Unsupervised Learning

DS 8015

OUTLINE

1. Introduction
2. Hierarchical Clustering
3. K-means Clustering
4. Python Implementations

Introduction

UNSUPERVISED LEARNING PARADIGMS

- Unsupervised learning: No teacher providing supervision as to how individual instances should be handled.

  ⇒ There are no class labels

  Common tasks:

  - Clustering: Separate instances into groups
  - Novelty detection: Find instances that are very different from the rest.

CLUSTERING

- Also called data segmentation

- Two major methods
  1. Hierarchical clustering
     a. Agglomerative methods proceed as a series of fusions

     b. Divisive methods successively separate data into finer groups

  2. k-means clustering

     Partitions data into k clusters so that each element belongs to the cluster with the closest mean

Hierarchical Clustering

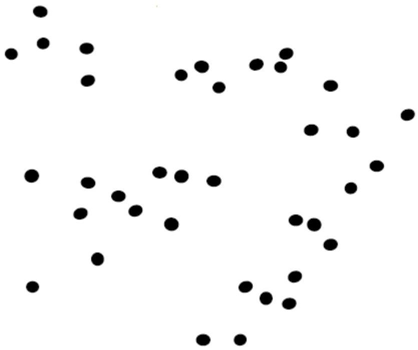
Hierarchical Clustering - 1

- Input:
  - A dataset $x_1, \dots, x_n$, each point is a numerical feature vector
  - Does NOT need the number of vectors.
- Overall logic:
  - Initially every point is in its own cluster.
  - Find the pair of clusters that are the closest.
  - Merge the two into a single cluster.
  - Repeat until the whole dataset is one giant cluster.

    $\Rightarrow$ You get a binary tree in the end.

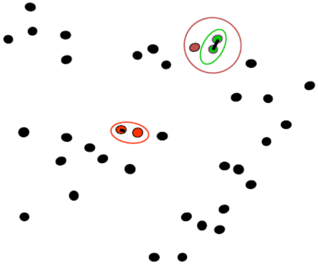
HIERARCHICAL CLUSTERING - 2

- Set of data points:



Hierarchical Clustering - 3

- Initial set of clusters:



Hierarchical Clustering - 4

- How do you measure the closeness between two clusters?
- Three ways:
  - **Single-linkage**: the shortest distance from any member of one cluster to any member of the other cluster. Formula?
  - **Complete-linkage**: the greatest distance from any member of one cluster to any member of the other cluster
  - **Average-linkage**: you guess it!
- The binary tree you get is often called a dendrogram, or taxonomy, or a hierarchy of data points
- The tree can be cut at various levels to produce different numbers of clusters: if you want k clusters, just cut the $(k-1)$ longest links
- Sometimes the hierarchy itself is more interesting than the clusters
- However there is not much theoretical justification to it...

Hierarchical Clustering - 5

Hierarchical Agglomerative Clustering

Input: a training sample $\{x_i\}_{i=1}^{n}$; a distance function $d()$

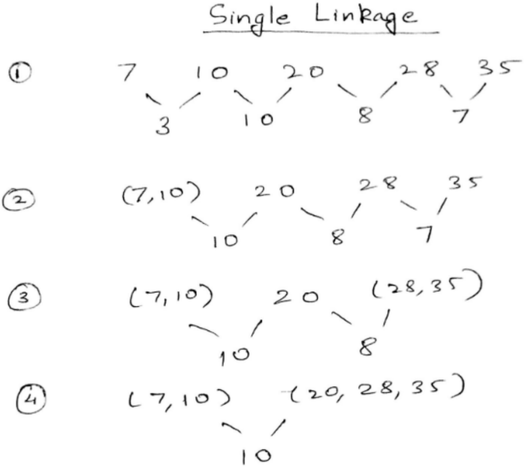
1. Initially, place each instance in its own cluster (called a singleton cluster)
2. while (number of clusters > 1) do:
3. Find the closest cluster pair A, B, i.e., they minimize $d(A, B)$
4. Merge $A,\ B$ to form a new cluster.

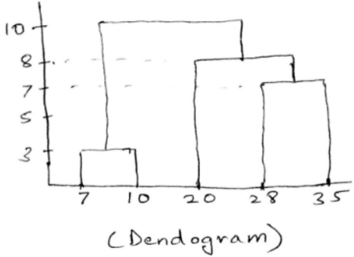
Output: a binary tree showing how clusters are gradually merged from singletons to a root cluster, which contains the whole training sample.

- Euclidean distance: $d(x_i, x_j) = ||x_i - x_j|| = \sqrt{\sum_{s=1}^{D}(x_{is} - x_{js})^2}$
- Manhattan distance: $d(x_i, x_j) = |x_i - x_j| = \sum_{s=1}^{D} |x_{is} - x_{js}|$
- What about the distance between two clusters?
  - Single linkage: $d(A, B) = \min\limits_{x \in A,\ x' \in B} d(x, x')$.
  - Complete linkage: replace min with max.

Hierarchical Clustering - Example 1a

- For the one dimensional data set 7,10,20,28,35, perform hierarchical clustering using **single linkage** and plot the dendogram to visualize it.

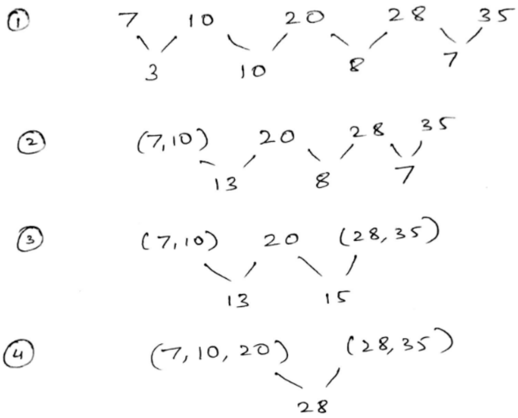


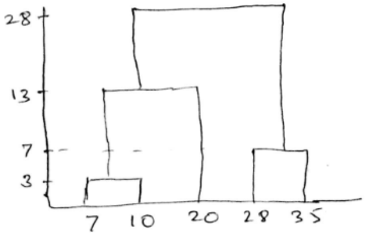


HIERARCHICAL CLUSTERING - EXAMPLE 1B

- For the one dimensional data set 7,10,20,28,35, perform hierarchical clustering using **complete linkage** and plot the dendogram to visualize it.

Complete Linkage





(Dendogram)

Hierarchical Clustering - Example 2a

- Given the dataset with five points (0,0),(2,0),(5,0),(0,4),(4,4), run **complete-linkage** hierarchical clustering by hand (you can use a calculator). Use L1 distance (Manhattan distance). For each iteration, show the cluster membership of each point.

Hierarchical Clustering - Example 2b

- Given the dataset with five points (0,0),(2,0),(5,0),(0,4),(4,4), run **single-linkage** hierarchical clustering by hand (you can use a calculator). Use L1 distance (Manhattan distance). For each iteration, show the cluster membership of each point.

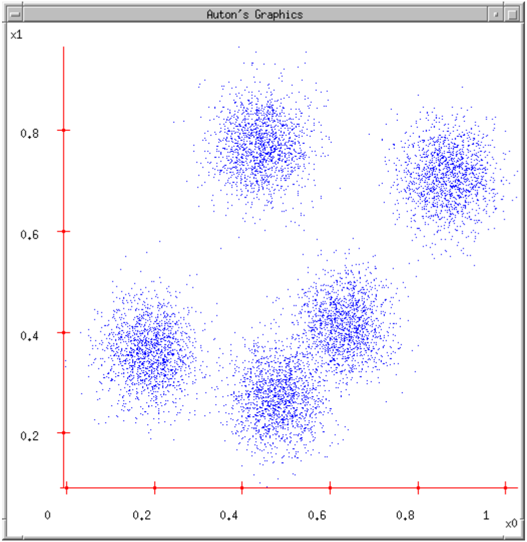
K-means Clustering

K-MEANS CLUSTERING - 1

- Very popular clustering method
- Don't confuse it with the k-NN classifier
- Input:
  - A dataset $x_1, \ldots, x_n$ each point is a numerical feature vector
  - Assume the number of clusters, k, is given.
- Overall logic:
  - Randomly picking k positions as initial cluster centers (not necessarily a data point)
  - Each point finds which cluster center it is closest to (very much like 1-NN). The point belongs to that cluster.
  - Each cluster computes its new centroid, based on which points belong to it
  - And repeat until convergence (cluster centers no longer move)...

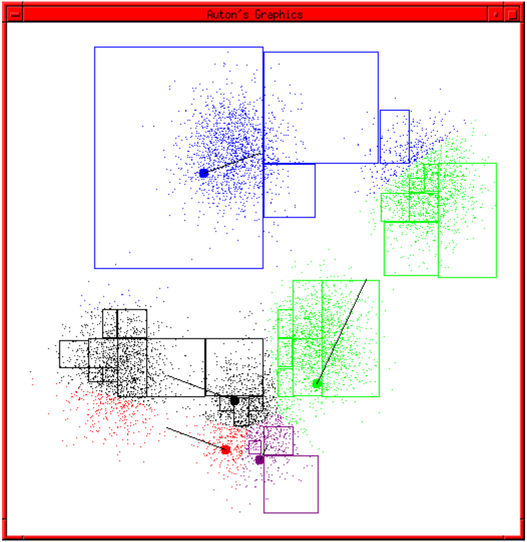
K-MEANS CLUSTERING - 2

- The dataset. Input $k = 5$



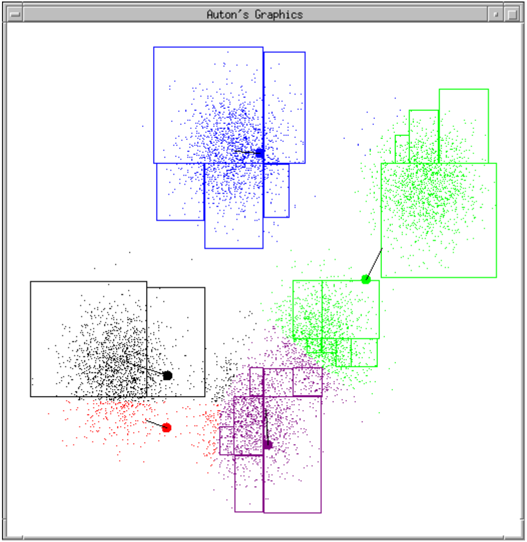
K-MEANS CLUSTERING - 3

- The dataset. Input $k = 5$



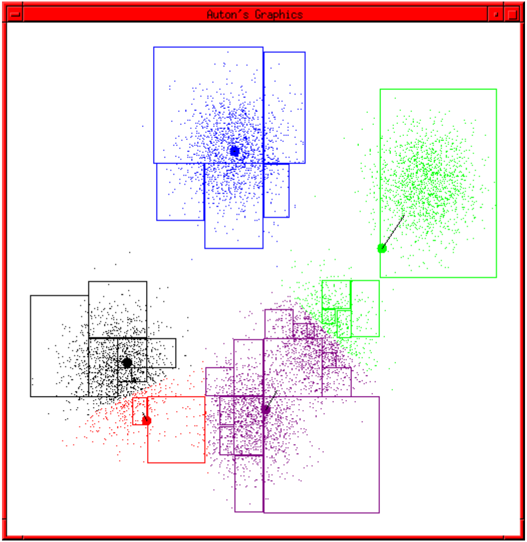
K-MEANS CLUSTERING - 4

- The dataset. Input $k = 5$



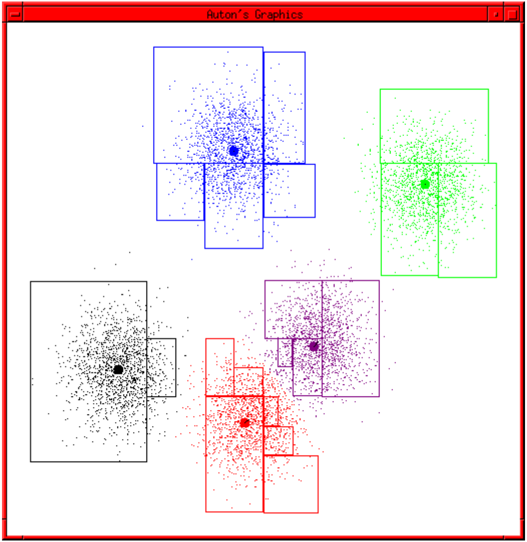
K-MEANS CLUSTERING - 5

- The dataset. Input $k = 5$



K-MEANS CLUSTERING - 6

- The dataset. Input $k = 5$. K-means stops!



K-MEANS CLUSTERING - 7

K-means Clustering Algorithm

Input: $x_1, \ldots, x_n,\ k$

Step 1: Select k cluster centers $c_1, \ldots, c_k$

Step 2: for each point x, determine its cluster: find the closest center in Euclidean space

Step 3: update all cluster centers as the centroids

$$c_i = \frac{\sum_{\{x \in \ cluster \ i\}} x}{SizeOf(cluster \ i)}$$

Repeat step 2, 3 until cluster centers no longer change

- What is k-means trying to optimize? $\Rightarrow$ total distortion over each data point
- Will k-means stop (converge)? $\Rightarrow$ yes. finite # of pts, finite cluster assignments
- Will it find a global or local optimum? $\Rightarrow$ not guaranteed to find global optimal
- How to pick starting cluster centers? $\Rightarrow$ various algorithms exists
- How many clusters should we use? $\Rightarrow$ domain knowledge?

DISTORTION

- Suppose for a point x, you replace its coordinates by the cluster center $c_{(x)}$ it belongs to (lossy compression)
- How far are you off? Measure it with squared Euclidean distance: $x(d)$ is the d-th feature dimension, $y(x)$ is the cluster ID that x is in.

$$\sum_{d=1,\ldots,D} \left[x(d) - c_{y(x)}(d)\right]^2$$

$\Rightarrow$ This is the distortion of a single point x. For the whole dataset, distortion is

$$\sum_{x} \sum_{d=1,\ldots,D} \left[x(d) - c_{y(x)}(d)\right]^2$$

- The minimization problem:

$$\min_{y(x_1)\ldots y(x_n), c_1(1)\ldots c_1(D),\ldots,c_k(1)\ldots c_k(D)} \sum_{x} \sum_{d=1,\ldots,D} \left[x(d) - c_{y(x)}(d)\right]^2$$

Python Implementations

EXAMPLE: CLUSTERING COLLEGES & UNIVERSITIES

- Cluster the Colleges and Universities data using the five numeric columns in the data set.

- Use the hierarchical method

	A	B	C	D	E		F	G
1	**Colleges and Universities**							
2								
3	**School**	**Type**	**Median SAT**	**Acceptance Rate**	**Expenditures/Student**		**Top 10% HS**	**Graduation %**
4	Amherst	Lib Arts	1315	22%	$	26,636	85	93
5	Barnard	Lib Arts	1220	53%	$	17,653	69	80
6	Bates	Lib Arts	1240	36%	$	17,554	58	88
7	Berkeley	University	1176	37%	$	23,665	95	68
8	Bowdoin	Lib Arts	1300	24%	$	25,703	78	90
9	Brown	University	1281	24%	$	24,201	80	90
10	Bryn Mawr	Lib Arts	1255	56%	$	18,847	70	84
11	Cal Tech	University	1400	31%	$	102,262	98	75

Example: Clustering Colleges & Universities

- Hierarchical clustering results for clusters 3 and 4

Cluster	School	Type	Median SAT	Acceptance Rate	Expenditures/ Student	Top 10% HS	Graduation %
3	Berkeley	University	1176	37%	$23,665	95	68
3	UCLA	University	1142	43%	$26,859	96	61
3	UNC	University	1109	32%	$19,684	82	73
4	Cal Tech	University	1400	31%	$102,262	98	75

- Schools in cluster 3 appear similar.

  Cluster 4 has considerably higher Median SAT and Expenditures/Student.

Clustering Implementation - 1

```
# sklearn clustering libraries
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import KMeans

# numpy
import numpy as np

# plotting library
import matplotlib.pyplot as plt

# random data generation library
from sklearn.datasets.samples_generator import make_blobs

X, y_true = make_blobs(n_samples=300, centers=4,
   cluster_std=0.60, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50) # plot pts
```

Clustering Implementation - 2

```
# hieararchical clustering via scikit-learn
hc = AgglomerativeClustering(n_clusters=4)
hc.fit(X)
y_hc = hc.fit_predict(X) # hierarchical.labels_

plt.scatter(X[:, 0], X[:, 1], c=y_hc, s=50, cmap='viridis')

# scipy implementation for hierarchical clustering
from scipy.cluster.hierarchy import linkage, fcluster, dendrogram

link_c = linkage(X, method='ward')
plt.figure()
dendrogram(link_c) # plot dendogram (binary clustering tree)
plt.show()

# plot clusters
max_d = 10 # max_distance btw clusters
lclusters = fcluster(link_c, max_d, criterion='distance')
plt.scatter(X[:, 0], X[:, 1], c=lclusters, s=50, cmap='viridis')
print(np.unique(lclusters)) # print cluster ids
```

Clustering Implementation - 3

```
# kmeans clustering via scikit-learn
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

# plot clusters
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black',
   s=200, alpha=0.5);
```

Clustering Implementation - 4

```
# normalizing the data
from sklearn.metrics.pairwise import euclidean_distances
# for scaling numpy array
from sklearn.preprocessing import StandardScaler

X = np.array([[1,2,100],[4,3,50],[1,1,75]])
# array([[  1,   2, 100],
#        [  4,   3,  50],
#        [  1,   1,  75]])

np.around(euclidean_distances(X), 2)
# array([[ 0.  , 50.1 , 25.02],
#        [50.1 ,  0.  , 25.26],
#        [25.02, 25.26,  0.  ]])

euclidean_distances(StandardScaler().fit_transform(X))
# array([[0.  , 3.46, 1.73],
#        [3.46, 0.  , 3.46],
#        [1.73, 3.46, 0.  ]])
```

Ryerson University

Clustering Implementation - 5

```
# clustering over pandas dataframe
my_data = pd.read_csv('universities.csv')
my_data.set_index('School', inplace=True) # set column as index

my_data["Type"] = my_data["Type"].astype("category").cat.codes

mat = my_data.values # convert dataframe to matrix
dist_mat = euclidean_distances(mat)
norm_mat = StandardScaler().fit_transform(mat)
dist_mat_norm = euclidean_distances(norm_mat)

km = sklearn.cluster.KMeans(n_clusters=4, init='random',
   n_init=1, verbose=0)
km.fit(norm_mat)
# Get cluster assignment labels
labels = km.labels_

# Format results as a DataFrame
results = pd.DataFrame([my_data.index,labels]).T
print(results)
```