K-means Clustering

葉建華

jhyeh@mail.au.edu.tw

http://jhyeh.csie.au.edu.tw/

Scenario

- In 2000 and 2004 US presidential elections were very close
  - Largest 50.7%, lowest 47.9%
  - Small groups of voters switched sides will change the election
  - These groups are not huge, but big enough
- Find them and appeal to them
  - Use clustering!
  - How?

Clustering

- A major type of unsupervised learning
  - Put similar things together
- One of the most common algorithm: k-means
  - Finds k unique clusters
  - The center of each cluster is the mean of the values in that cluster

K-means Clustering

k-means clustering

Pros: Easy to implement

Cons: Can converge at local minima; slow on very large datasets

Works with: Numeric values

- Find k unique clusters, k is user-defined
- Each cluster is described by a single point called centroid: center of the cluster

Pseudo-Code

```
Create k points for starting centroids (often randomly)
While any point has changed cluster assignment
    for every point in our dataset:
        for every centroid
            calculate the distance between the centroid and point
        assign the point to the cluster with the lowest distance
    for every cluster calculate the mean of the points in that cluster
        assign the centroid to the mean
```

General Approach

General approach to k-means clustering

1. Collect: Any method.
2. Prepare: Numeric values are needed for a distance calculation, and nominal values can be mapped into binary values for distance calculations.
3. Analyze: Any method.
4. Train: Doesn't apply to unsupervised learning.
5. Test: Apply the clustering algorithm and inspect the results. Quantitative error measurements such as sum of squared error (introduced later) can be used.
6. Use: Anything you wish. Often, the clusters centers can be treated as representative data of the whole cluster to make decisions.

Distance Measure

- Many kinds of distance measure can be applied
  - Eucledian distance
  - Pearson similarity
  - Cosine similarity
  - etc.

Support Functions

Listing 10.1 k-means support functions

```
from numpy import *

def loadDataSet(fileName):
    dataMat = []
    fr = open(fileName)
    for line in fr.readlines():
        curLine = line.strip().split('\t')
        fltLine = map(float,curLine)
        dataMat.append(fltLine)
    return dataMat

def distEclud(vecA, vecB):
    return sqrt(sum(power(vecA - vecB, 2)))

def randCent(dataSet, k):
    n = shape(dataSet)[1]
    centroids = mat(zeros((k,n)))
    for j in range(n):
        minJ = min(dataSet[:,j])
        rangeJ = float(max(dataSet[:,j]) - minJ)
        centroids[:,j] = minJ + rangeJ * random.rand(k,1)
    return centroids
```

Create cluster centroids

Core Algorithm

Listing 10.2 The k-means clustering algorithm

```
def kMeans(dataSet, k, distMeas=distEclud, createCent=randCent):
    m = shape(dataSet)[0]
    clusterAssment = mat(zeros((m,2)))
    centroids = createCent(dataSet, k)
    clusterChanged = True
    while clusterChanged:
        clusterChanged = False
        for i in range(m):
            minDist = inf; minIndex = -1
            for j in range(k):
                distJI = distMeas(centroids[j,:],dataSet[i,:])
                if distJI < minDist:
                    minDist = distJI; minIndex = j
            if clusterAssment[i,0] != minIndex: clusterChanged = True
            clusterAssment[i,:] = minIndex,minDist**2
        print centroids
        for cent in range(k):
            ptsInClust = dataSet[nonzero(clusterAssment[:,0].A==cent)[0]]
            centroids[cent,:] = mean(ptsInClust, axis=0)
    return centroids, clusterAssment
```

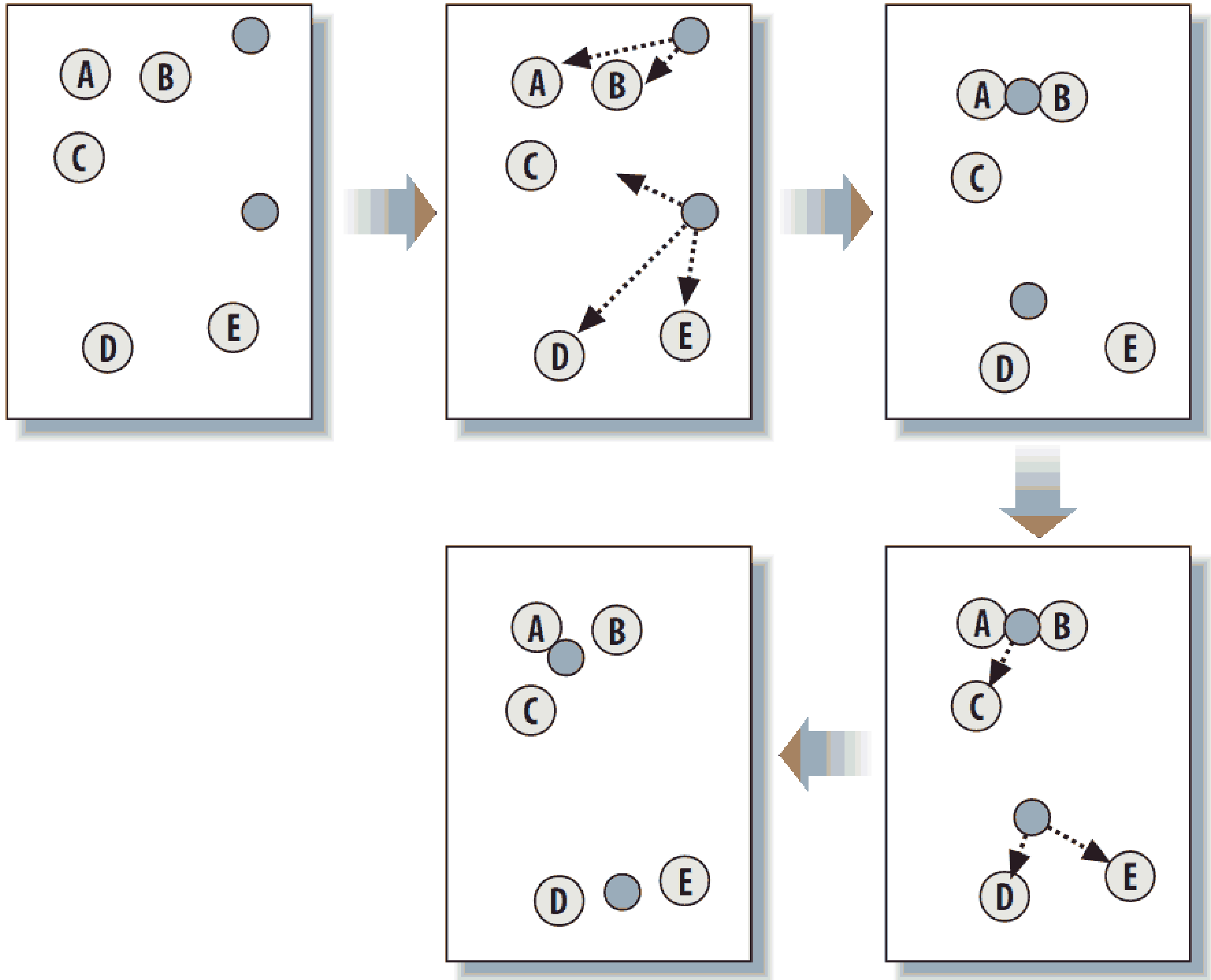
1 Find the closest centroid

2 Update centroid location

Flow of K-means

- Begins with k randomly placed centroids
- Assigns every item to the nearest one (centroid)
- Centroid relocation
- Redo the assignments
- Stop on stable assignments

Example Flow



Cluster Result Example

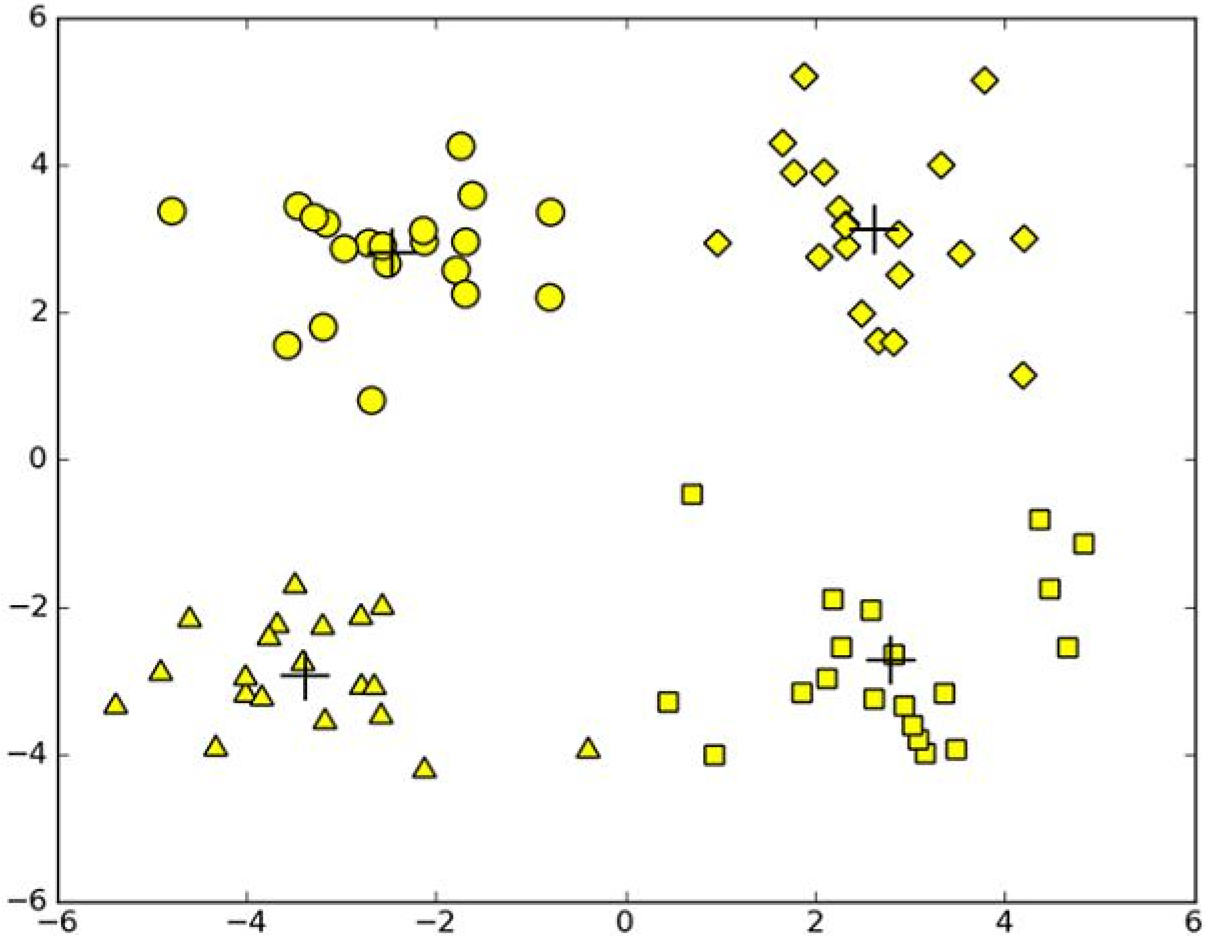


Figure 10.1 Clusters resulting from k-means clustering. After three iterations, the algorithm converged on these results. Data points with similar shapes are in similar clusters. The cluster centers are marked with a cross.

Improving K-means

- How does the user know that k is the right number?
- How do you know that the clusters are good clusters?

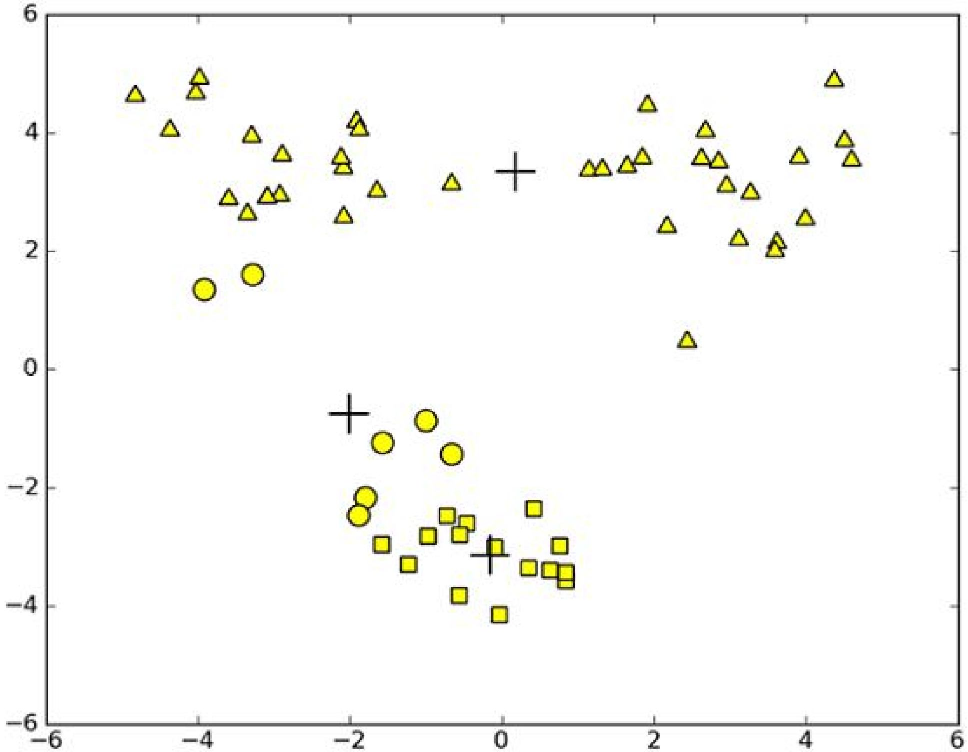


Figure 10.2 Cluster centroids incorrectly assigned because of poor initialization with random initialization in k-means. Additional postprocessing is required to clean up the clusters.

Cluster Quality

- SSE(sum of squared error)
  - A lower SSE means that points are closer to their centroids, better job!
  - The error is squared: places more emphasis on points far from the centroid
- But how about increasing cluster number?
  - Bigger k will reduce SSE

Cluster Postprocessing

- Possible postprocess the clusters
  - Cluster with the highest SSE: split into two clusters
  - Merging two centroids that increase total SSE the least
  - Merging two clusters and then calculating the total SSE

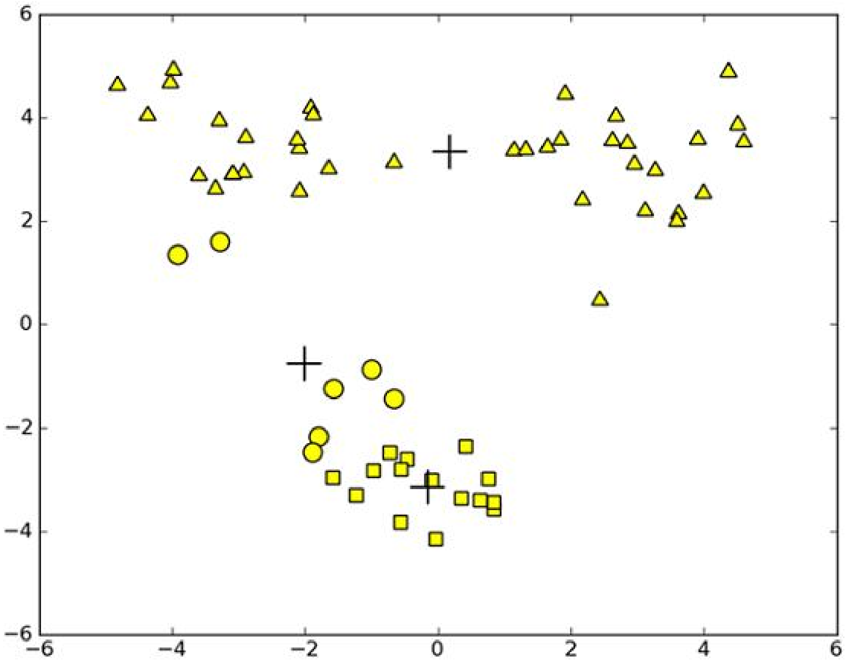


Figure 10.2 Cluster centroids incorrectly assigned because of poor initialization with random initialization in k-means. Additional postprocessing is required to clean up the clusters.

Bisecting K-means

Start with all the points in one cluster
While the number of clusters is less than k
    for every cluster
        measure total error
        perform k-means clustering with k=2 on the given cluster
        measure total error after k-means has split the cluster in two
    choose the cluster split that gives the lowest error and commit this split

Bisecting K-means

Listing 10.3 The bisecting k-means clustering algorithm

```
def biKmeans(dataSet, k, distMeas=distEclud):
    m = shape(dataSet)[0]
    clusterAssment = mat(zeros((m,2)))
    centroid0 = mean(dataSet, axis=0).tolist()[0]
    centList =[centroid0]
    for j in range(m):
        clusterAssment[j,1] = distMeas(mat(centroid0), dataSet[j,:])**2
    while (len(centList) < k):
        lowestSSE = inf
        for i in range(len(centList)):
            ptsInCurrCluster =\
                    dataSet[nonzero(clusterAssment[:,0].A==i)[0],:]
            centroidMat, splitClustAss = \
                    kMeans(ptsInCurrCluster, 2 , distMeas)
            sseSplit = sum(splitClustAss[:,1])
            sseNotSplit = \
              sum(clusterAssment[nonzero(clusterAssment[:,0].A!=i)[0],1])
            print "sseSplit, and notSplit: ",sseSplit,sseNotSplit
            if (sseSplit + sseNotSplit) < lowestSSE:
                bestCentToSplit = i
                bestNewCents = centroidMat
                bestClustAss = splitClustAss.copy()
                lowestSSE = sseSplit + sseNotSplit
        bestClustAss[nonzero(bestClustAss[:,0].A == 1)[0],0] =\
                             len(centList)
        bestClustAss[nonzero(bestClustAss[:,0].A == 0)[0],0] =\
                             bestCentToSplit
        print 'the bestCentToSplit is: ',bestCentToSplit
        print 'the len of bestClustAss is: ', len(bestClustAss)
        centList[bestCentToSplit] = bestNewCents[0,:]
        centList.append(bestNewCents[1,:])
        clusterAssment[nonzero(clusterAssment[:,0].A == \
                             bestCentToSplit)[0],:]= bestClustAss
    return mat(centList), clusterAssment
```

1. Initially create one cluster
2. Try splitting every cluster
3. Update the cluster assignments

Bisecting Result

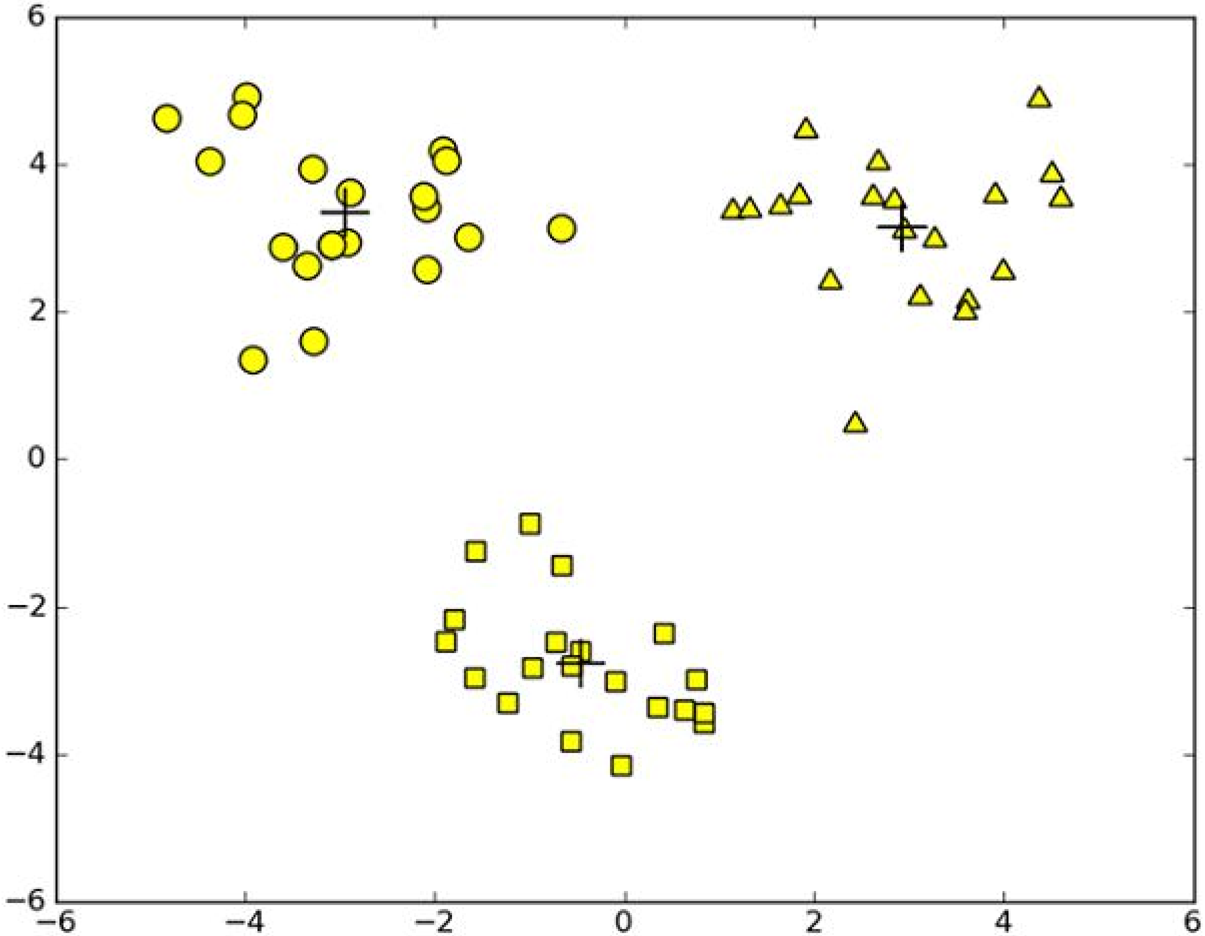


Figure 10.3 Cluster assignment after running the bisecting k-means algorithm. The cluster assignment always results in good clusters.

Example: Map Point Clustering

- Clustering points on a map using Yahoo! PlaceFinder API

Example: using bisecting k-means on geographic data

1. Collect: Use the Yahoo! PlaceFinder API to collect data.
2. Prepare: Remove all data except latitude and longitude.
3. Analyze: Use Matplotlib to make 2D plots of our data, with clusters and map.
4. Train: Doesn't apply to unsupervised learning.
5. Test: Use `biKmeans()`, developed in section 10.4.
6. Use: The final product will be your map with the clusters and cluster centers.

Example : Map Point Clustering

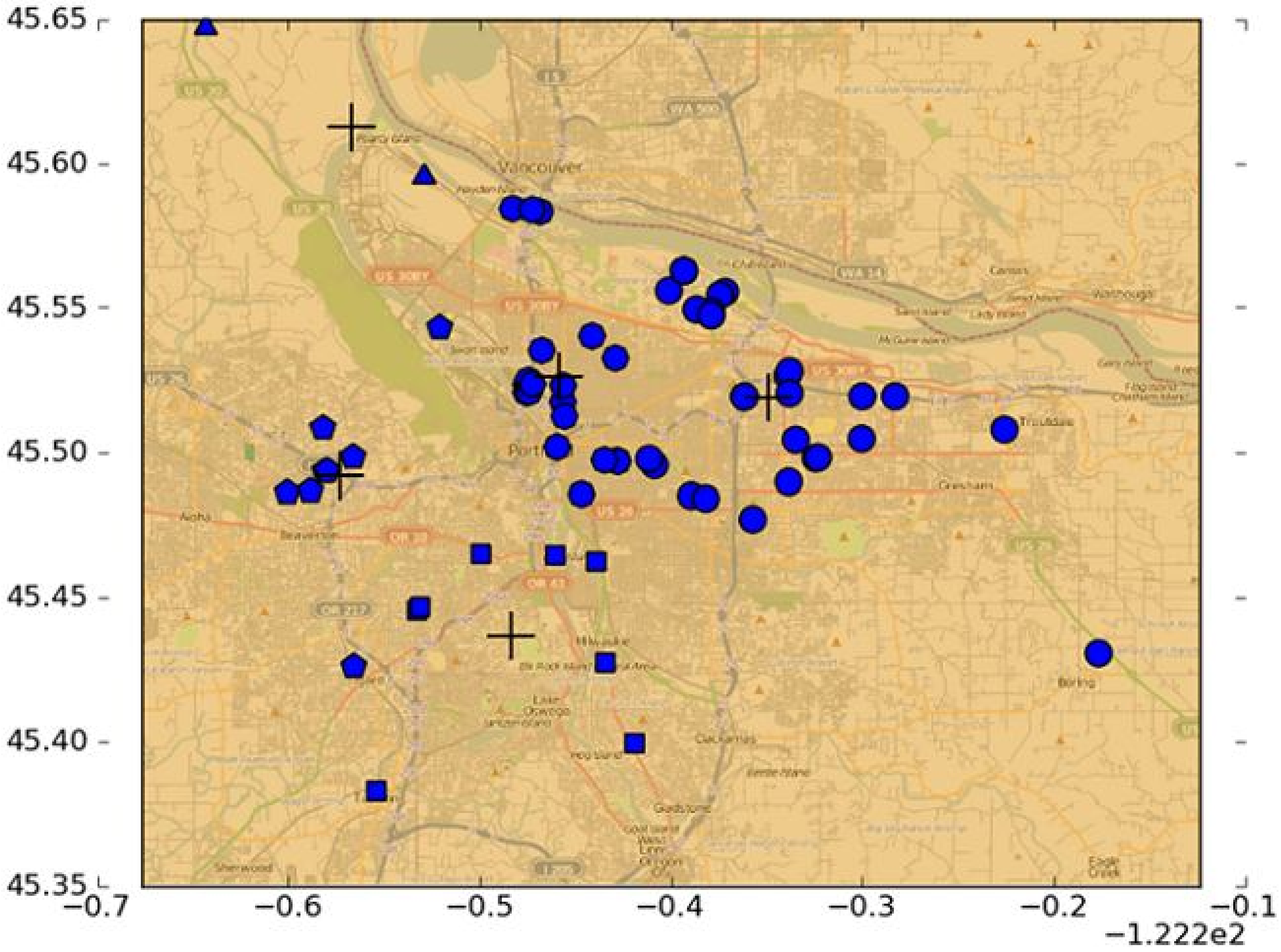


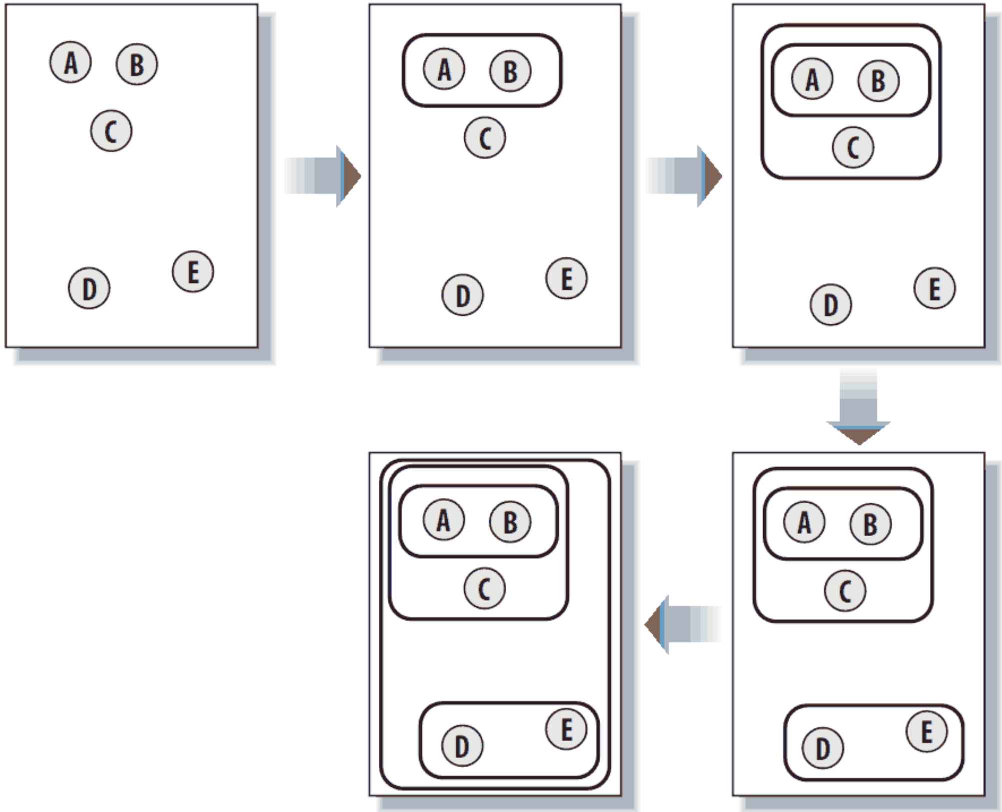
Figure 10.4 Clustering of nighttime entertainment locations in Portland, Oregon

Summary

- K-means and its derivatives aren't the only clustering algorithms
- Another type of clustering, known as hierarchical agglomerative clustering(HAC), is also a widely used clustering algorithm

Suppliment: HAC

- HAC = Hierarchical Agglomerative Clustering
- Build up a hierarchy of groups by continuously merging the two most similar groups
  - Again, similarity!



HAC Result: Dendrogram

- The dendrogram not only uses connections to show cluster items, also show how far apart the items were
  - eg. A-B versus D-E