Data Analysis \rightarrow Data Mining

Recursion

Recall: Data analysis

- access data
- file processing
- inspect (clean) data
- store data in Python objects/data structures
- analyze data, e.g.,
  - mean
  - range
  - median
  - standard deviation
  - mode
- report on results of data analysis, e.g.,
  - frequency occurrence table
  - frequency occurrence chart

Data Analysis and Data Mining

- access data
- file processing
- inspect (clean) data
- store data in (appropriate) Python data structures
- data analysis (summarizing), data mining (exploring)
  - mean
  - range
  - k-means cluster analysis
  - median
  - standard deviation
  - mode
- report on results of data analysis, e.g.,
  - frequency occurrence table/chart
  - data visualization

Data mining:

application of automated techniques that attempt to discover underlying patterns in the data.

For example, cluster analysis:

data mining technique that attempts to divide the data into meaningful groups called clusters.

A primary goal of data visualization is to communicate information clearly and efficiently via statistical graphics, plots, and information graphics.

Numerical data may be encoded ... to visually communicate a quantitative message.

Effective visualization helps users analyze and reason about data and evidence. It makes complex data more accessible, understandable and usable.
def visualizeQuakes(k, r, fname):
    """ (int, int, str) \rightarrow None

Data mining.
""

```
>>> visualizeQuakes(6, 7, 'earthquakes.txt')
qdatadict = readfile(fname)    # access file data
qCentroids = createCentroids(...)  # process data
qClusters = createClusters(...)  # cluster analysis
eqDraw(..., 1000, 500)          # report - visualization
return None
```

---

CIS 210

**Data mining:**
application of automated techniques that attempt to
*discover underlying patterns* in the data.

**Cluster analysis:**
data mining technique that attempts to *divide the data*
into meaningful groups called clusters.

**Cluster:**
data values that show some kind of similarity to each
other while exhibiting dissimilarity to data values
outside the cluster.

---

**IMPLEMENTING K-MEANS CLUSTER ANALYSIS**

1) review k-means clustering algorithm
2) work some examples – simple data, earthquake data
3) review high level structure of program – key functions
4) review important data structures (including earthquake data file)
5) code from the bottom up
6) testing each function thoroughly before proceeding to the next
7) integrated testing at each level

---

![Exam % vs Homework % scatter plot](image)

---

![Possible clusters within the data](image)
1. review k-means clustering algorithm
   - decide how many clusters (k)
   - assign each item from the data set to a cluster

   **Clustering the data – k-means clustering algorithm**

1) decide how many clusters (k)

   **Choose k depending on the data and desired results.**

   For example:
   - too hot, too cold, just right
   - group1, group2, group3, ..., group 10

   Run the analysis for different values of k.

---

2. assign each item from the data set to a cluster:
   - determine similarity of the item to other data in the cluster
   - assign item to cluster where data are most similar

   **k-means cluster analysis**

1) decide how many clusters (k)

2) assign each data item to a cluster:
   - determine similarity of the data item to other data in the cluster
     - determine a centroid for each cluster
     - determine a measure of similarity
     - measure the similarity of the data item to the centroid of each cluster
   - assign item to cluster where data are the most similar

3) do this until done

---

2. assign each data item to a cluster:
   - determine similarity of the data item to other data in the cluster
     - determine a centroid for each cluster
     - the mean value of the items in the cluster
     - determine a measure of similarity
     - distance between item and centroid
     - measure the similarity of the data item to the centroid for each cluster
   - assign item to cluster where data are the most similar

   **Clustering the data – k-means clustering algorithm**

2) assign each data item to a cluster:
   - by measuring the distance from a data item to the centroid of each cluster, then choosing the closest cluster
     - a) need initial centroids for initial clusters to form around
     - b) need to be precise about similarity (distance) measure
   - do until done:
     - c) need to determine when to stop

   **k-means cluster analysis**

1) decide how many clusters (k)

2) assign each data item to a cluster:
   - need initial centroids for initial clusters to form around
   - need to be precise about similarity (distance) measure

3) do until done:
   - need to determine when to stop
Initial centroids?

→ choose k random points from the data set for starter centroids.

Precise about similarity/distance measure?

Earthquake data items are points

→ use distance between two data points:
  -- data point from the data set
  -- a cluster centroid point

Distance: Euclidean distance

\[ d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \]

Each cluster is a collection of data points associated - measured by Euclidean distance - with the centroid of the cluster.

Centroid:
mean of a cluster (collection of data points)

Note that, after the initial set of centroids, the centroid might not be a member of the collection of data points.

k-means cluster analysis

1) decide how many clusters (k)
2) assign each data item to a cluster:

randomly choose k of the data points to serve as starter centroids for the k clusters.
assign each data point to the cluster with the centroid that is the closest (Euclidean distance) to the data point
3) do until done:
   c) need to determine when to stop

k-means cluster analysis

1) decide how many clusters (k)
2) assign each data item to a cluster:

randomly choose k of the data points to serve as starter centroids for the k clusters.
assign each data point to the cluster with the centroid that is the closest (Euclidean distance) to the data point
3) repeat until done: need to determine when to stop

recalculate the centroids of each cluster
re-assign data items to clusters

K-Means Cluster Analysis Algorithm

3) repeat until done: need to determine when to stop

e.g., # repetitions, clusters are stable

this algorithm: choose a reasonable value for r
k-means cluster analysis

1) decide how many clusters (k)

2) assign each data item to a cluster:
   randomly choose k of the data points to serve as starter centroids for
   the k clusters.
   assign each data point to the cluster with the centroid that is the closest
   (Euclidean distance) to the data point

3) until done (clusters created r times):
   recalibrate the centroid (mean) for each cluster
   assign each data point to the cluster with the centroid that is the closest
   (Euclidean distance) to the data point

```
def visualizeQuakes(k, r, fname):
    ''' (int, int, str) → None

    Data mining.
    >>> visualizeQuakes(6, 7, 'earthquakes.txt')
    '''
    qdatadict = readFile(fname)  # access file data
    qCentroids = createCentroids(...)  # process data -
    qClusters = createClusters(...)  # cluster analysis
    eqDraw(...)  # report - visualization
    return None
```

For Example:

```
data points = [34, 44, 10, 99, 50, 67, 0, 29, 50, 100]
k - ? design decision, e.g., 3 (low, medium, high)

centroids = [10, 50, 67] (initially, random choices from data)
clusters = [[], [], []] (first round)
centroids -> [10, 50, 67] (mean of each first round cluster)
clusters -> [??, ??, ??] (second round - use new centroids
                     to create new clusters)
centroids -> [??, ??, ??] (mean of each second round cluster)
clusters -> [??, ??, ??] (repeat until done)
```

For Example:  using a data dictionary (createClusters)

```
data points = {1: [34], 2 : [44], 3 : [10], 4 : [99], 5 : [50],
               6 : [67], 7 : [0], 8 : [29], 9 : [50], 10 : [100]}
k - ? design decision, e.g., 3 (low, medium, high)

centroids = [10, 50, 67] (initially, random choices from data)
clusters = [[], [], []] -> [[0,10,29], [34,44,50,50], [67,99,100]]
centroids -> [13.0, 44.5, 88.67] (mean of each cluster)
clusters -> [0, 10, 29], [34, 44, 50, 50], [67, 99, 100] (mean of each new cluster)
```

For Example:

```
data points = [34, 44, 10, 99, 50, 67, 0, 29, 50, 100]
k - ? design decision, e.g., 3 (low, medium, high)

centroids = [10, 50, 67] (initially, random choices from data)
clusters = [[], [], []] -> [[0,10,29], [34,44,50,50], [67,99,100]]
centroids -> [13.0, 44.5, 88.67] (mean of each cluster)
clusters -> [0, 10, 29], [34, 44, 50, 50], [67, 99, 100] (mean of each new cluster)
```
For Example: using a data dictionary

data dict = {1: [144.8897, 38.0555], 2: [45.5993, 34.9144], 3: [45.9411, 34.8857], 4: [58.2176, 9.1259], 5: [168.5847, -21.5194], 6: [-14.0995, -11.7482], 7: [168.4966, -21.4067], 8: [73.9098, 38.2556], 9: [129.9681, -7.0579], 10: [-84.0888, -41.4839], 11: [168.5912, -21.5266], 12: [141.5594, 32.4901]}

centroids = [[-84.0888, -41.4839], [141.5594, 32.4901], [144.8897, 38.0555], [45.5993, 34.9144], [73.9098, 38.2556], [168.5847, -21.5194]]

# clusters are lists of dictionary keys
[[5, 7, 11], [2], [3, 4, 8], [6], [1, 9, 12], [10]]

IMPLEMENTING K-MEANS CLUSTERING
✓ review k-means clustering algorithm
✓ work some examples – simple data, earthquake data
1) review high level structure of program – key functions
2) review important data structures (including earthquake data file)
3) code from the bottom up
4) testing each function thoroughly before proceeding to the next
5) integrated testing at each level

CIS 210

def readFile(filename):
    with open(filename, "r") as datafile:
        datadict = {}
        key = 0
        for aline in datafile:
            items = aline.split()
            key = key + 1
            lat = float(items[3])
            lon = float(items[4])
            datadict[key] = [lon, lat]
        return datadict

CIS 210
def createCentroids(k, datadict):
    centroids = []
    centroidCount = 0
    centroidKeys = []
    while centroidCount < k:
        rkey = random.randint(1, len(datadict))
        if rkey not in centroidKeys:
            centroids.append(datadict[rkey])
            centroidKeys.append(rkey)
            centroidCount = centroidCount + 1
    return centroids

CIS 210
def createClusters(k, centroids, datadict, r):
    on each pass (for a total of r passes):
    (1) set clusters to empty, e.g., [[], [], []]
    (2) for each key in datadict
       for each centroid
       compute distance between
datadict value(s) and the centroid
       choose the minimum distance
       put key in corresponding cluster
createClusters(k, centroids, datadict, r), cont’d on each pass (until r passes):

(3) for each cluster
determine the mean of the cluster
update centroids list with that mean

[repeat – next pass – until r passes]
CIS 210 Learning Outcomes

• understand, develop, implement algorithms for computational problem solving;

• use structured design and testing methods to develop and implement programs;

• read, write, revise, document, test, and debug code;

• demonstrate robust mental models of data representation and code execution;

• demonstrate good understanding of a high level programming language;

• introduce and/or implement a sampling of classic computer science problem domains and algorithms.