İSTANBUL ŞEHİR ÜNİVERSİTY

# ENGR 212: Programming Practice

## *Week 8*

# Structuring

# &

# Visualization

İSTANBUL ŞEHİR ÜNİVERSİTY

# Implementation details

İSTANBUL ŞEHİR ÜNIVERSITY

```
class bicluster:
  def __init__(self,vec,left=None,right=None,distance=0.0,id=None):
    self.left=left
    self.right=right
    self.vec=vec
    self.id=id
    self.distance=distance
```

# Clustering - I

İSTANBUL ŞEHİR ÜNİVERSİTY

```
def readfile(filename):
  lines=[line for line in file(filename)]

  # First line is the column titles
  colnames=lines[0].strip( ).split('\t')[1:]
  rownames=[]
  data=[]
  for line in lines[1:]:
    p=line.strip( ).split('\t')
    # First column in each row is the rowname
    rownames.append(p[0])
    # The data for this row is the remainder of the row
    data.append([float(x) for x in p[1:]])
  return rownames,colnames,data
```

# Clustering - II

İSTANBUL ŞEHİR ÜNİVERSİTY

```
def hcluster(rows,distance=pearson):
  distances={}
  currentclustid=-1

  # Clusters are initially just the rows
  clust=[bicluster(rows[i],id=i) for i in range(len(rows))]

  while len(clust)>1:
    lowestpair=(0,1)
    closest=distance(clust[0].vec,clust[1].vec)

    # loop through every pair looking for the smallest distance
    for i in range(len(clust)):
      for j in range(i+1,len(clust)):
        # distances is the cache of distance calculations
        if (clust[i].id,clust[j].id) not in distances:
          distances[(clust[i].id,clust[j].id)]=distance(clust[i].vec,clust[j].vec)

        d=distances[(clust[i].id,clust[j].id)]

        if d<closest:
          closest=d
          lowestpair=(i,j)
```

# Clustering - III

İSTANBUL ŞEHİR ÜNİVERSİTY

```
    # calculate the average of the two clusters
    mergevec=[
    (clust[lowestpair[0]].vec[i]+clust[lowestpair[1]].vec[i])/2.0
    for i in range(len(clust[0].vec))]

    # create the new cluster
    newcluster=bicluster(mergevec,left=clust[lowestpair[0]],
                         right=clust[lowestpair[1]],
                         distance=closest,id=currentclustid)

    # cluster ids that weren't in the original set are negative
    currentclustid-=1
    del clust[lowestpair[1]]
    del clust[lowestpair[0]]
    clust.append(newcluster)

return clust[0]
```

# K-Means Clustering

İSTANBUL ŞEHİR ÜNİVERSİTY

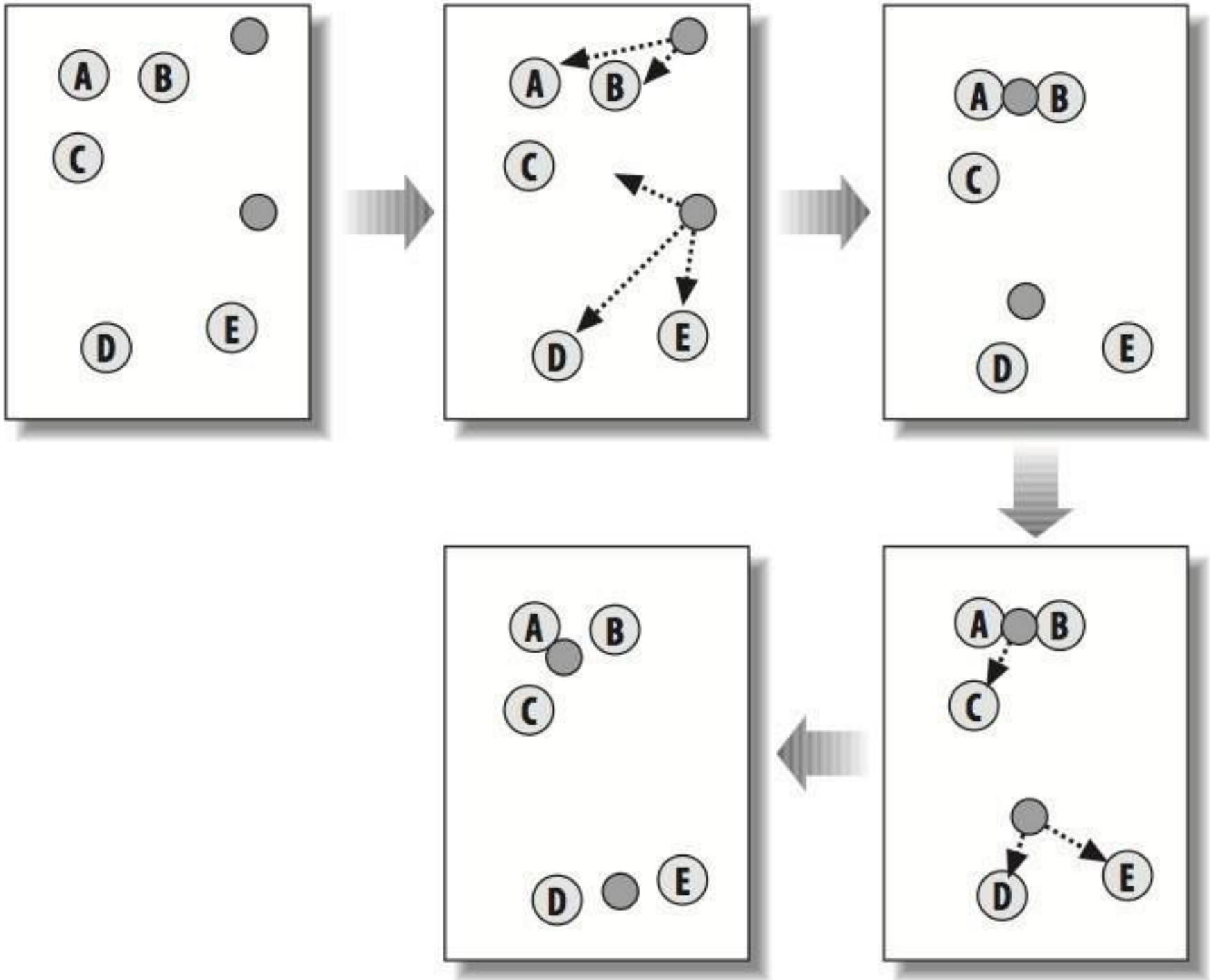
- Hierarchical clustering:
  - the tree view requires a good eye to inspect :-)
  - computationally intensive: quadratic.

- K-means clustering
  - apriori info: how many distinct clusters to generate.

# K-Means Clustering

İSTANBUL ŞEHİR ÜNİVERSİTY

- k randomly placed centroids (points in space that represent the center of the cluster).
- Assign every item to the nearest one.
- After the assignment, the centroids are moved to the average location of all the nodes assigned to them, and the assignments are redone.
- This process repeats until the assignments stop changing.


A
B
C
D
E
A
B
C
D
E
A
B
C
D
E
A
B
C
D
E
A
B
C
D
E

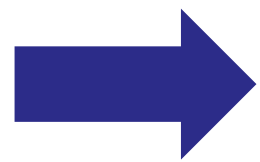
# K-means clustering in Python

İSTANBUL ŞEHİR ÜNİVERSİTY

```
import random

def kcluster(rows,distance=pearson,k=4):
  # Determine the minimum and maximum values for each point
  ranges=[(min([row[i] for row in rows]),max([row[i] for row in rows]))
  for i in range(len(rows[0]))]

  # Create k randomly placed centroids
  clusters=[[random.random( )*(ranges[i][1]-ranges[i][0])+ranges[i][0]
  for i in range(len(rows[0]))] for j in range(k)]
```



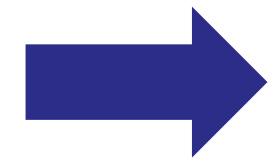
# K-means clustering in Python

İSTANBUL ŞEHİR ÜNİVERSİTY

```
lastmatches=None
for t in range(100):
  print 'Iteration %d' % t
  bestmatches=[[] for i in range(k)]

  # Find which centroid is the closest for each row
  for j in range(len(rows)):
    row=rows[j]
    bestmatch=0
    for i in range(k):
      d=distance(clusters[i],row)
      if d<distance(clusters[bestmatch],row): bestmatch=i
    bestmatches[bestmatch].append(j)

  # If the results are the same as last time, this is complete
  if bestmatches==lastmatches: break
  lastmatches=bestmatches
```



# K-means clustering in Python

İSTANBUL ŞEHİR ÜNİVERSİTY

```
  # If the results are the same as last time, this is complete
  if bestmatches==lastmatches: break
  lastmatches=bestmatches

  # Move the centroids to the average of their members
  for i in range(k):
    avgs=[0.0]*len(rows[0])
    if len(bestmatches[i])>0:
      for rowid in bestmatches[i]:
        for m in range(len(rows[rowid])):
          avgs[m]+=rows[rowid][m]
      for j in range(len(avgs)):
        avgs[j]/=len(bestmatches[i])
      clusters[i]=avgs

return bestmatches
```

# Play with k-means clustering

İSTANBUL ŞEHİR ÜNİVERSİTY

- /Week 08/Code folder on LMS contains all the code. Download them into a directory.

```
>>> import clusters
>>> blognames, words, data = clusters.readfile('blogdata.txt')
>>> kclust=clusters.kcluster(data,k=10)
>>> [blognames[r] for r in kclust [0]]
>>> [blognames[r] for r in kclust [1]]
```