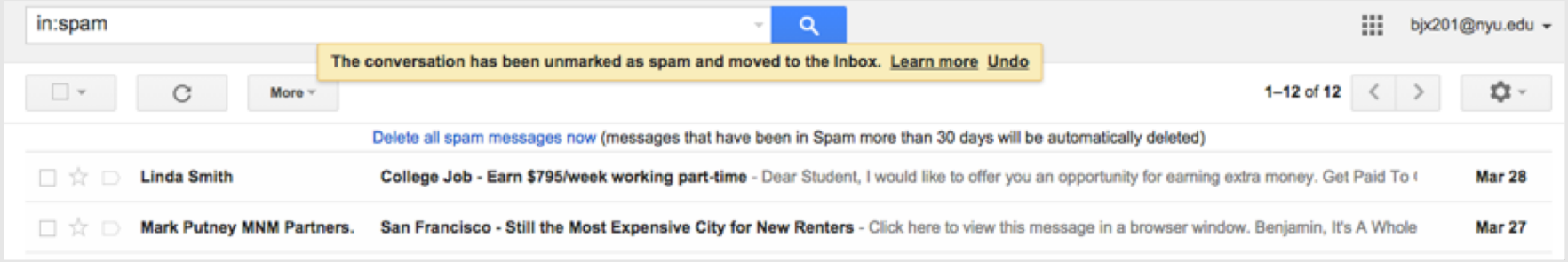
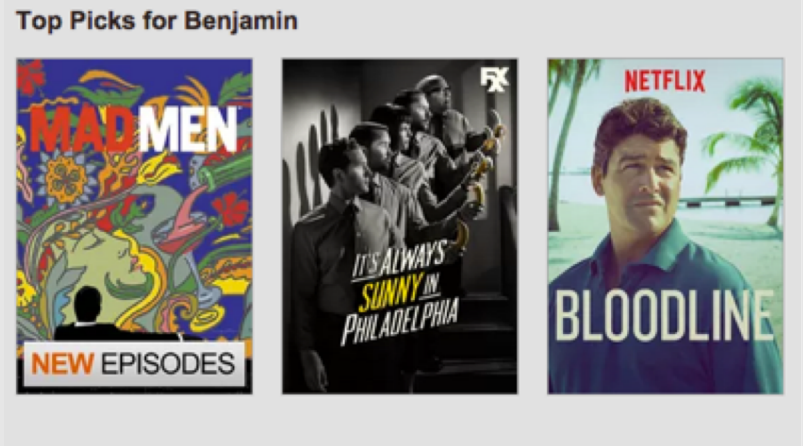
Data Science II

Introduction to machine learning

You are exposed to ML *CONSTANTLY*





Everything from E-mail spam filters to Netflix recommendation engines & Amazon suggested products all rely on machines crunching incomprehensible amounts of data in order to better "learn" your preferences!

Agenda

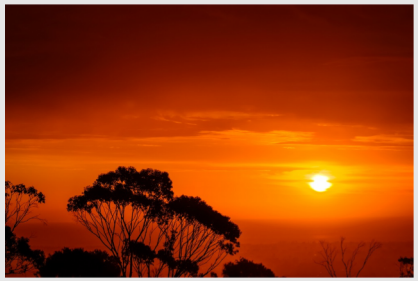
- Machine learning introduction
  - Defining ML
  - Supervised vs. unsupervised
- Clustering
  - k-means clustering algorithm

What is machine learning?

Human beings are good learners

We have proverbs, telling us what will happen in future.

- “Red sky at night, shepherd's delight. Red sky in the morning, shepherd's warning.”（朝霞不出门，晚霞行千里）



Tomorrow will be sunny.

Human beings are good learners

We also have life tricks.

- Methods for picking out a perfect watermelon in a grocery.

How do we know these?

- “Red sky at night” indicates “tomorrow is sunny”
- “A dark field spot” means “the watermelon is ripe”

We know them because we are experienced.

- Human beings can learn from experience.
- We find general rules by reasoning from the observed facts.

As an extension of our brains, can computers do the same thing?

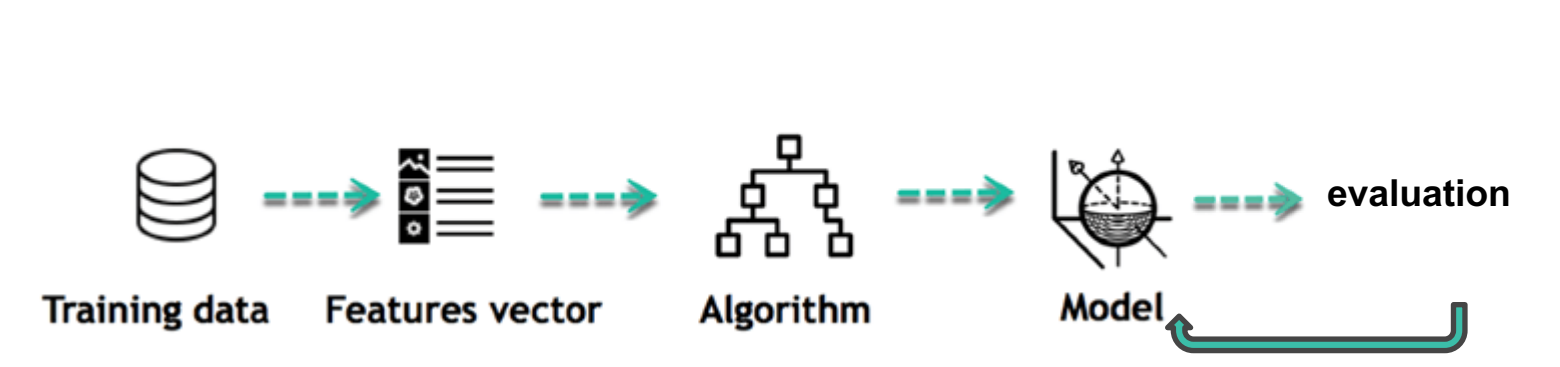
ML is PET

A computer program is said to learn from experience E with respect to some class of task T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

--- Tom Mitchell

Tom Mitchell (1951-),
E. Fredkin University Professor,
Machine learning department,
School of computer science,
CMU

Machine learning



Experience: training data
Task: to predict target/objective variable
Performance measure: evaluation metrics

Data Representation

customer ID	gender	occupation	education	age	salary	marital	default on payment
541	F	teacher	university	28	4500	S	0
542	M	developer	high school	27	4000	M	1
543	F	engineer	university	35	9000	M	0
544	F	developer	university	25	3800	S	1
545	M	engineer	university	46	5600	S	0
546	F	teacher	university	28	5200	M	0
547	F	developer	high school	32	4700	M	0

Feature vector: a useful mathematical description of our data.

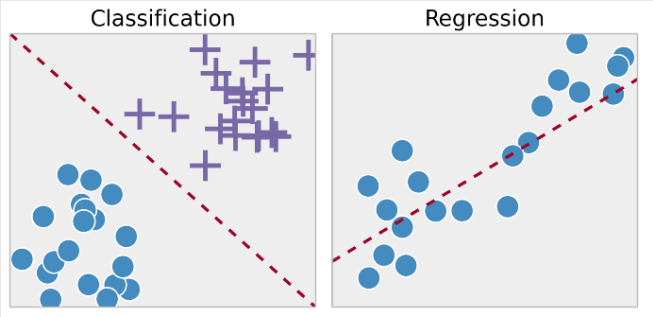
For some raw data format， we need to convert the data to feature vectors first.

What is machine learning?

Andrew Ng video

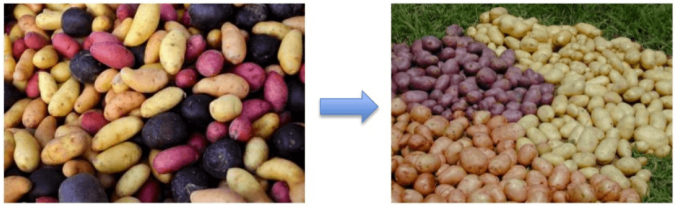
Full course @ youtube:
https://www.youtube.com/watch?v=PPLop4L2eGk&list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN&index=1

Supervised learning & Unsupervised learning



Supervised model

- Labeled data (X, y)
- We want to find mapping function y = f(X)
- Classification, Regression



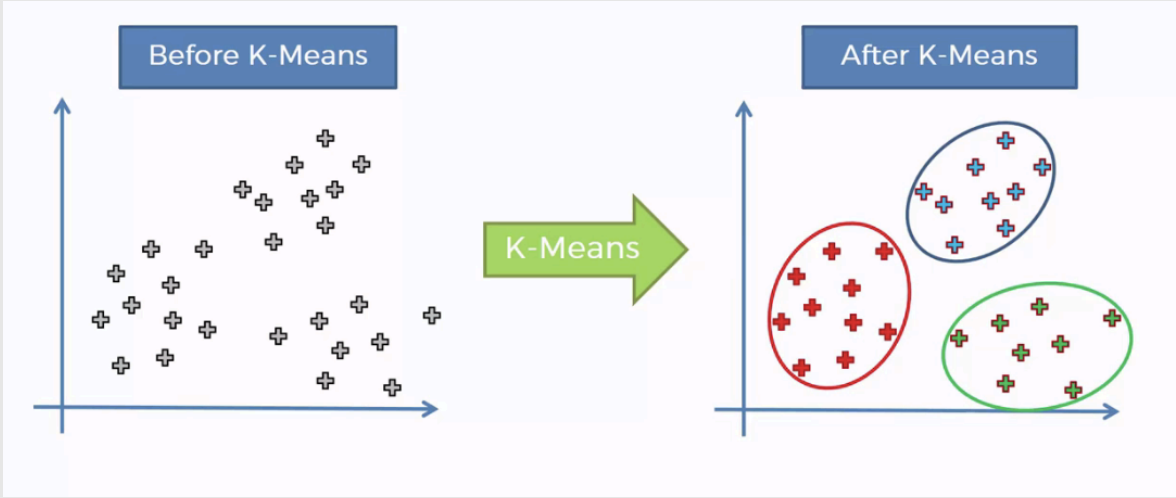
Unsupervised model

- Only have input data X
- Want to find underlying structure or distribution of the data
- Clustering, dimension reduction

Clustering

Clustering

- The goal:
  - “similar” members go into the same group



Distance Metrics

- We're in a world of "feature vectors" which can map to co-ordinates

The Minkowski distance of order *p* between two points

$$X = (x_1, x_2, \ldots, x_n) \text{ and } Y = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$$

$$D(X, Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

- The generalized **Minkowski Distance** can often be a useful measure
    - In 2-dimensions, it is the same as the Euclidean Distance when p is 2
    - When p is 1, it's Manhattan distance

Clustering

The goal:

- “Similar” members go into the same group

Key metrics:

- Intra-cluster variance (spread within a group)
- Inter-cluster variance (spread across groups)

Common objective:

- Maximize inter-cluster variance
- Minimize intra-cluster variance

Clustering

Given n data points $x_1, \dots, x_n$, where x_i is a d-dimensional vector. K-Means clustering aims to partition the set of n data points to k sets $\{S_1, \dots, S_k\}$

Common objective:

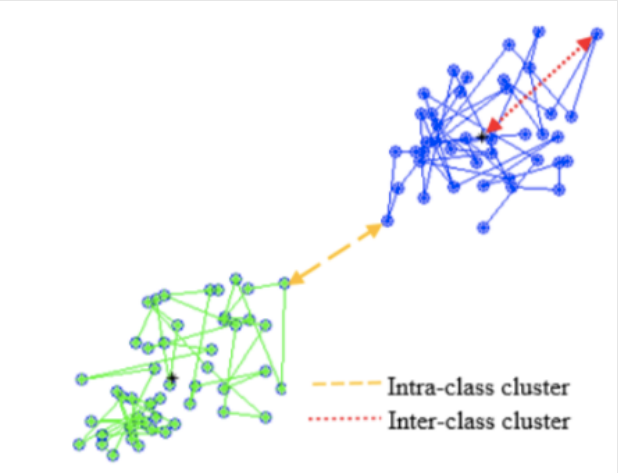
- Minimize intra-cluster variance

$$\sum_{i=1}^{k} \sum_{x \in S_i} \|x - c_i\|^2 = \sum_{i=1}^{k} |S_i| \, Var S_i$$

- Maximize inter-cluster variance

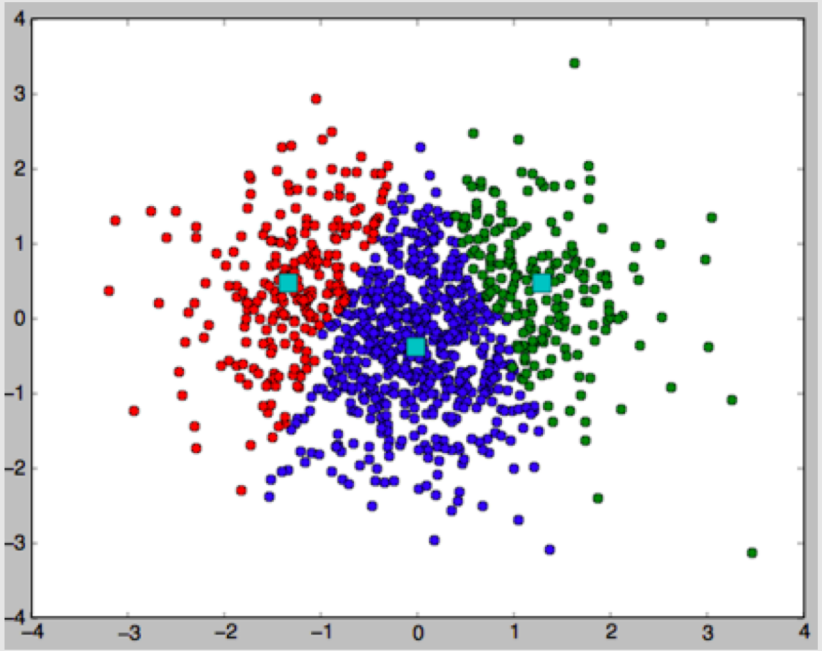
$$\sum_{i=1}^{k} |S_i| \cdot \|c_i - \bar{x}\|^2$$

c_i is the centroid for S_i, $\bar{x}$ is the mean of all n points



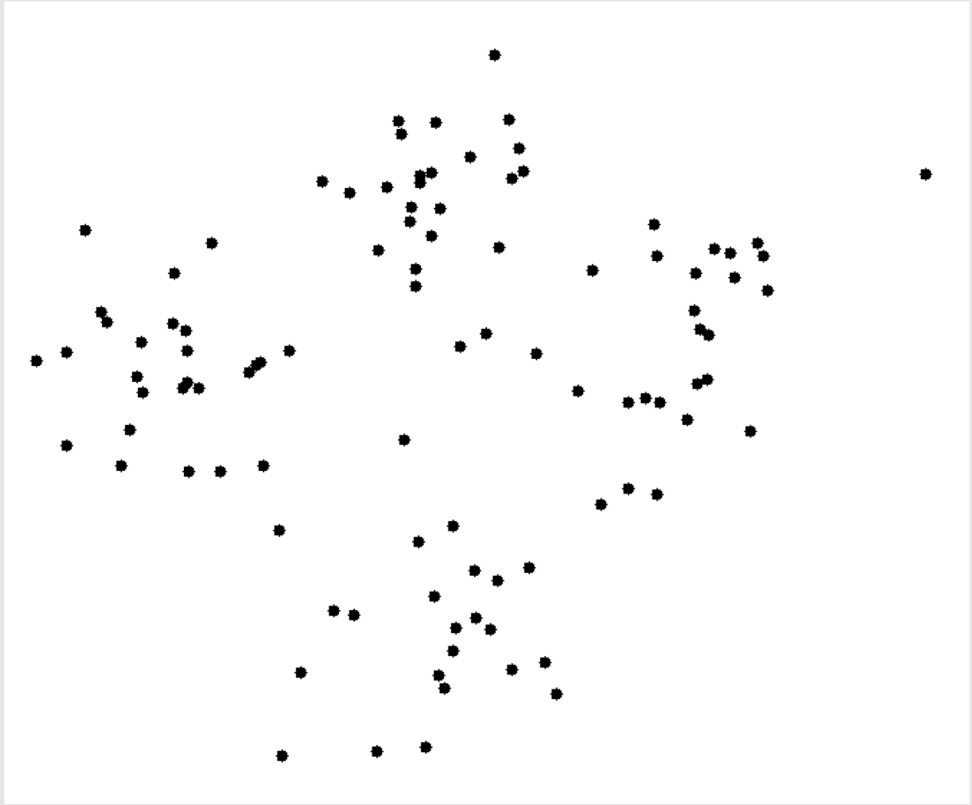
k-Means clustering: Algorithm

A very nice demo: http://shabal.in/visuals/kmeans/4.html

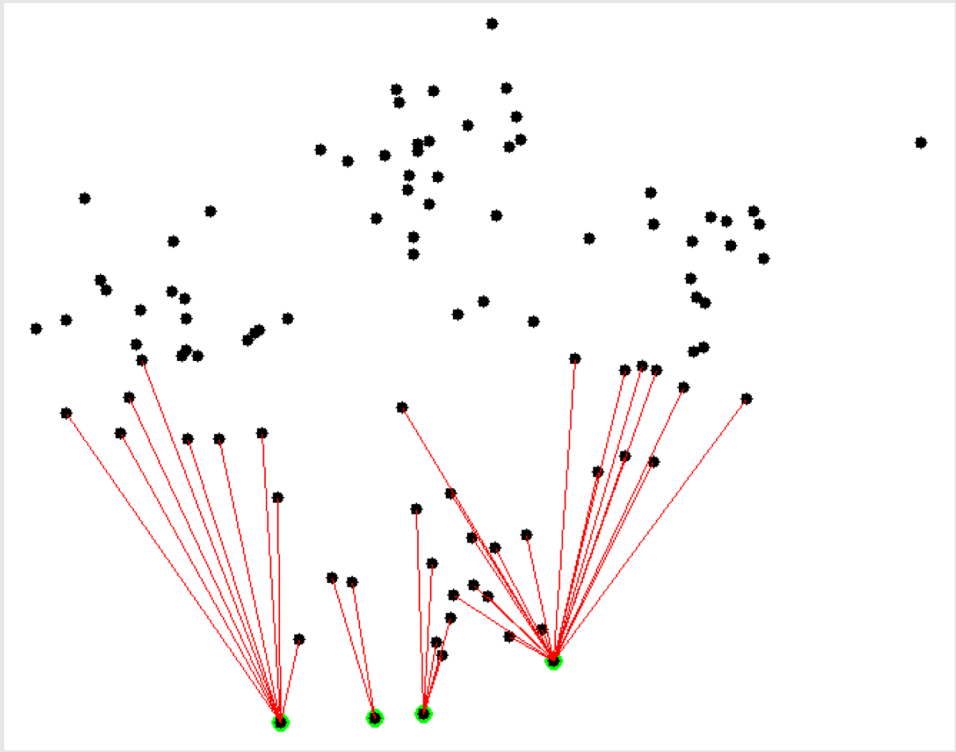


Try the link, can you figure it out?

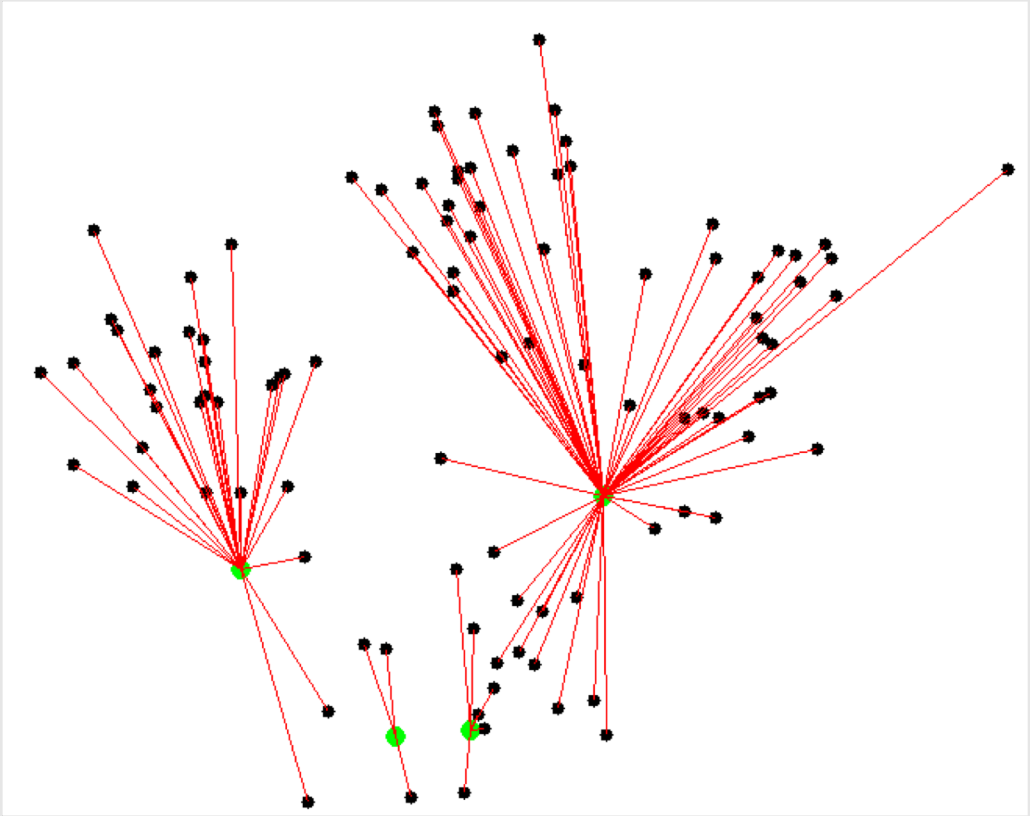
k-Means: steps



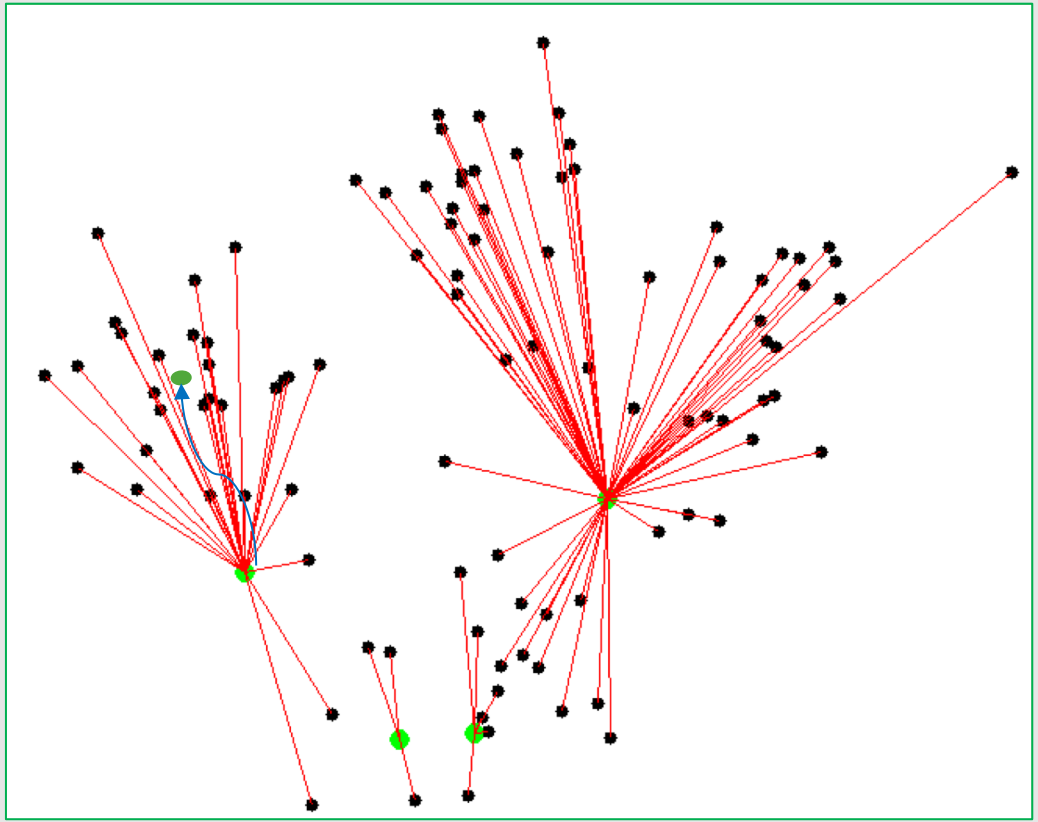
k-Means: steps



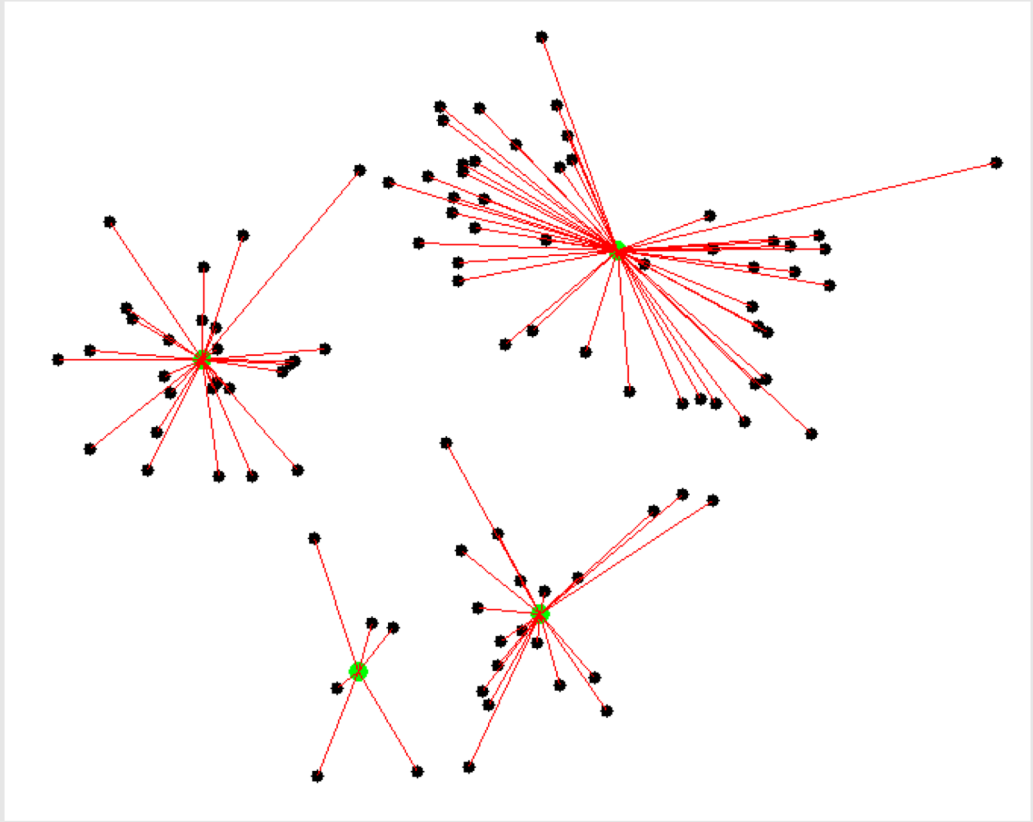
k-Means: steps



k-Means: steps



k-Means: steps



k-Means: the algorithm

Initialization:

- k empty clusters, each center is at one of k randomly picked members

Iterate until converge (i.e. no cluster moves its center).

1. Each member finds its closest cluster, add itself to the cluster
2. Each cluster computes its new centroid
3. If no cluster changes its centroid – done!
4. Otherwise, all clusters clear its members, go to 1.

Let's play! (k-means: https://www.youtube.com/watch?v=_aWzGGNrcic)

Before you start, think about how you will arrange your code

How will you write a k-mean algorithm?

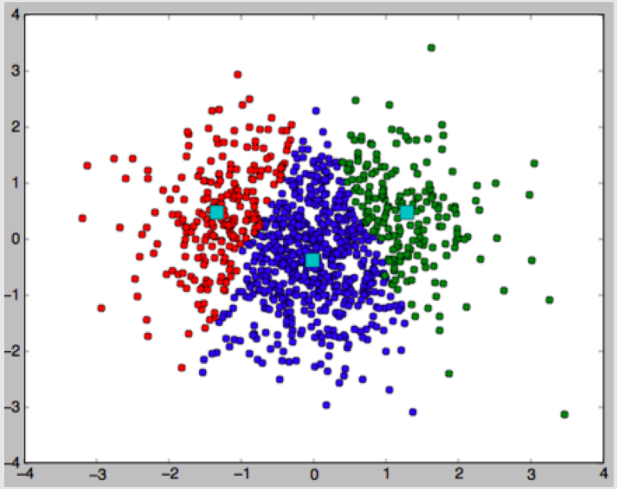
- What class/classes will you define?
- What are the attributes it has?
- What are the methods?

These are open questions, solutions are not unique.

One possible solution

- Set a cluster as a class

```
class Cluster:

    def __init__(self, samples):
        self.samples = samples
        self.centroids = self.computeCentroids()

    def computeCentroids(self):
        pass

    def update(self, samples):
        pass
## also needs getters and setters
    pass
```



- A data point is an instance of Sample class

```
class Sample:

    def __init__(self, name, features, label):
        pass

    def distance(self, other):
        pass
```

Implementation: the sample class

A sample has:

- A name, a constant number of features, a label

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

```
34 def __add__(self, other):
35     f = []
36     for i in range(self.dimensionality()):
37         f.append(self.getFeatures()[i] + other.getFeatures()[i])
38     return Sample(self.name + '+' + other.name, f)
```

```
60 if __name__ == "__main__":
61     a = Sample('a', [1, 1])
62     b = Sample('b', [-1, -1])
63     print(a)
64     print(b)
65     print(a + b)
66     print(a - b)
67     print(a/2)
```

```
46 #### Implement an overwrite of the '-' operator here!
47 def __sub__(self, other):
48     ''' replace the line below with you code
49         refer to the __add__ for ideas '''
50     return helper.__sub__(self, other)
```

Always remember to test your code !!!

Implementation: the Cluster class

This class hold data & methods for one cluster.

- Data attribute:
  - centroid
  - members (all data points in this cluster, each as a Sample instance)
- Methods:
  - computer centroid
  - update centroid with a new set of members

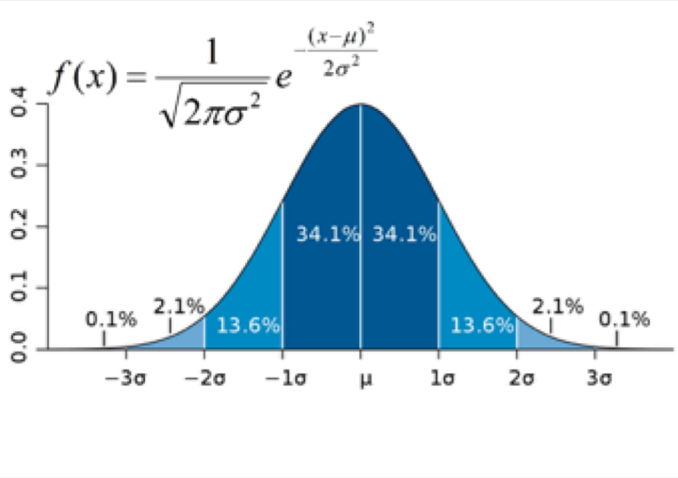
```
class Cluster(object):

    def __init__(self, samples):
        """Assumes samples is a list of Sample Class instances"""
        self.samples = samples # [a, b], a = sample.Sample('a', [1, 9]),
        """ centroid is also an instance of Sample class """
        self.centroid = self.computeCentroid()
```

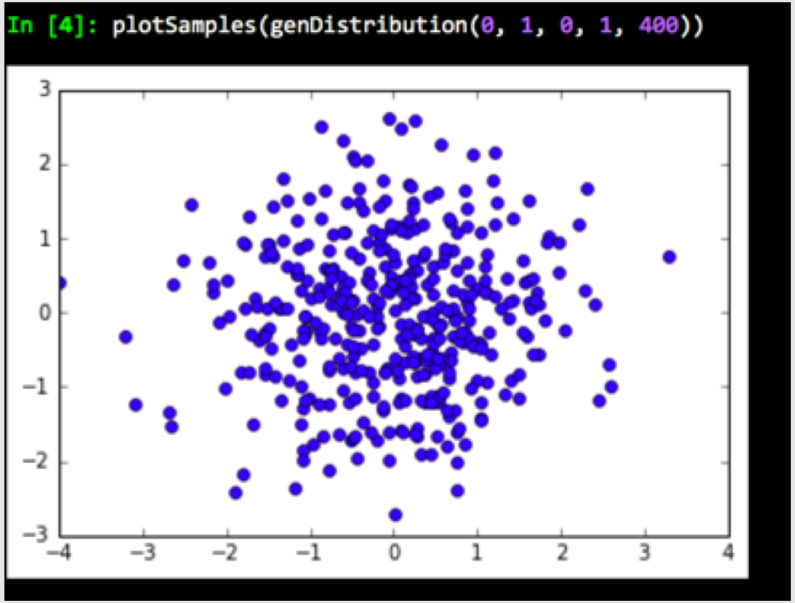
```
24    #### Implement the centroid computing function here!
25    def computeCentroid(self):
26        '''
27        return an instance of Sample, its features should be
28        the center of all the samples in the cluster
29        '''
30        return helper.computeCentroid(self)
```

```
32    #### Implement the centroid updating function here!
33    def update(self, samples):
34        """Replace the samples in the cluster by new samples
35           Return: how much the centroid has changed"""
36        return helper.update(self, samples)
```

Implementation: plotting and data gen in util module

```
19 # generating samples from Gaussian distribution
20 def genDistribution(xMean=0, xSD=1, yMean=0, ySD=1, n=50, namePrefi
21     samples = []
22     for s in range(n):
23         x = random.gauss(xMean, xSD)
24         y = random.gauss(yMean, ySD)
25         samples.append(sample.Sample(namePrefix+str(s), [x, y]))
26     return samples
```





Implementation: the util module

Distance to another sample: $L_p(\boldsymbol{x}, \boldsymbol{y}) = (\Sigma(x_i - y_i)^p)^{\left(\frac{1}{p}\right)}$

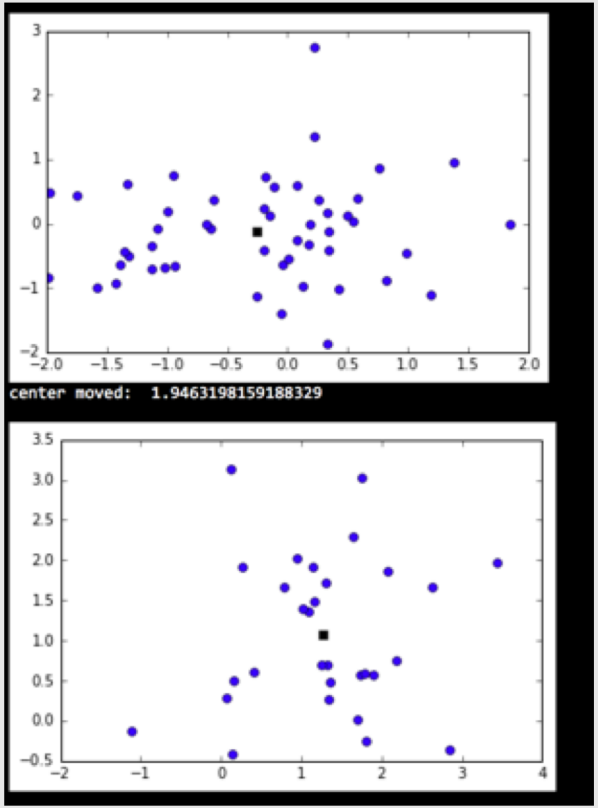
```
31 def distance(self, other):
32     return util.minkowskiDist(self.features, other.getFeatures(), 2)
```

In util.py

```
11 def minkowskiDist(v1, v2, p):
12     """Assumes v1 and v2 are equal-length arrays of numbers
13        Returns Minkowski distance of order p between v1 and v2"""
14     dist = 0.0
15     for i in range(len(v1)):
16         dist += abs(v1[i] - v2[i])**p
17     return dist**(1.0/p)
```

Implementation: the cluster class

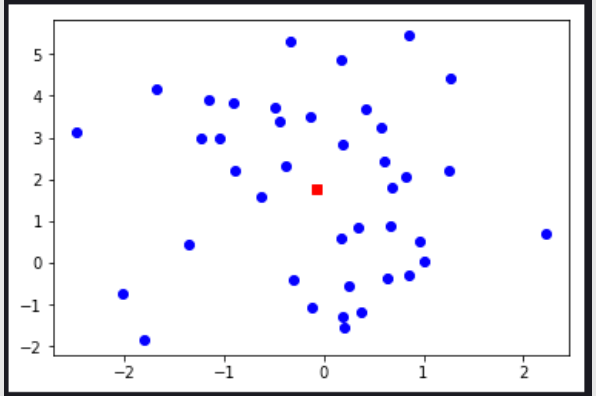
```
49 if __name__ == "__main__":
50     test_samples = util.genDistribution()
51     c = Cluster(test_samples)
52     print(c.centroid)
53     print("cluster center: ", c.centroid.features)
54     util.plot_cluster([c])
55
56     # now assign the cluster new samples, and move it
57     test_samples2 = util.genDistribution(1, 1, 1, 1, 30)
58     diff = c.update(test_samples2)
59     print("center moved: ", diff)
60     # plot_cluster expects an array of cluster...
61     util.plot_cluster([c])
```



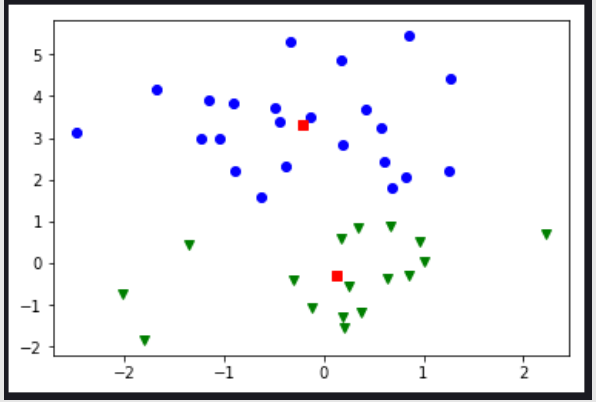
Implementation: k-means.py

```python
# one run of kmeans
def main(k=2, n=20, verbose=False):
    random.seed(100)
    d1samples = util.genDistribution(n=20)
    d2samples = util.genDistribution(xMean=0, xSD=1, yMean=4, ySD=1, n=20)
    allSamples = d1samples + d2samples

    print("before clustering")
    util.plot_cluster([cluster.Cluster(allSamples)])

    print("after clustering")
    clusters = kmeans(allSamples, 2, verbose)
    util.plot_cluster(clusters, verbose)

    for c in clusters:
        print(c)


if __name__ == "__main__":
    random.seed(0)
    main(k = 2)
```

before clustering



after clustering



Implementation: k-means main loop

```
25 # Kmeans: take a list of samples and make k clusters
26 def kmeans(samples, k, verbose):
27     """Assumes samples is a list of samples of class Sample,
28          k is a positive int, verbose is a Boolean
29        Returns a list containing k clusters. """
30
31     #Get k randomly chosen initial centroids
32     initialCentroids = random.sample(samples, k)
33
34     #Create a singleton cluster for each centroid
35     clusters = []
36     for e in initialCentroids:
37         clusters.append(cluster.Cluster([e]))
38
39     #Iterate until centroids do not change
40     converged = False
41     numIterations = 0
42     while not converged:
43
44         numIterations += 1
45
46         # replace the following line by implementing
47         # kmeans_iter(samples, clusters, k) in this file
48         converged = helper.kmeans_iter(samples, clusters, k)
49         #converged = kmeans_iter(samples, clusters, k)
50
51         if verbose:
52             print('Iteration #' + str(numIterations))
53             for c in clusters:
54                 print(c)
55             print('\n') #add blank line
56     return clusters
```

```
13 def kmeans_iter(samples, clusters, k):
14     '''
15         implement one iteration of kmeans:
16         - each sample finds the closest cluster by computing
17           its distance to the centroids of all the clusters
18         - then compute every cluster's new centroid
19         - the algorithm converges if cluster's centroid does not
20           move any more
21     '''
22     converged = True
23     return converged
```

Complexity of k-Means

n: number of points

k: number of clusters

d: number of dimension of each point

i: number of total iterations

Complexity is O(inkd)

Breast Cancer Data

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
842302	M	17.99	10.38	122.8	1001	0.1184
842517	M	20.57	17.77	132.9	1326	0.08474
84300903	M	19.69	21.25	130	1203	0.1096
84348301	M	11.42	20.38	77.58	386.1	0.1425
84358402	M	20.29	14.34	135.1	1297	0.1003
843786	M	12.45	15.7	82.57	477.1	0.1278
844359	M	18.25	19.98	119.6	1040	0.09463
84458202	M	13.71	20.83	90.2	577.9	0.1189
844981	M	13	21.82	87.5	519.8	0.1273
84501001	M	12.46	24.04	83.97	475.9	0.1186
845636	M	16.02	23.24	102.7	797.8	0.08206

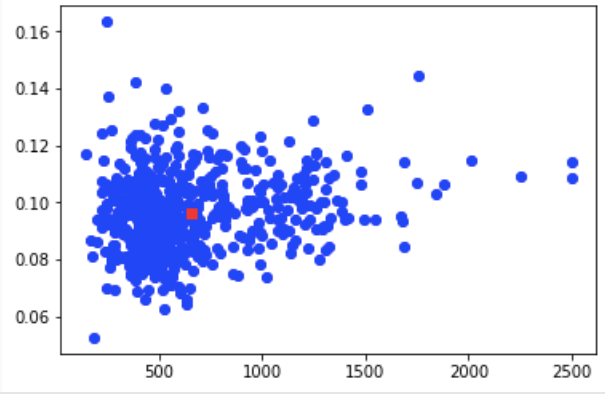
Feature vectors

Clustering based on area_mean, smoothness_mean

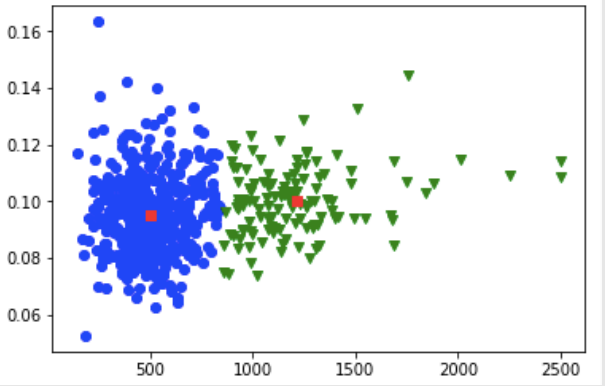
Label Breast-Cancer: kmeans_BC.py

```
def kmeansTest(k=2, n=20, verbose=False):
    random.seed(0)

    l = open("BC_data.csv").readlines()[1:]
    data = [[i for i in line.strip().split(',')] for line in l]

    # you can adjust the input features below
    allSamples = [sample.Sample(i[0], [float(j)
                                       for j in i[5:7]], i[1]) for i in data]
    print("before clustering")
    util.plot_cluster([cluster.Cluster(allSamples)])

    cluster_b = [s for s in allSamples if s.getLabel() == 'B']
    cluster_m = [s for s in allSamples if s.getLabel() == 'M']
    clusters = [cluster.Cluster(cluster_b), cluster.Cluster(cluster_m)]
    print("label info")
    util.plot_cluster(clusters)

    print("after k-means clustering")
    clusters = kmeans(allSamples, k, verbose)
```

before clustering



after k-means clustering



How accurate is k-means clustering?

- Note that we only used 2 features for k-means cluster and here is the cluster result compare to original labels.