Lecture 12: Clustering

Reading

- Chapter 23

Machine Learning Paradigm

- Observe set of examples: training data
- Infer something about process that generated that data
- Use inference to make predictions about previously unseen data: test data
- Supervised: given a set of feature/label pairs, find a rule that predicts the label associated with a previously unseen input
- *Unsupervised*: given a set of feature vectors (without labels) group them into "natural clusters"

Clustering Is an Optimization Problem

$$variability(c) = \sum_{e \in c} distance(mean(c), e)^2$$

$$dissimilarity(C) = \sum_{c \in C} variability(c)$$

- Why not divide variability by size of cluster?
  - Big and bad worse than small and bad
- Is optimization problem finding a C that minimizes *dissimilarity(C)*?
  - No, otherwise could put each example in its own cluster
- Need a constraint, e.g.,
  - Minimum distance between clusters
  - Number of clusters

Two Popular Methods

- Hierarchical clustering
- K-means clustering

Hiearchical Clustering

1. Start by assigning each item to a cluster, so that if you have N items, you now have N clusters, each containing just one item.

2. Find the closest (most similar) pair of clusters and merge them into a single cluster, so that now you have one fewer cluster.

3. Continue the process until all items are clustered into a single cluster of size N.

What does distance mean?

Linkage Metrics

- *Single-linkage:* consider the distance between one cluster and another cluster to be equal to the shortest distance from any member of one cluster to any member of the other cluster

- *Complete-linkage*: consider the distance between one cluster and another cluster to be equal to the greatest distance from any member of one cluster to any member of the other cluster

- *Average-linkage:* consider the distance between one cluster and another cluster to be equal to the average distance from any member of one cluster to any member of the other cluster

Example of Hierarchical Clustering

	BOS	**NY**	**CHI**	**DEN**	**SF**	**SEA**
BOS	0	206	963	1949	3095	2979
NY		0	802	1771	2934	2815
CHI			0	966	2142	2013
DEN				0	1235	1307
SF					0	808
SEA						0

{BOS} {NY} {CHI} {DEN} {SF} {SEA}

{BOS, NY} {CHI} {DEN} {SF} {SEA}

{BOS, NY, CHI} {DEN} {SF} {SEA}

{BOS, NY, CHI} {DEN} {SF, SEA}

{BOS, NY, CHI, DEN} {SF, SEA} Single linkage

or

{BOS, NY, CHI} {DEN, SF, SEA} Complete linkage

Clustering Algorithms

- Hierarchical clustering
  - Can select number of clusters using dendogram
  - Deterministic
  - Flexible with respect to linkage criteria
  - Slow
    - Naïve algorithm n^3
    - n^2 algorithms exist for some linkage criteria
- K-means a much faster greedy algorithm
  - Most useful when you know how many clusters you want

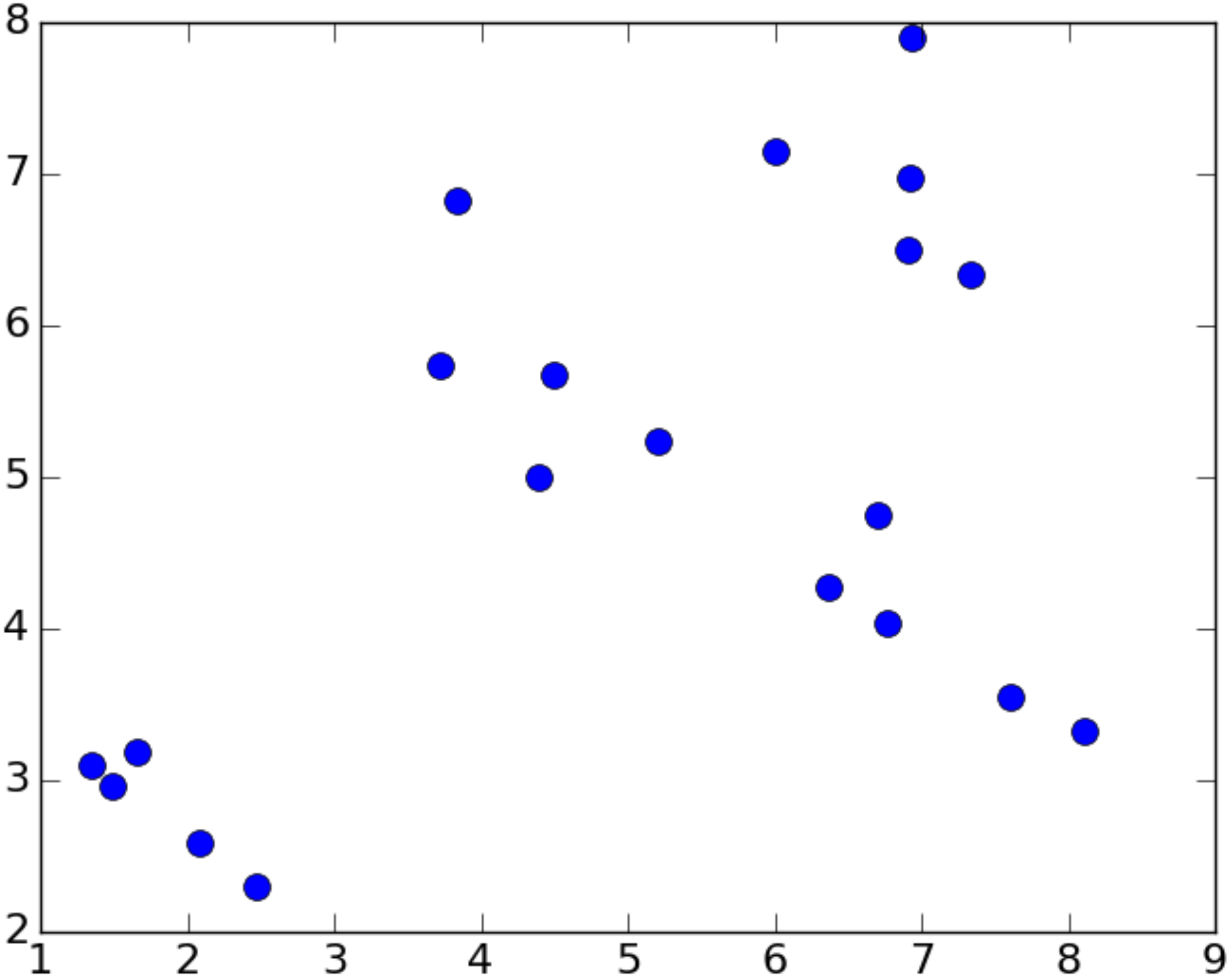
K-means Algorithm

```
randomly chose k examples as initial centroids
while true:
   create k clusters by assigning each
       example to closest centroid
   compute k new centroids by averaging
       examples in each cluster
   if centroids don’t change:
       break
```

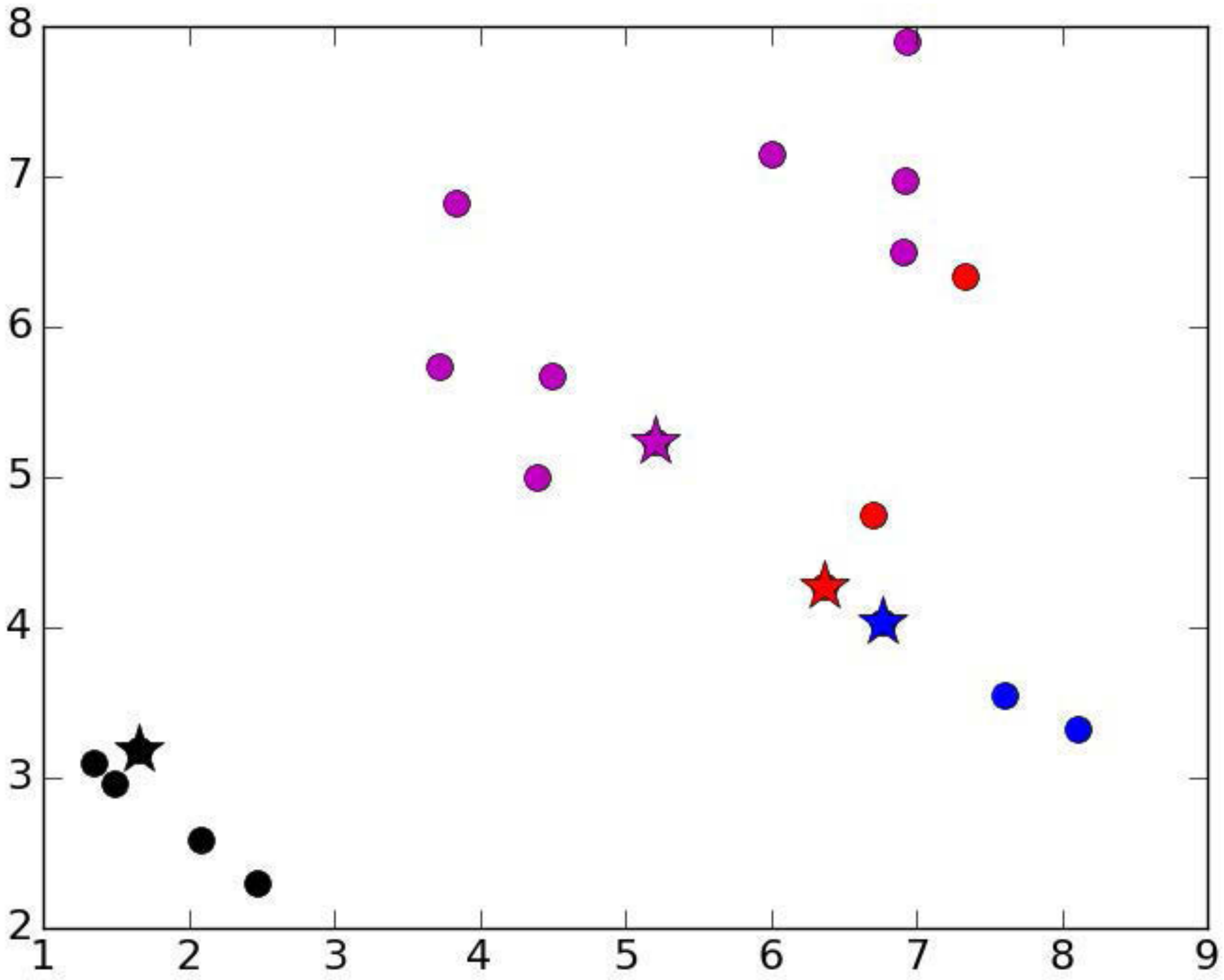
What is complexity of one iteration?

k*n*d, where n is number of points and d time required to compute the distance between a pair of points

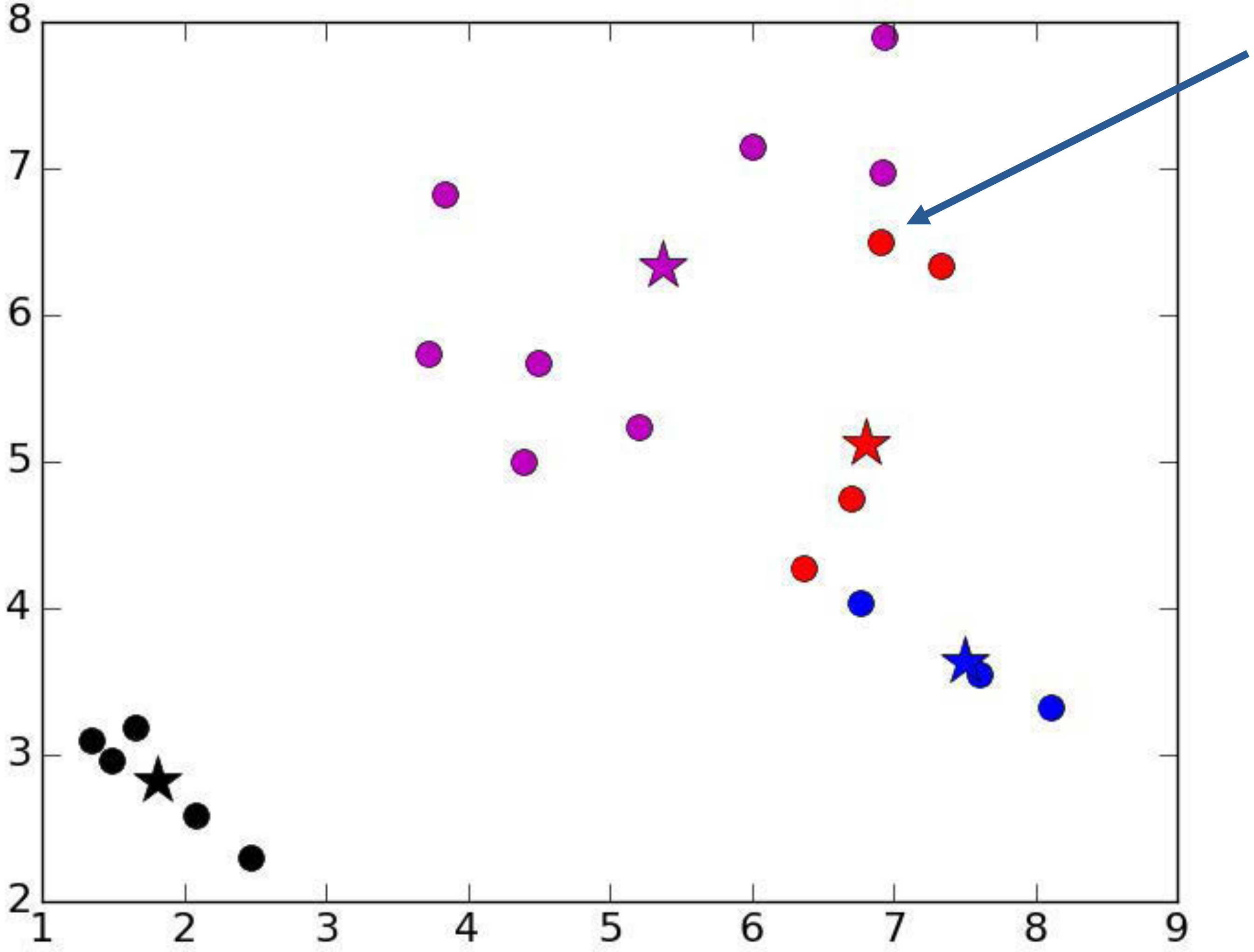
An Example



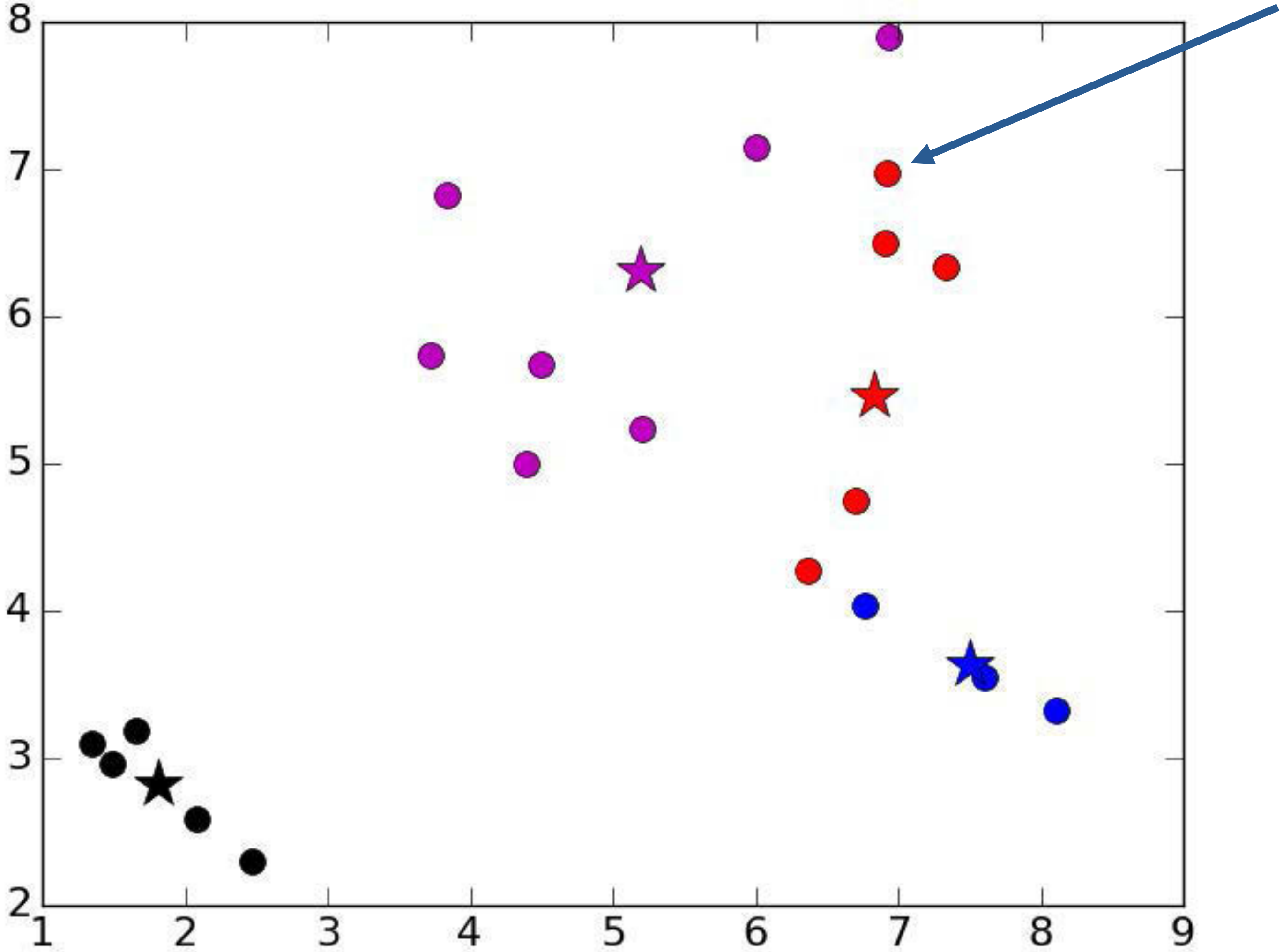
K = 4, Initial Centroids



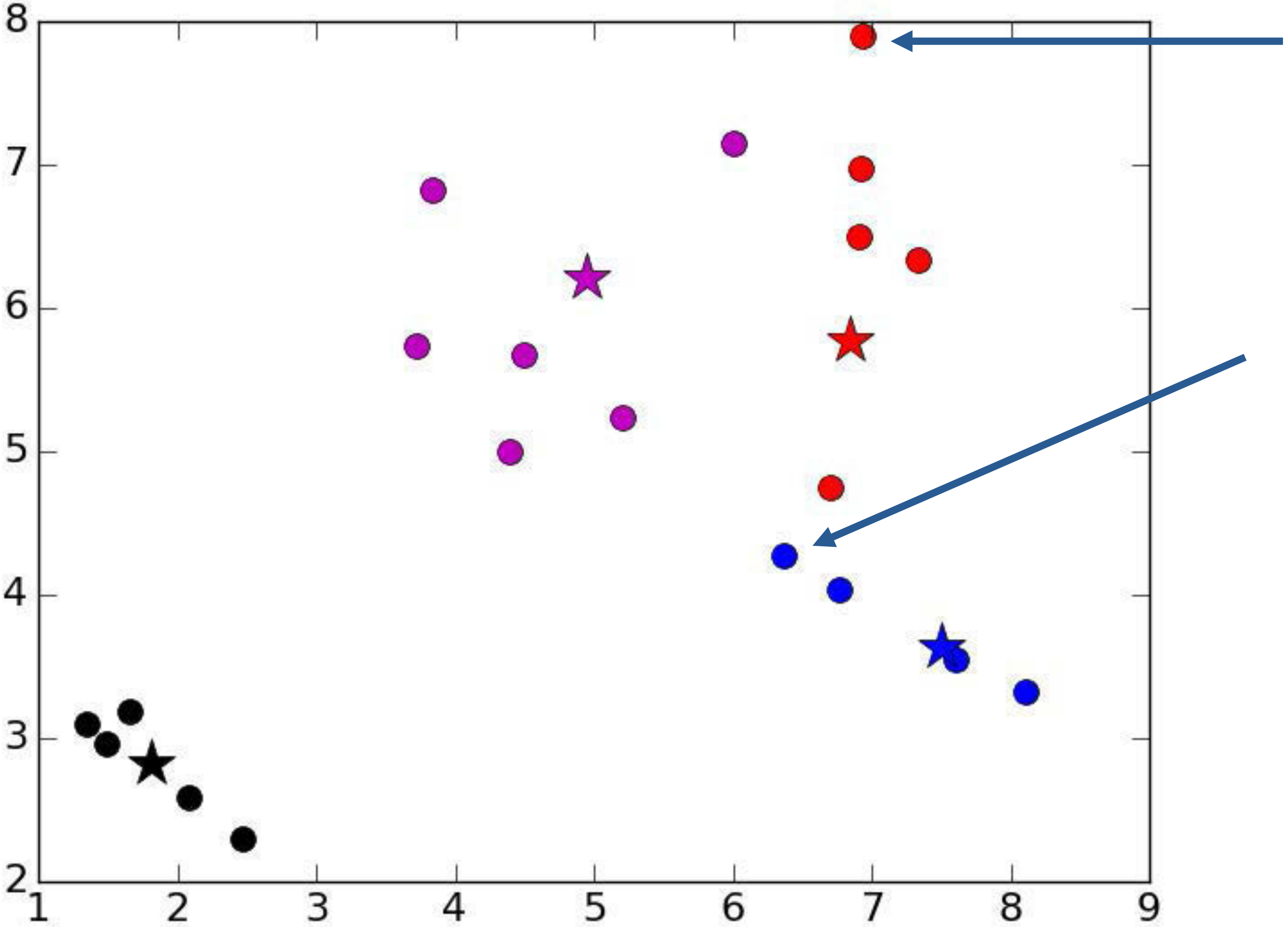
Iteration 1



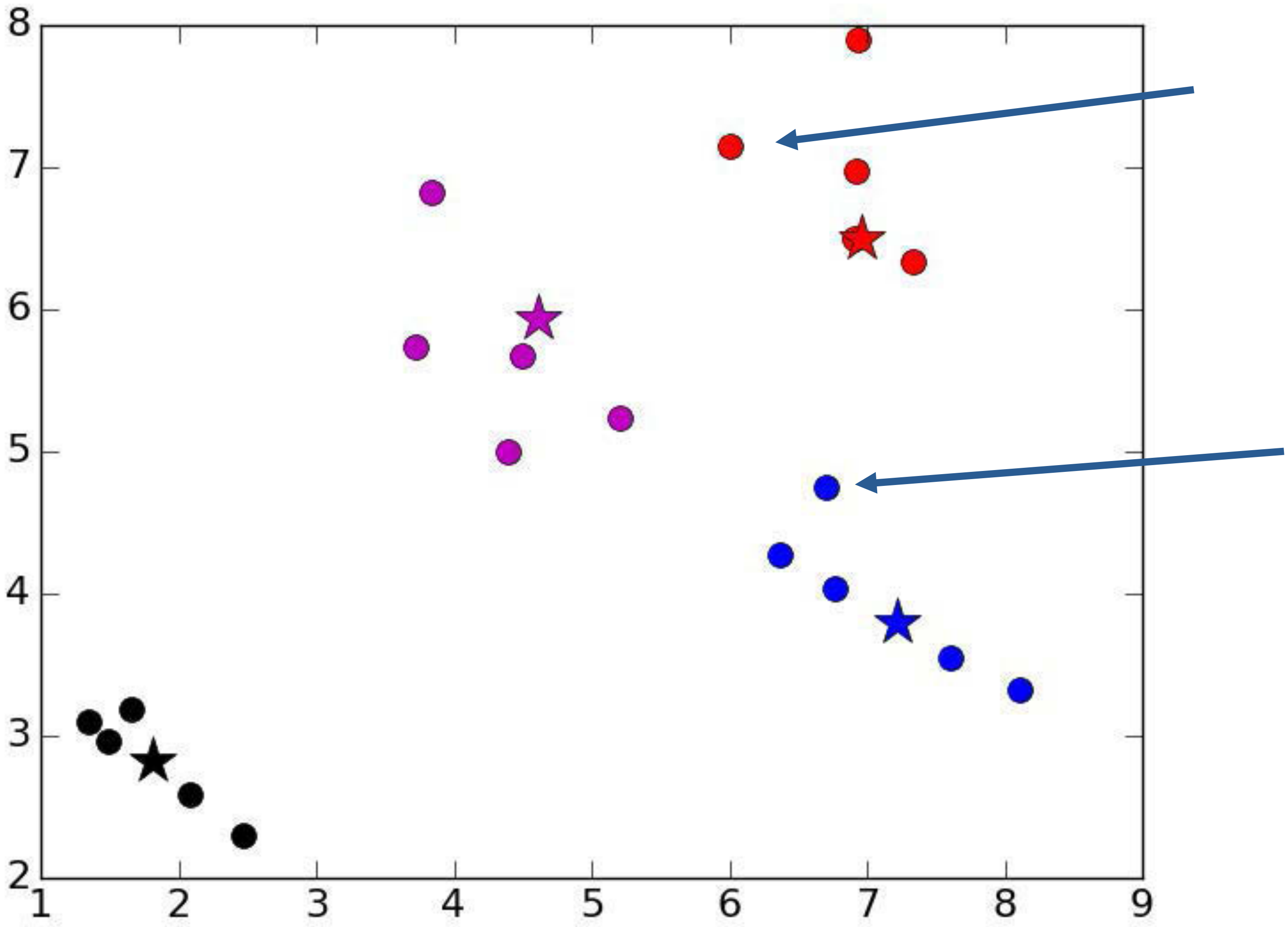
Iteration 2



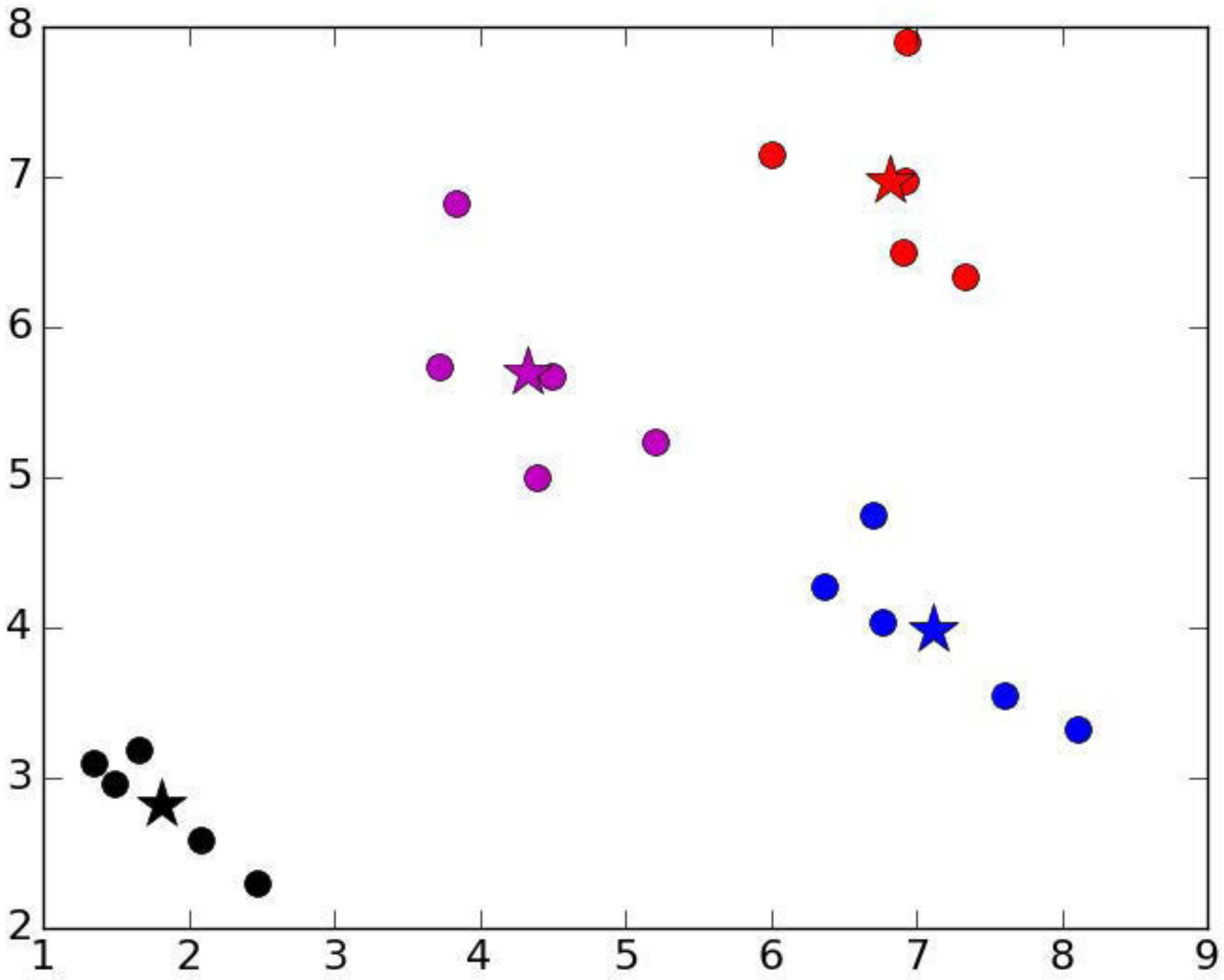
Iteration 3



Iteration 4

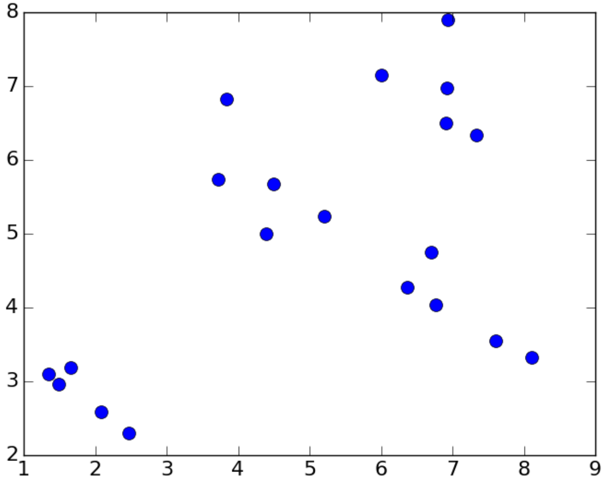


Iteration 5



Issues with k-means

- Choosing the “wrong” k can lead to strange results
  - Consider k = 3

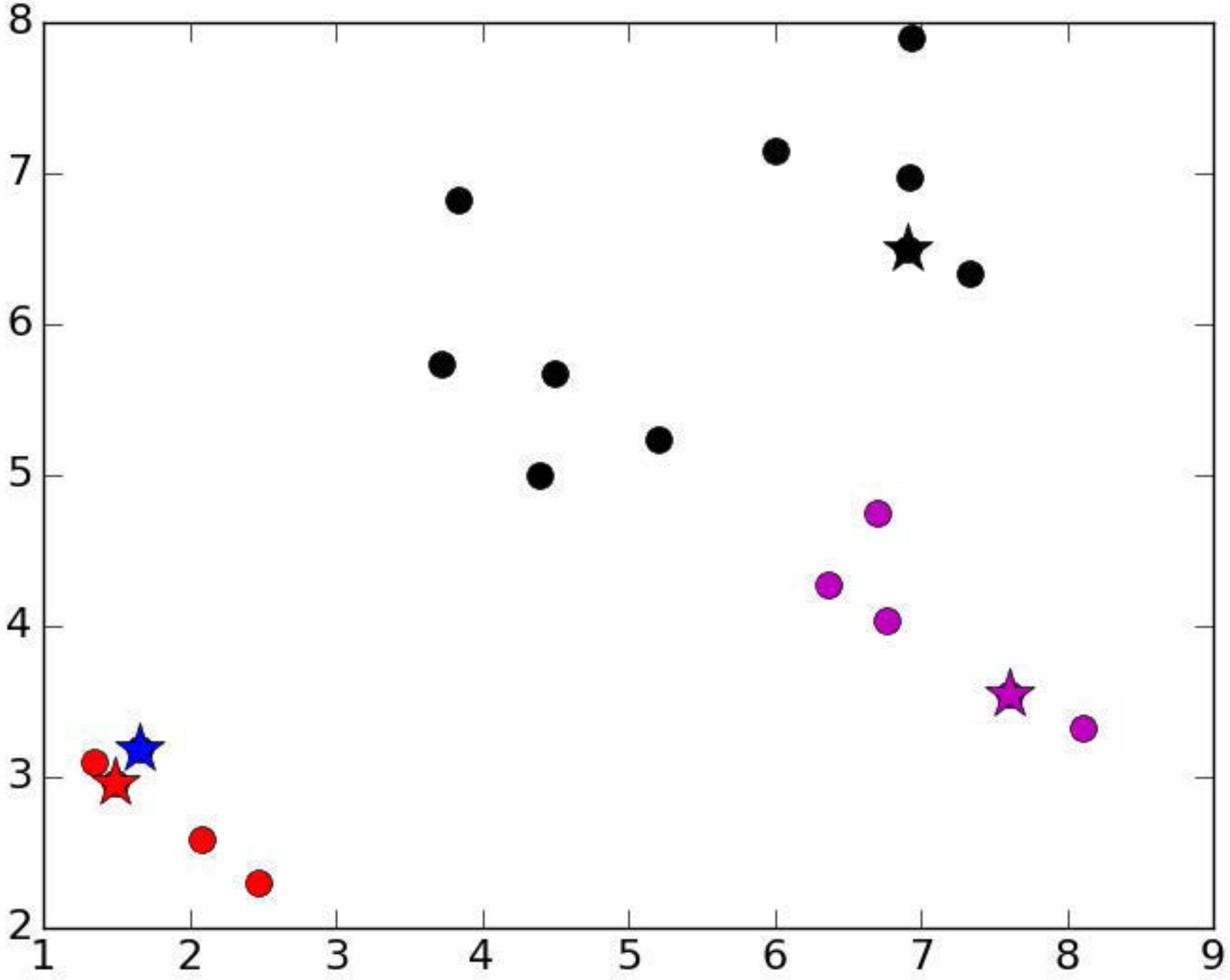


- Result can depend upon initial centroids
  - Number of iterations
  - Even final result
    - Greedy algorithm can find different local optimas

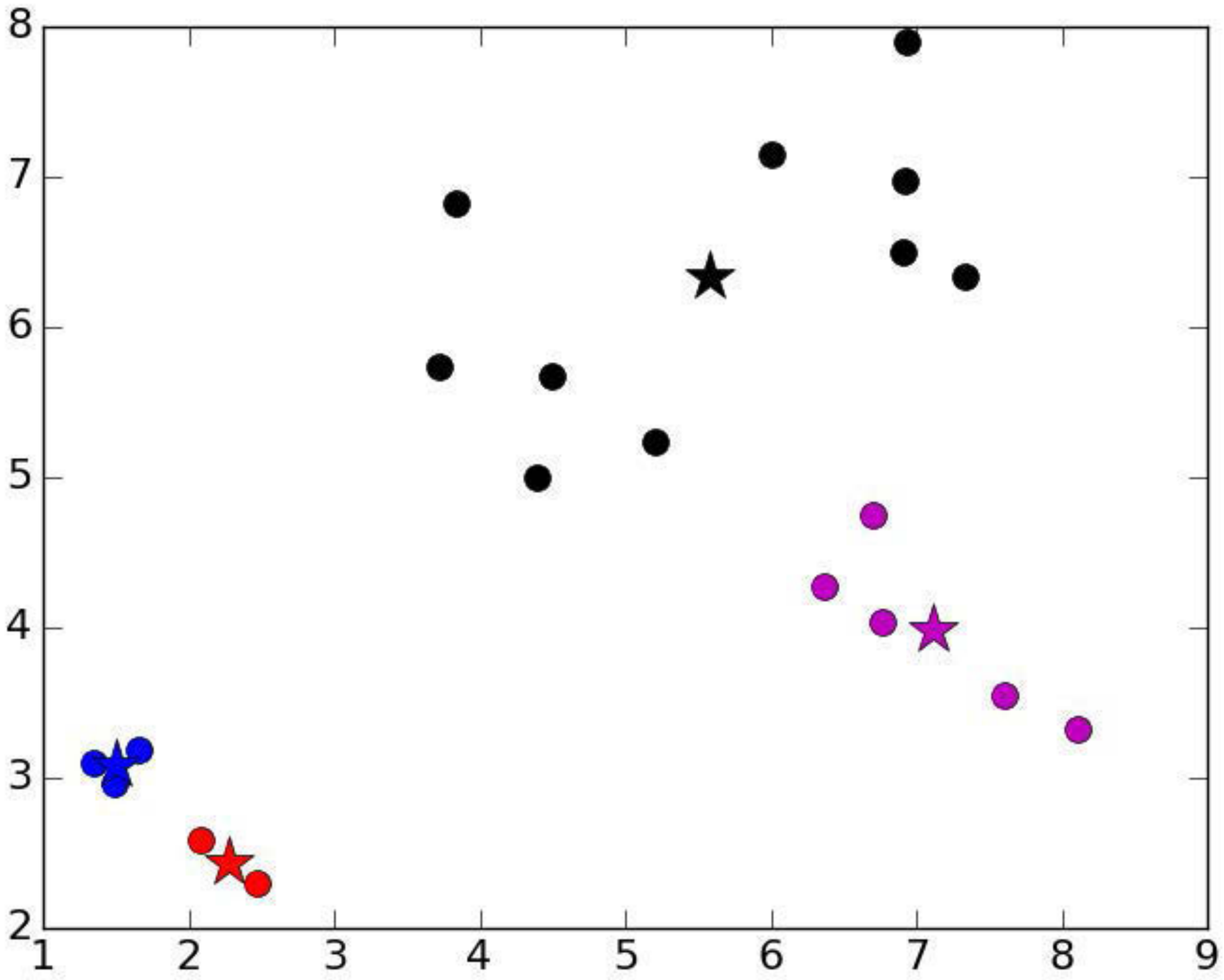
How to Choose K

- *A priori* knowledge about application domain
  - There are two kinds of people in the world: k = 2
  - There are five different types of bacteria: k = 5
- Search for a good k
  - Try different values of k and evaluate quality of results
  - Run hierarchical clustering on subset of data

Unlucky Initial Centroids



Converges On



Mitigating Dependence on Initial Centroids

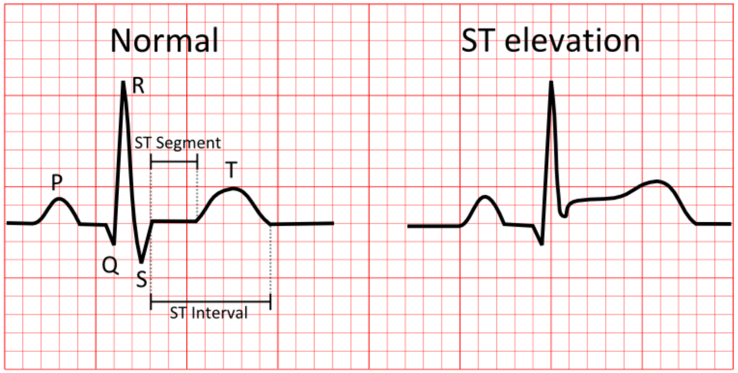
Try multiple sets of randomly chosen initial centroids

Select “best” result

```
best = kMeans(points)
for t in range(numTrials):
   C = kMeans(points)
   if dissimilarity(C) < dissimilarity(best):
      best = C
return best
```

An Example

- Many patients with 4 features each
  - Heart rate in beats per minute
  - Number of past heart attacks
  - Age
  - ST elevation (binary)

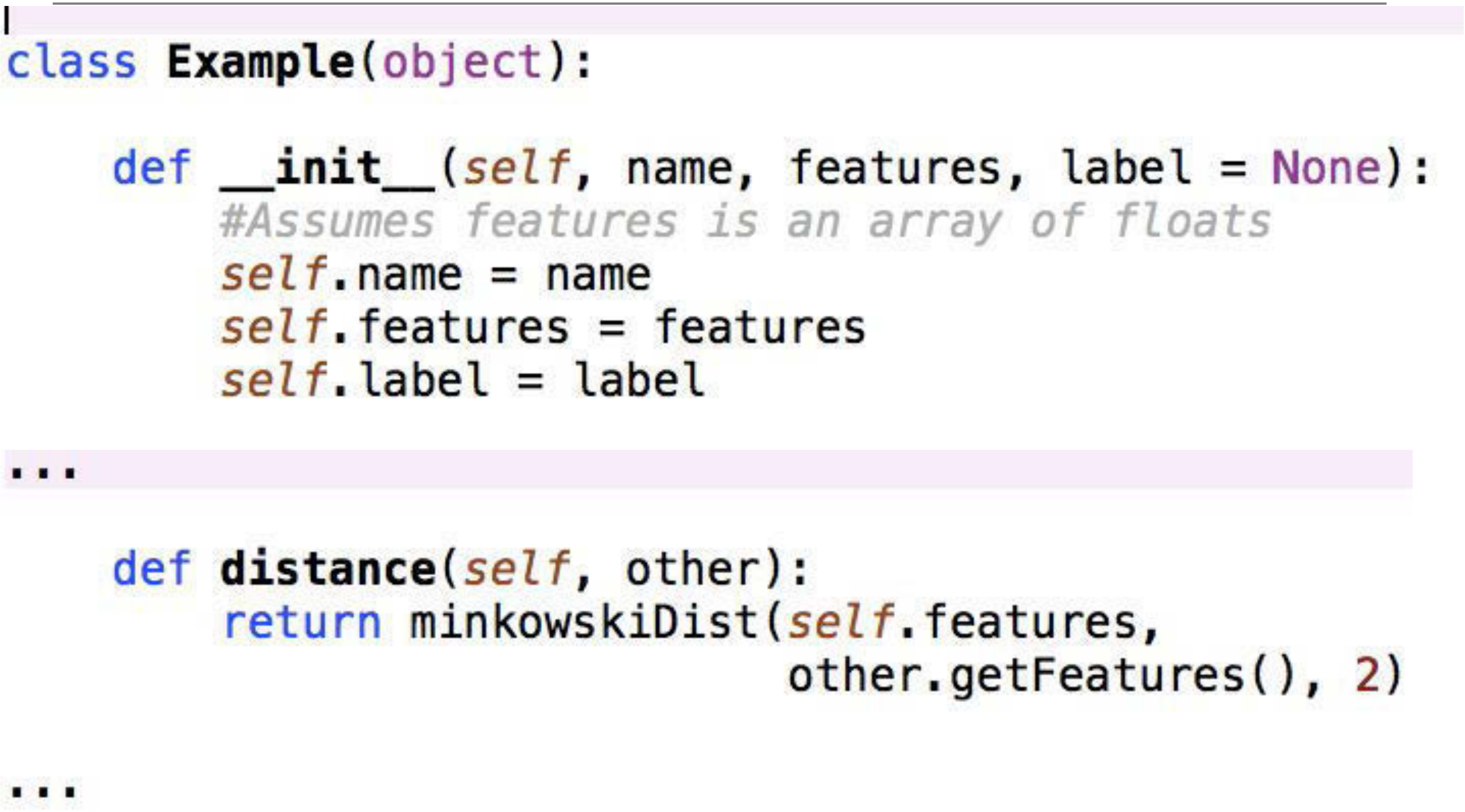


- Outcome (death) based on features
  - Probabilistic, not deterministic
  - E.g., older people with multiple heart attacks at higher risk
- Cluster, and examine purity of clusters relative to outcomes

Data Sample

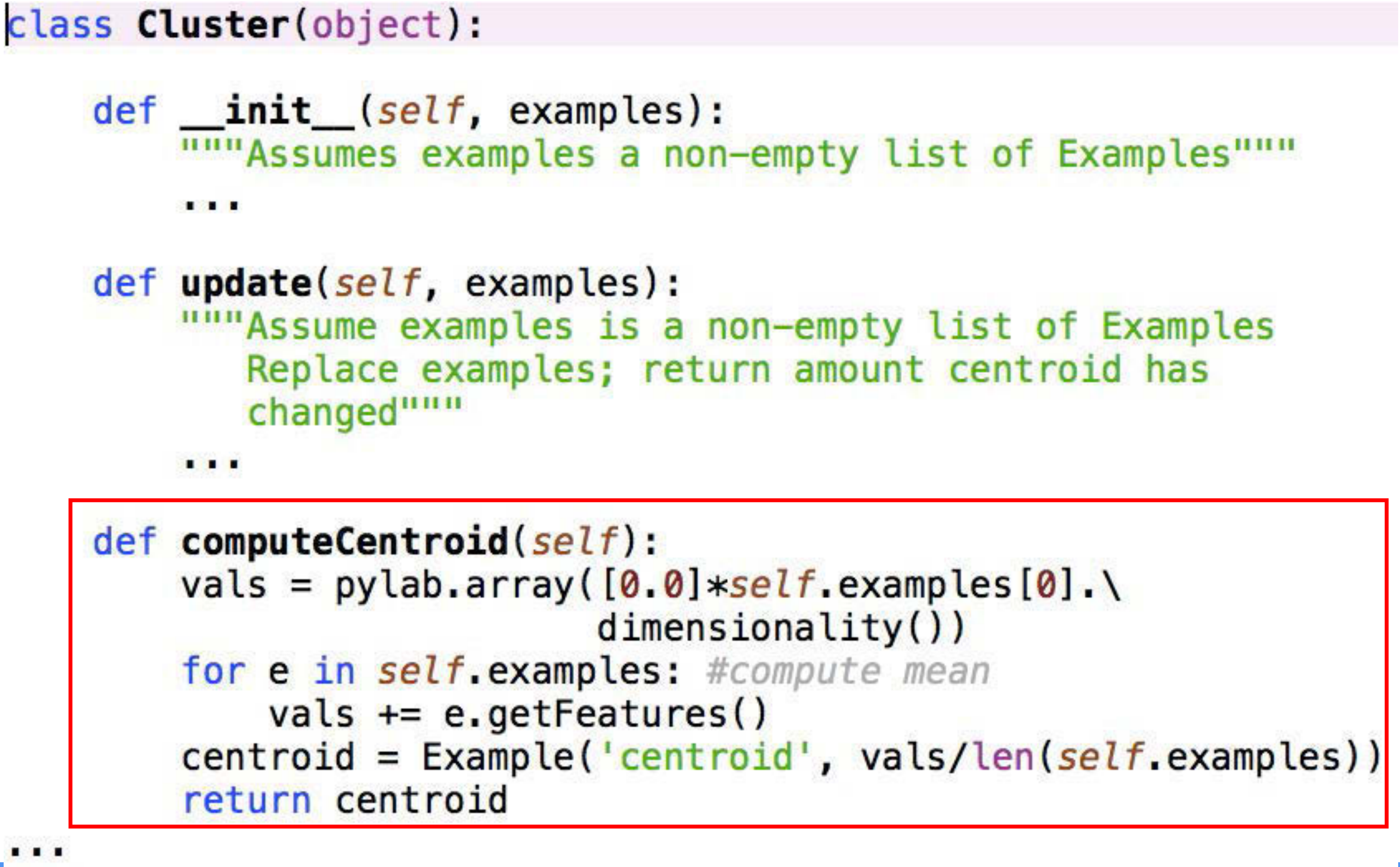
```
       HR    Att  STE  Age Outcome
P000:[ 89.   1.   0.  66.]:1
P001:[ 59.   0.   0.  72.]:0
P002:[ 73.   0.   0.  73.]:0
P003:[ 56.   1.   0.  65.]:0
P004:[ 75.   1.   1.  68.]:1
P005:[ 68.   1.   0.  56.]:0
P006:[ 73.   1.   0.  75.]:1
P007:[ 72.   0.   0.  65.]:0
P008:[ 73.   1.   0.  64.]:1
P009:[ 73.   0.   0.  58.]:0
P010:[ 100.  0.   0.  75.]:0
P011:[ 79.   0.   0.  31.]:0
P012:[ 81.   0.   0.  58.]:0
P013:[ 89.   1.   0.  50.]:1
P014:[ 81.   0.   0.  70.]:0
```

Class Example



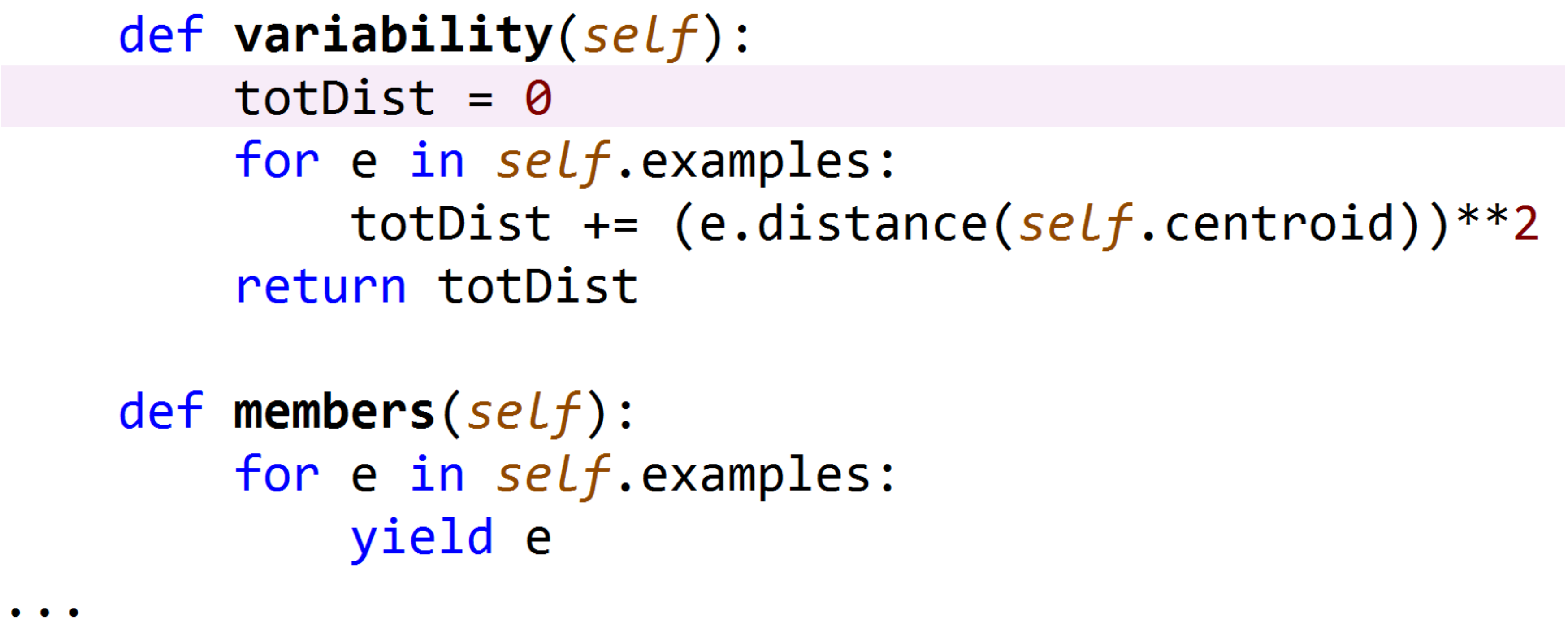
```
class Example(object):

    def __init__(self, name, features, label = None):
        #Assumes features is an array of floats
        self.name = name
        self.features = features
        self.label = label

...

    def distance(self, other):
        return minkowskiDist(self.features,
                             other.getFeatures(), 2)

...
```

Class Cluster



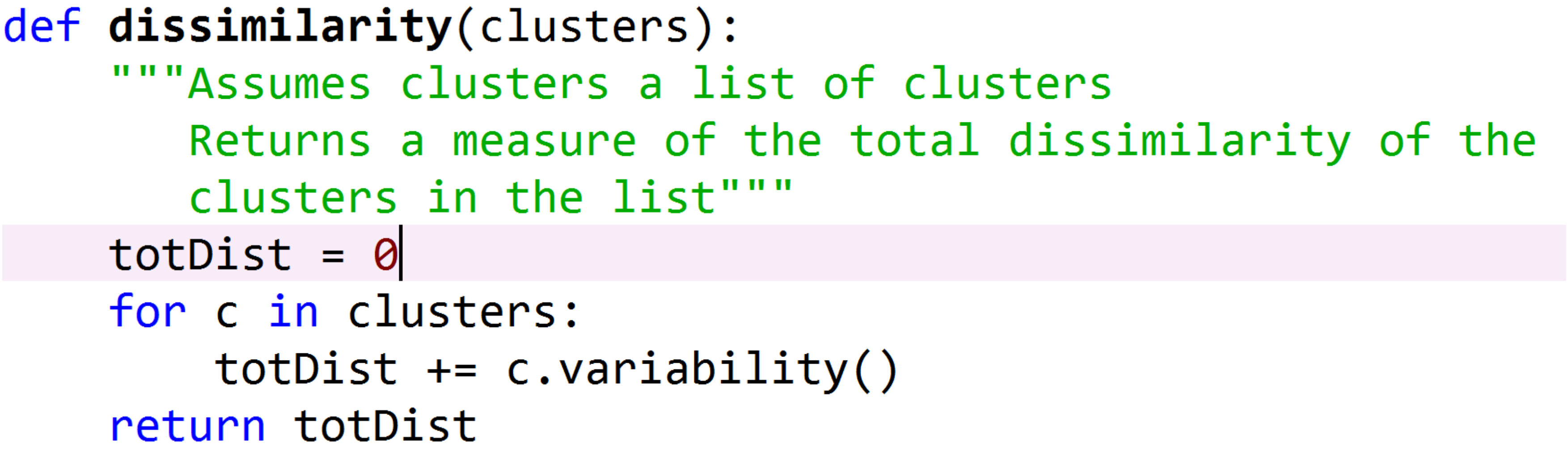
```
class Cluster(object):

    def __init__(self, examples):
        """Assumes examples a non-empty list of Examples"""
        ...

    def update(self, examples):
        """Assume examples is a non-empty list of Examples
           Replace examples; return amount centroid has
           changed"""
        ...

    def computeCentroid(self):
        vals = pylab.array([0.0]*self.examples[0].\
                                  dimensionality())
        for e in self.examples: #compute mean
            vals += e.getFeatures()
        centroid = Example('centroid', vals/len(self.examples))
        return centroid
...
```

Class Cluster, cont.



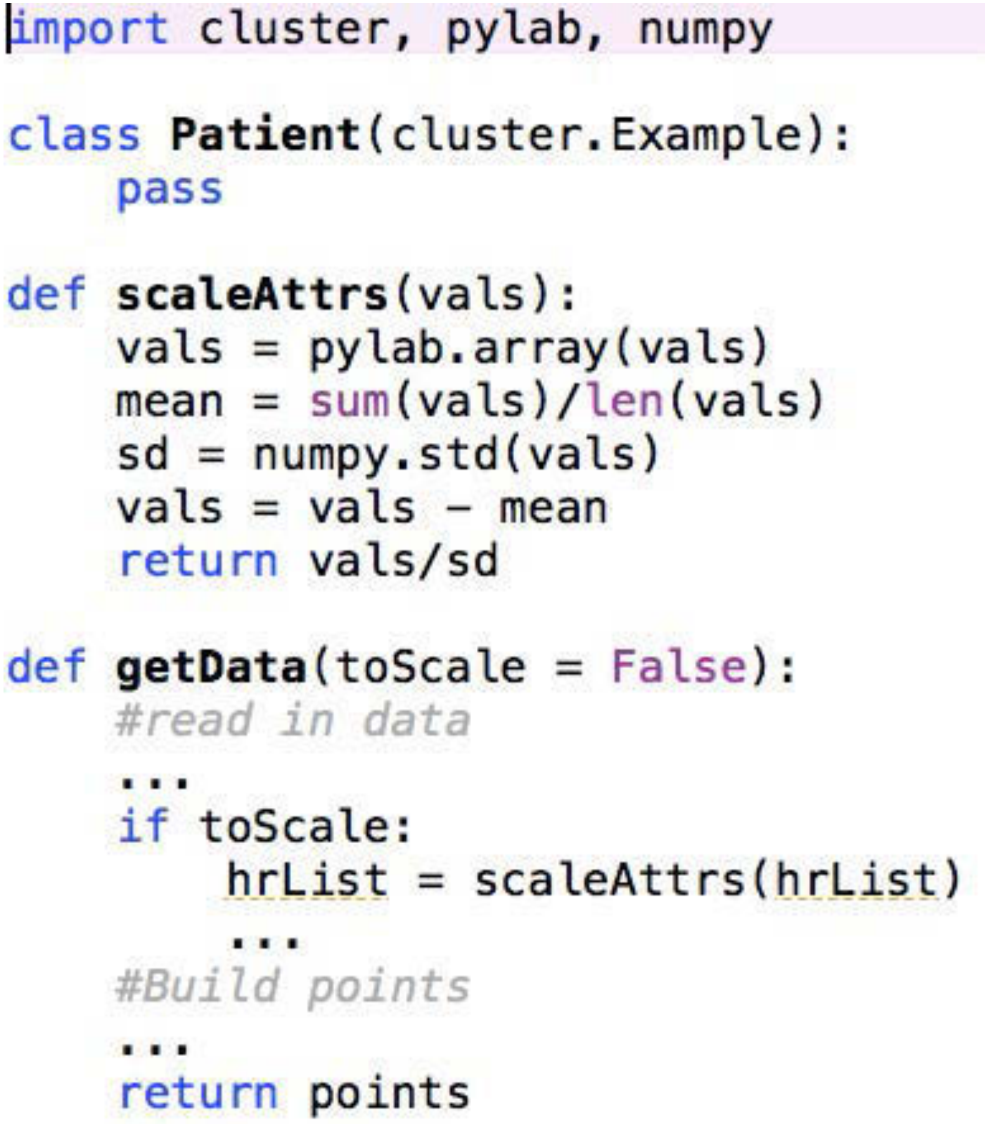
```
    def variability(self):
        totDist = 0
        for e in self.examples:
            totDist += (e.distance(self.centroid))**2
        return totDist

    def members(self):
        for e in self.examples:
            yield e
...
```

Evaluating a Clustering



```python
def dissimilarity(clusters):
    """Assumes clusters a list of clusters
       Returns a measure of the total dissimilarity of the
       clusters in the list"""
    totDist = 0
    for c in clusters:
        totDist += c.variability()
    return totDist
```

Patients



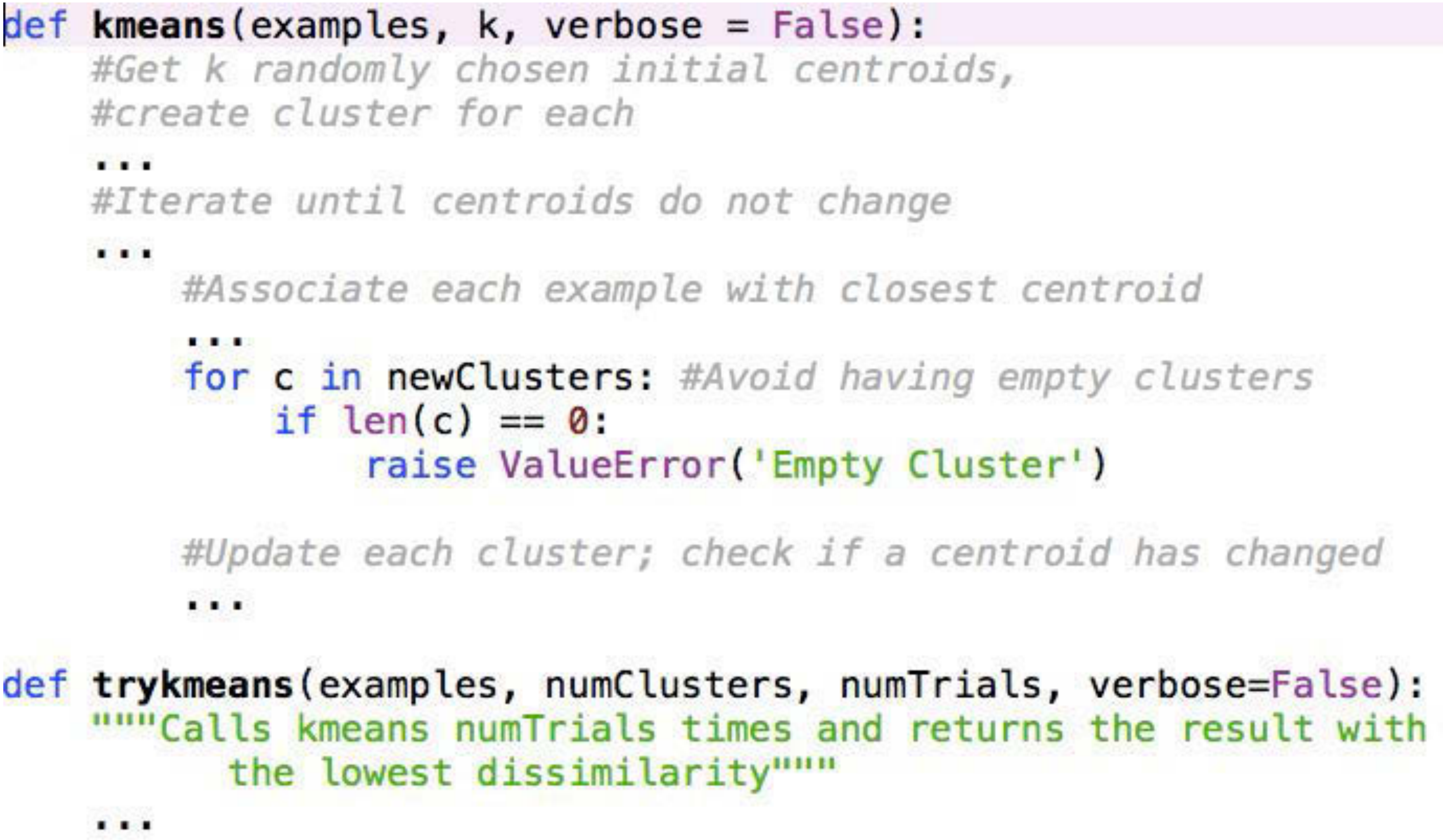
```
import cluster, pylab, numpy

class Patient(cluster.Example):
    pass

def scaleAttrs(vals):
    vals = pylab.array(vals)
    mean = sum(vals)/len(vals)
    sd = numpy.std(vals)
    vals = vals - mean
    return vals/sd

def getData(toScale = False):
    #read in data
    ...
    if toScale:
        hrList = scaleAttrs(hrList)
        ...
    #Build points
    ...
    return points
```

Z-Scaling

Mean = ?

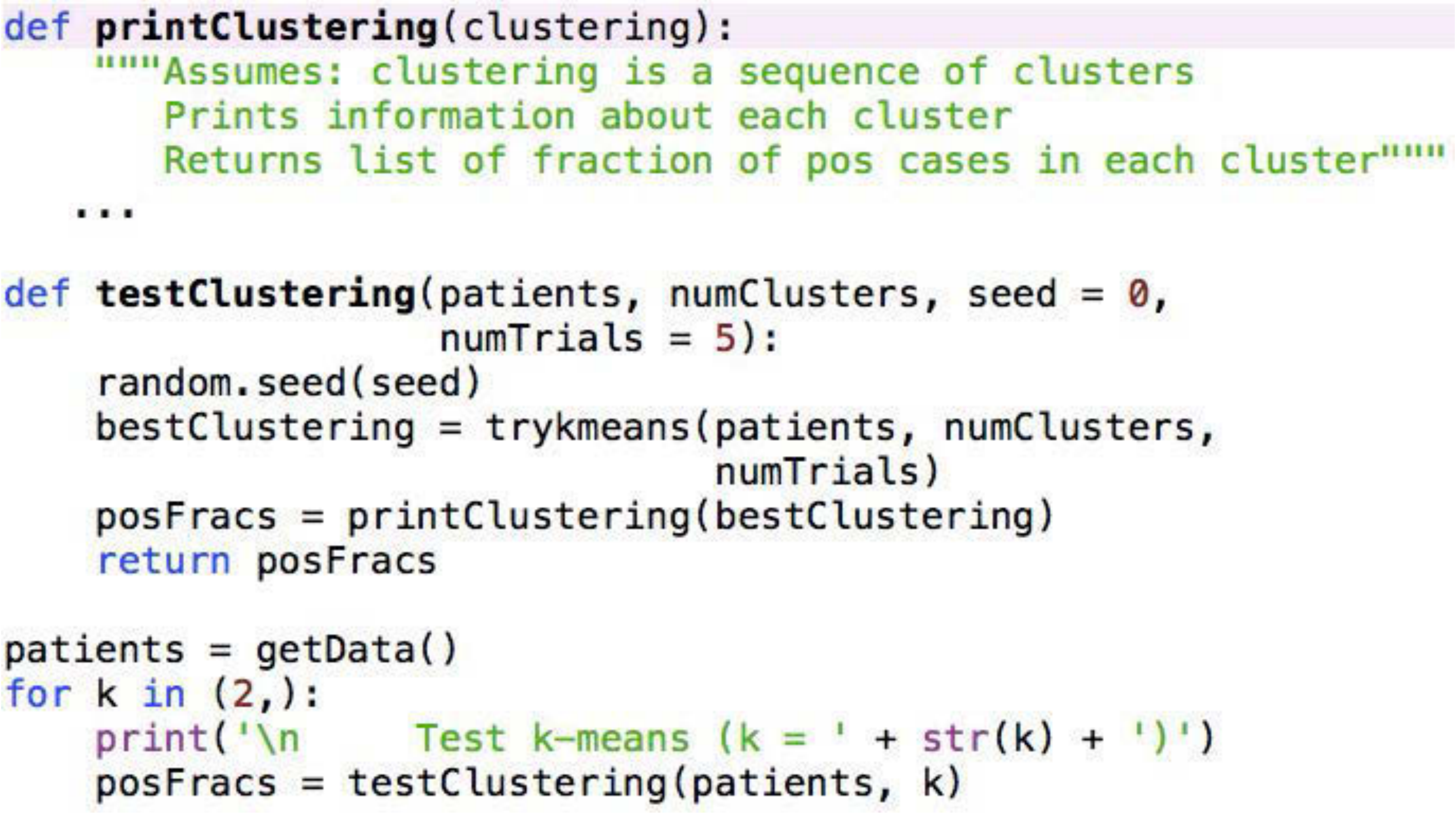
Std = ?

kmeans



```
def kmeans(examples, k, verbose = False):
    #Get k randomly chosen initial centroids,
    #create cluster for each
    ...
    #Iterate until centroids do not change
    ...
        #Associate each example with closest centroid
        ...
        for c in newClusters: #Avoid having empty clusters
            if len(c) == 0:
                raise ValueError('Empty Cluster')

        #Update each cluster; check if a centroid has changed
        ...

def trykmeans(examples, numClusters, numTrials, verbose=False):
    """Calls kmeans numTrials times and returns the result with
          the lowest dissimilarity"""
    ...
```

Examining Results



```
def printClustering(clustering):
    """Assumes: clustering is a sequence of clusters
       Prints information about each cluster
       Returns list of fraction of pos cases in each cluster"""
  ...

def testClustering(patients, numClusters, seed = 0,
                   numTrials = 5):
    random.seed(seed)
    bestClustering = trykmeans(patients, numClusters,
                               numTrials)
    posFracs = printClustering(bestClustering)
    return posFracs

patients = getData()
for k in (2,):
    print('\n     Test k-means (k = ' + str(k) + ')')
    posFracs = testClustering(patients, k)
```

Result of Running It

```
        Test k-means (k = 2)
Cluster of size 118 with fraction of positives = 0.3305
Cluster of size 132 with fraction of positives = 0.3333
```

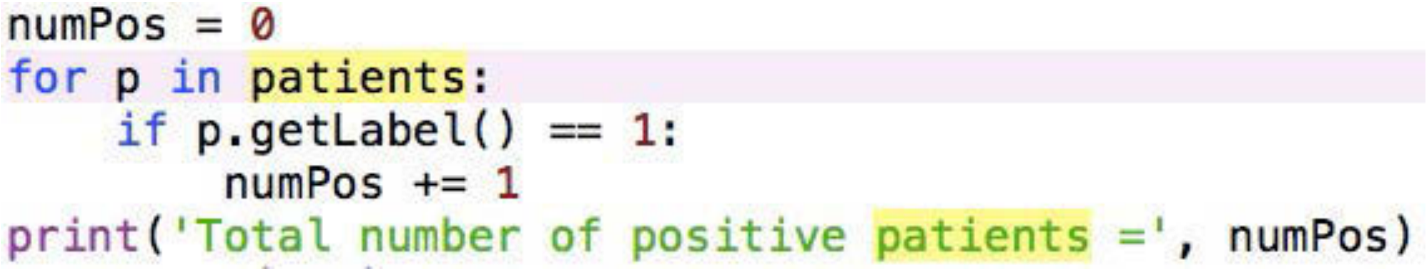
Like it?

Try patients = getData(True)

```
        Test k-means (k = 2)
Cluster of size 224 with fraction of positives = 0.2902
Cluster of size 26 with fraction of positives = 0.6923
```

Happy with sensitivity?

How Many Positives Are There?



```
numPos = 0
for p in patients:
    if p.getLabel() == 1:
        numPos += 1
print('Total number of positive patients =', numPos)
```

```
Total number of positive patients = 83
```

```
        Test k-means (k = 2)
Cluster of size 224 with fraction of positives = 0.2902
Cluster of size 26 with fraction of positives = 0.6923
```

A Hypothesis

- Different subgroups of positive patients have different characteristics
- How might we test this?
- Try some other values of k

```
patients = getData()
for k in (2,4,6):
    print('\n     Test k-means (k = ' + str(k) + ')')
    posFracs = testClustering(patients, k, 2)
```

Testing Multiple Values of k

```
 Test k-means (k = 2)
Cluster of size 224 with fraction of positives = 0.2902
Cluster of size 26 with fraction of positives = 0.6923
     Test k-means (k = 4)
Cluster of size 26 with fraction of positives = 0.6923
Cluster of size 86 with fraction of positives = 0.0814
Cluster of size 76 with fraction of positives = 0.7105
Cluster of size 62 with fraction of positives = 0.0645
     Test k-means (k = 6)
Cluster of size 49 with fraction of positives = 0.0204
Cluster of size 26 with fraction of positives = 0.6923
Cluster of size 45 with fraction of positives = 0.0889
Cluster of size 54 with fraction of positives = 0.0926
Cluster of size 36 with fraction of positives = 0.7778
Cluster of size 40 with fraction of positives = 0.675
```

maybe 4 is
the right
answer

Pick a k