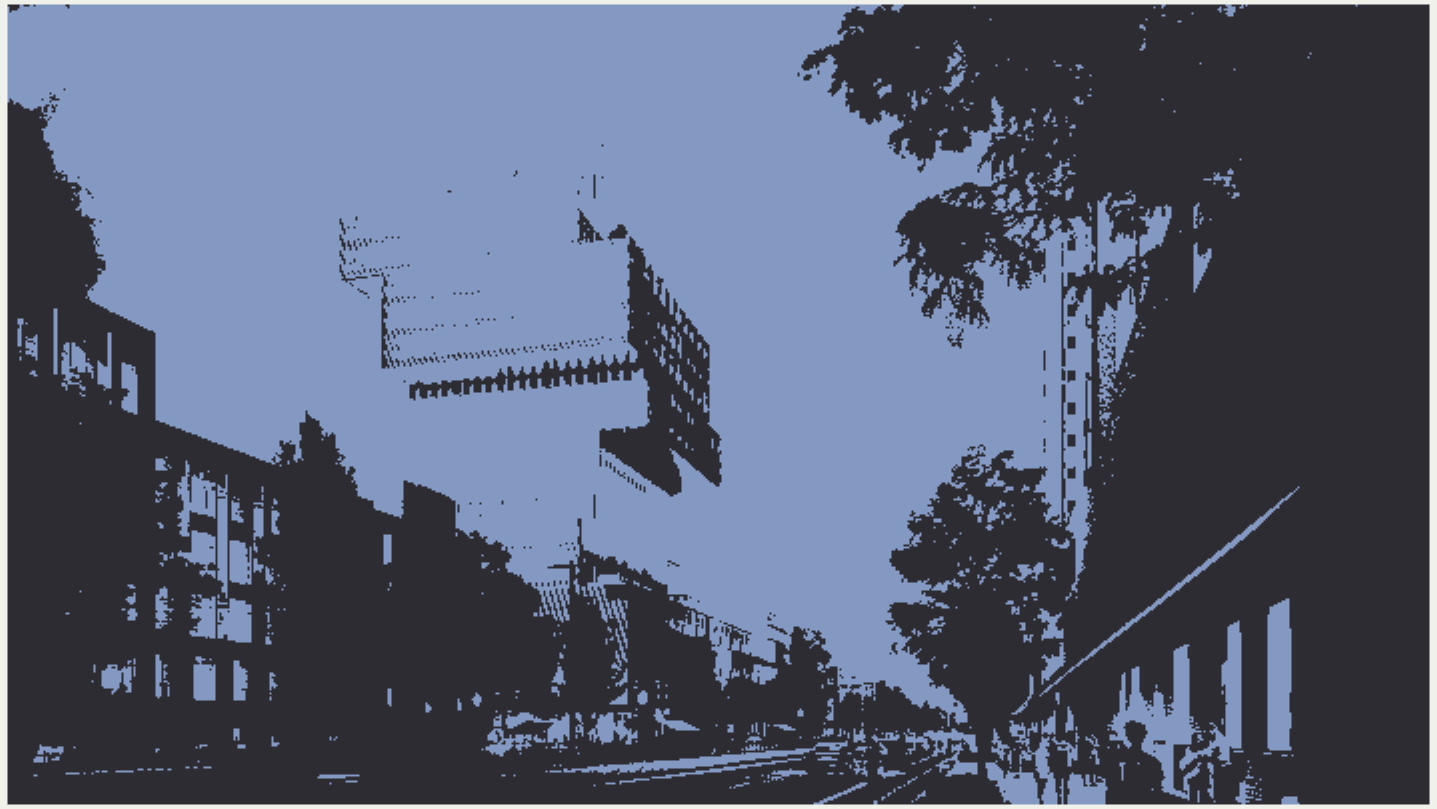
In [9]:
```
newarr = newarr.reshape(height,width,color_dim)
newarr
```

Out[9]:
```
array([[[ 44,  44,  50],
        [ 44,  44,  50],
        [ 44,  44,  50],
        ...,
        [ 44,  44,  50],
        [ 44,  44,  50],
        [ 44,  44,  50]],

       [[131, 153, 194],
        [ 44,  44,  50],
        [ 44,  44,  50],
        ...,
        [ 44,  44,  50],
        [ 44,  44,  50],
        [ 44,  44,  50]],

       [[131, 153, 194],
        [ 44,  44,  50],
        [ 44,  44,  50],
        ...,
        [ 44,  44,  50],
        [ 44,  44,  50],
        [ 44,  44,  50]],
```

Example: Reduce number of colors in an image

In [10]:
```
image = Image.fromarray(newarr)
image.save("test.png")
image
```

Out[10]:



Example: Reduce number of colors in an image

In [4]:

```
# save dimensions
height,width,color_dim = arr.shape
# turn into a "1D" array of pixels
arr = arr.reshape(-1,color_dim)
arr
```

Out[4]:

```
array([[ 53,  82, 152],
       [ 52,  82, 152],
       [ 50,  80, 152],
       ...,
       [ 42,  44,  59],
       [ 39,  43,  61],
       [ 43,  50,  69]], dtype=uint8)
```

In [11]:

```
from scipy.cluster.vq import kmeans, kmeans2
arr = arr.astype(np.float32)
#codebook,_ = kmeans(arr,2)
codebook,_ = kmeans2(arr,16,minit='++')
codebook
```

Out[11]:

```
array([[ 20.919214,  20.857271,  20.544157],
       [224.16663 , 230.47906 , 237.03946 ],
       [ 82.67631 , 129.56158 , 200.35356 ],
       [169.48045 , 160.82599 , 166.21234 ],
       [ 36.128433,  35.52045 ,  38.220963],
       [ 62.307972,  75.69994 , 105.28504 ],
       [125.72108 , 131.8019  , 152.09633 ],
       [177.50243 , 196.5755  , 228.81677 ],
       [129.51418 , 165.71692 , 217.94589 ],
       [138.75111 , 117.28548 , 113.688225],
       [ 87.882614,  98.428535, 122.83324 ],
       [ 52.486977,  51.429337,  59.03327 ],
       [106.24081 ,  89.817   ,  89.622925],
       [ 77.11434 ,  68.19252 ,  71.484474],
       [205.93745 , 197.98878 , 199.40974 ],
       [ 62.69038 ,  96.97973 , 166.9947  ]], dtype=floa
t32)
```

Example: Reduce number of colors in an image

In [12]:
```
# assign closest center to each pixel
from scipy.cluster.vq import vq
encoding,_ = vq(arr,codebook)
encoding
```

Out[12]:
```
array([15, 15, 15, ..., 11, 11, 11], dtype=int32)
```

Example: Reduce number of colors in an image

In [12]:
```
# assign closest center to each pixel
from scipy.cluster.vq import vq
encoding,_ = vq(arr,codebook)
encoding
```

Out[12]:
```
array([15, 15, 15, ..., 11, 11, 11], dtype=int32)
```

In [13]:
```
# make color coordinates small integers
codebook = codebook.astype(np.uint8)
codebook
```

Out[13]:
```
array([[ 20,  20,  20],
       [224, 230, 237],
       [ 82, 129, 200],
       [169, 160, 166],
       [ 36,  35,  38],
       [ 62,  75, 105],
       [125, 131, 152],
       [177, 196, 228],
       [129, 165, 217],
       [138, 117, 113],
       [ 87,  98, 122],
       [ 52,  51,  59],
       [106,  89,  89],
       [ 77,  68,  71],
       [205, 197, 199],
       [ 62,  96, 166]], dtype=uint8)
```

Example: Reduce number of colors in an image

In [12]:
```python
# assign closest center to each pixel
from scipy.cluster.vq import vq
encoding,_ = vq(arr,codebook)
encoding
```

Out[12]:
```
array([15, 15, 15, ..., 11, 11, 11], dtype=int32)
```

In [13]:
```python
# make color coordinates small integers
codebook = codebook.astype(np.uint8)
codebook
```

Out[13]:
```
array([[ 20,  20,  20],
       [224, 230, 237],
       [ 82, 129, 200],
       [169, 160, 166],
       [ 36,  35,  38],
       [ 62,  75, 105],
       [125, 131, 152],
       [177, 196, 228],
       [129, 165, 217],
       [138, 117, 113],
       [ 87,  98, 122],
       [ 52,  51,  59],
       [106,  89,  89],
       [ 77,  68,  71],
       [205, 197, 199],
       [ 62,  96, 166]], dtype=uint8)
```

In [14]:
```python
# map entries to closest colors
newarr = [codebook[entry] for entry in encoding]
newarr = np.array(newarr)
newarr
```

Out[14]:
```
array([[ 62,  96, 166],
       [ 62,  96, 166],
       [ 62,  96, 166],
       ...,
       [ 52,  51,  59],
       [ 52,  51,  59],
       [ 52,  51,  59]], dtype=uint8)
```

Example: Reduce number of colors in an image

In [12]:
```
# assign closest center to each pixel
from scipy.cluster.vq import vq
encoding,_ = vq(arr,codebook)
encoding
```

Out[12]:
```
array([15, 15, 15, ..., 11, 11, 11], dtype=int32)
```

In [13]:
```
# make color coordinates small integers
codebook = codebook.astype(np.uint8)
codebook
```

Out[13]:
```
array([[ 20,  20,  20],
       [224, 230, 237],
       [ 82, 129, 200],
       [169, 160, 166],
       [ 36,  35,  38],
       [ 62,  75, 105],
       [125, 131, 152],
       [177, 196, 228],
       [129, 165, 217],
       [138, 117, 113],
       [ 87,  98, 122],
       [ 52,  51,  59],
       [106,  89,  89],
       [ 77,  68,  71],
       [205, 197, 199],
       [ 62,  96, 166]], dtype=uint8)
```

In [14]:
```
# map entries to closest colors
newarr = [codebook[entry] for entry in encoding]
newarr = np.array(newarr)
newarr
```

Out[14]:
```
array([[ 62,  96, 166],
       [ 62,  96, 166],
       [ 62,  96, 166],
       ...,
       [ 52,  51,  59],
       [ 52,  51,  59],
       [ 52,  51,  59]], dtype=uint8)
```

In [15]:
```
newarr = newarr.reshape(height,width,color_dim)
newarr
```

Out[15]:
```
array([[[ 62,  96, 166],
        [ 62,  96, 166],
        [ 62,  96, 166],
        ...,
        [ 20,  20,  20],
        [ 20,  20,  20],
        [ 20,  20,  20]],

       [[ 62,  96, 166],
        [ 62,  96, 166],
```

Example: Reduce number of colors in an image

In [16]:
```
image = Image.fromarray(newarr)
image.save("test.png")
image
```

Out[16]:

Final comments

Warning: Normalizing your data may be useful or crucial

- You have to make sure that all relevant coordinates have some impact
- **Sample solution:** make the variance / standard deviation of each coordinate identical
- Implemented as `scipy.cluster.vq.whiten`

Final comments

Warning: Normalizing your data may be useful or crucial

- You have to make sure that all relevant coordinates have some impact
- **Sample solution:** make the variance / standard deviation of each coordinate identical
- Implemented as `scipy.cluster.vq.whiten`

Some k–means implementations

- Two implementations in SciPy
  - `scipy.cluster.vq.kmeans`
  - `scipy.cluster.vq.kmeans2`
- `scikit-learn`: `sklearn.cluster.KMeans`
- Feel free to experiment to see which one is better