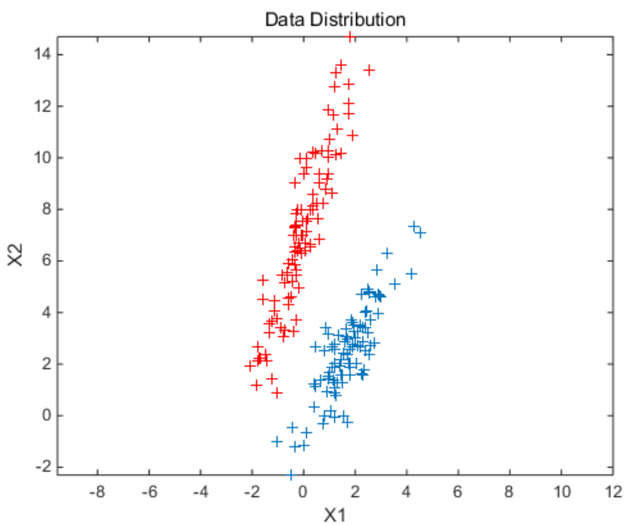
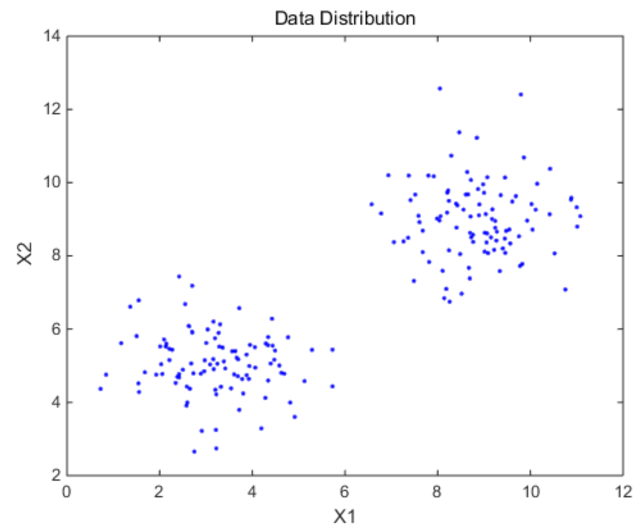
Clustering: K-means

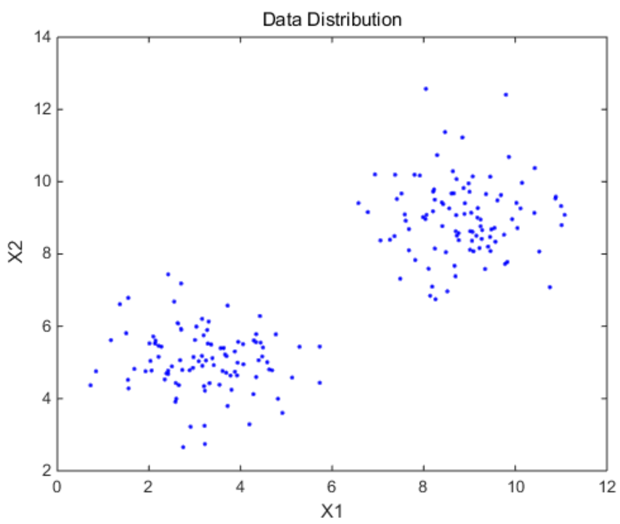
Industrial AI Lab.

Supervised vs. Unsupervised Learning

<table>
<tr><th>Supervised Learning</th><th>Unsupervised Learning</th></tr>
<tr><td>Building a model from labeled data</td><td>Clustering from unlabeled data</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>$$\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$$ $$\{y^{(1)}, y^{(2)}, \cdots, y^{(m)}\}$$ $\Rightarrow$ Classification</td><td>$$\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$$ $\Rightarrow$ Clustering</td></tr>
</table>

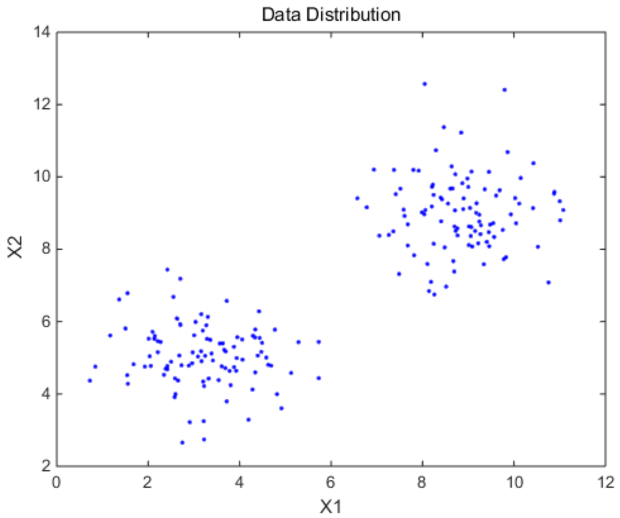
Data Clustering

- Data clustering is an unsupervised learning problem
- Given:
  - m unlabeled examples $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$
  - the number of partitions k
- Goal: group the examples into k partitions



$$\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\} \quad \Rightarrow \quad \text{Clustering}$$

Data Clustering: Similarity



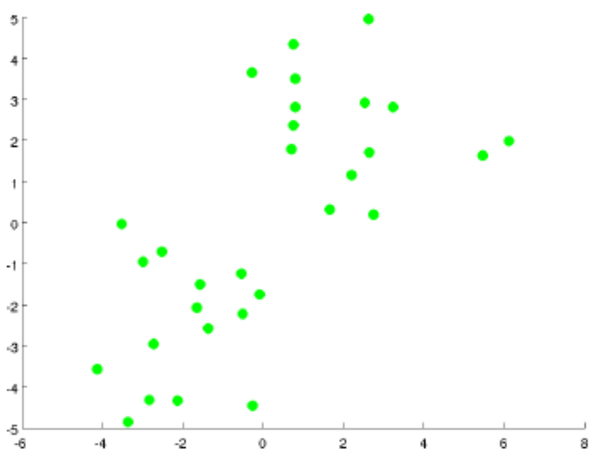
$$\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\} \quad \Rightarrow \quad \text{Clustering}$$

- The only information clustering uses is the mutual similarity between samples
- A good clustering is one that achieves:
  - high within-cluster similarity
  - low inter-cluster similarity

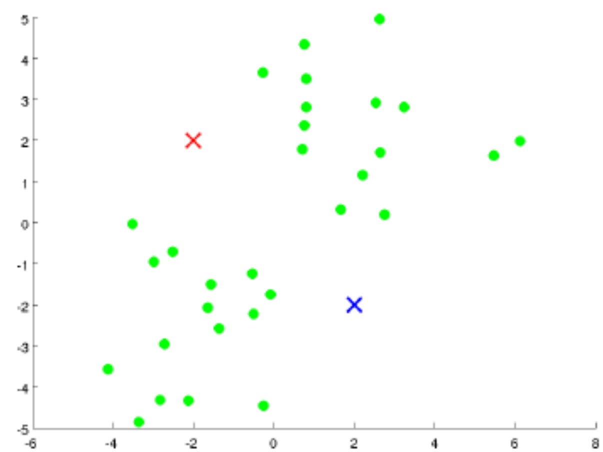
K-means: (Iterative) Algorithm

1) Initialization

- Input
  - k: the number of clusters
  - Training set $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$
- Randomly initialize cluster centers anywhere in $\mathbb{R}^n$



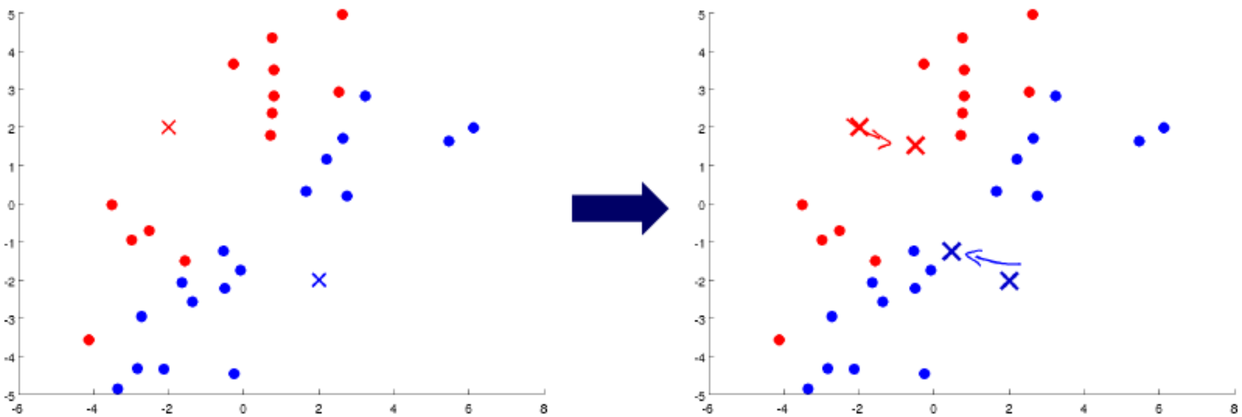
Training set data



Place centroids at random locations (K=2)

K-means: (Iterative) Algorithm

2) Iteration



Cluster Assignment

- For each point $x^{(i)}$
- Find nearest centroid μ_k
- Assign the point $x^{(i)}$ to cluster k

Move centroids

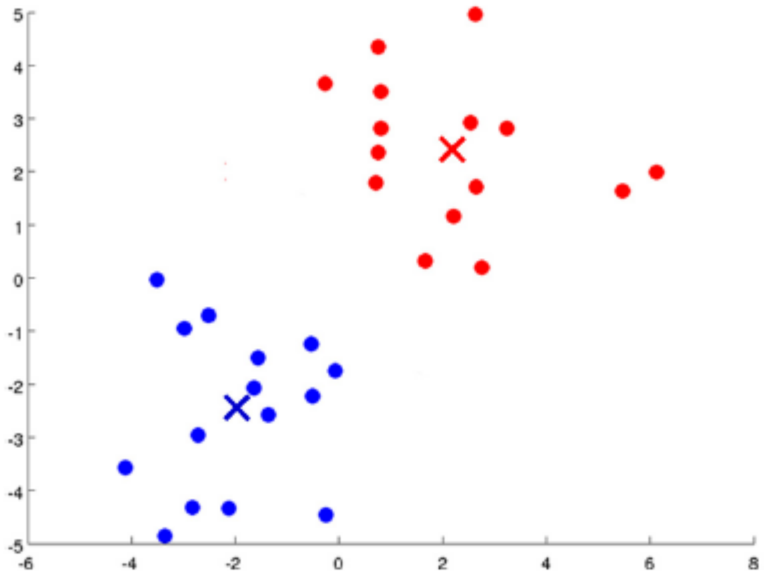
- Move centroids to mean of all points assigned to cluster k

- Repeat until convergence (a possible convergence criteria: cluster centers do not change anymore)

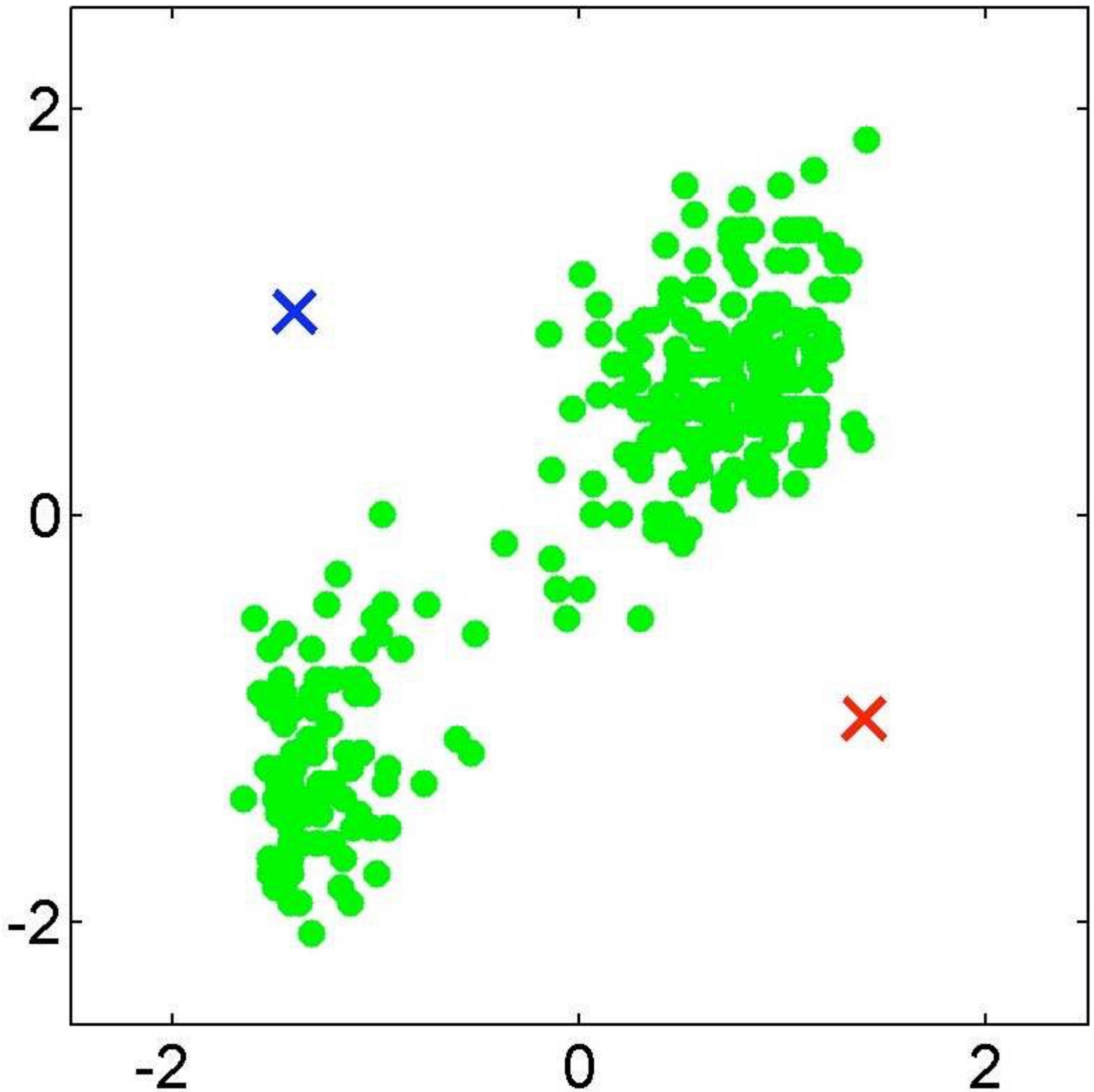
K-means: (Iterative) Algorithm

3) Output

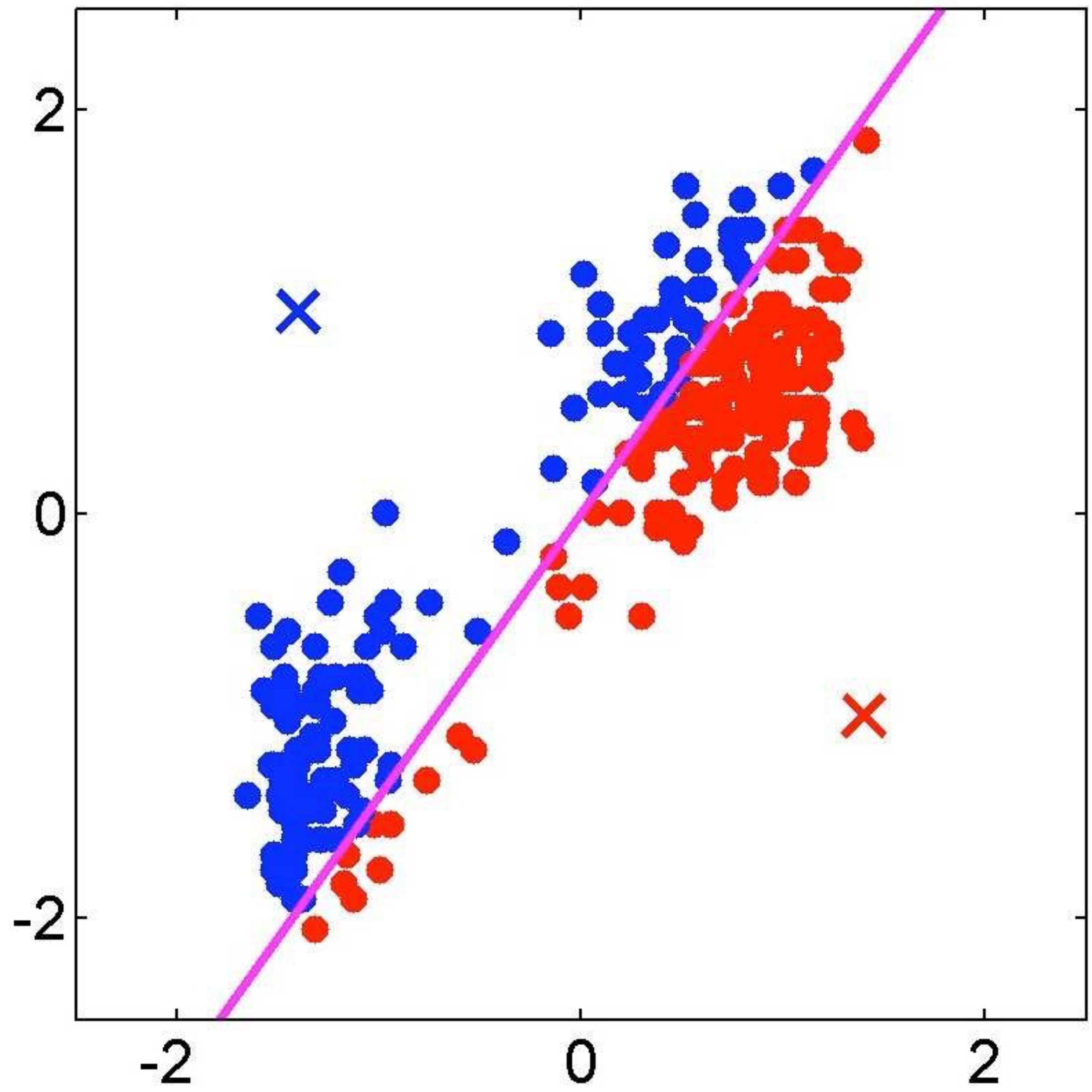
- c (label) : index (1 to k) of cluster centroid (centers)
- μ: averages (mean) of points assigned to cluster $\{\mu_1, \mu_2, \cdots, \mu_k\}$



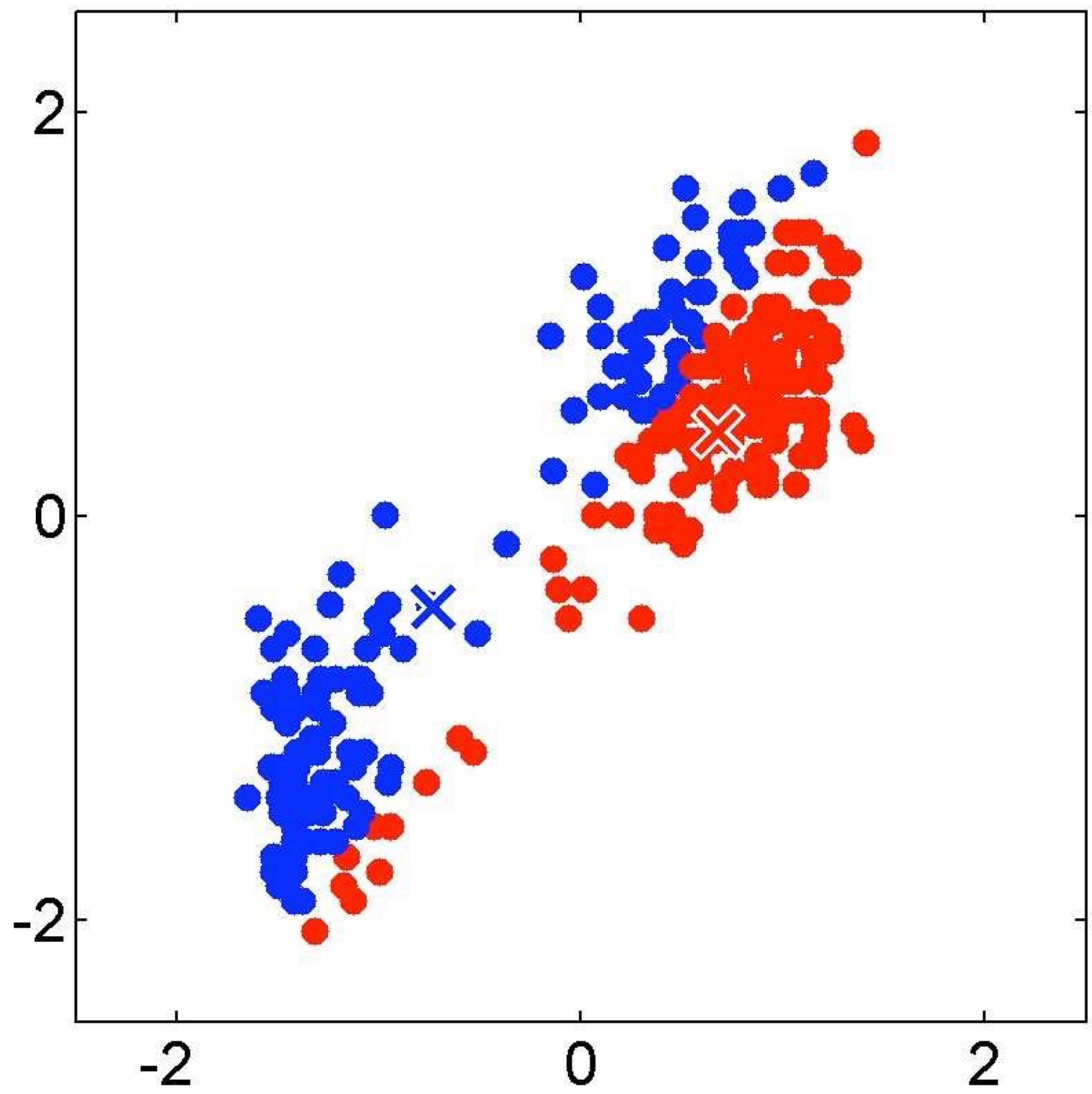
Initialization ($k = 2$)



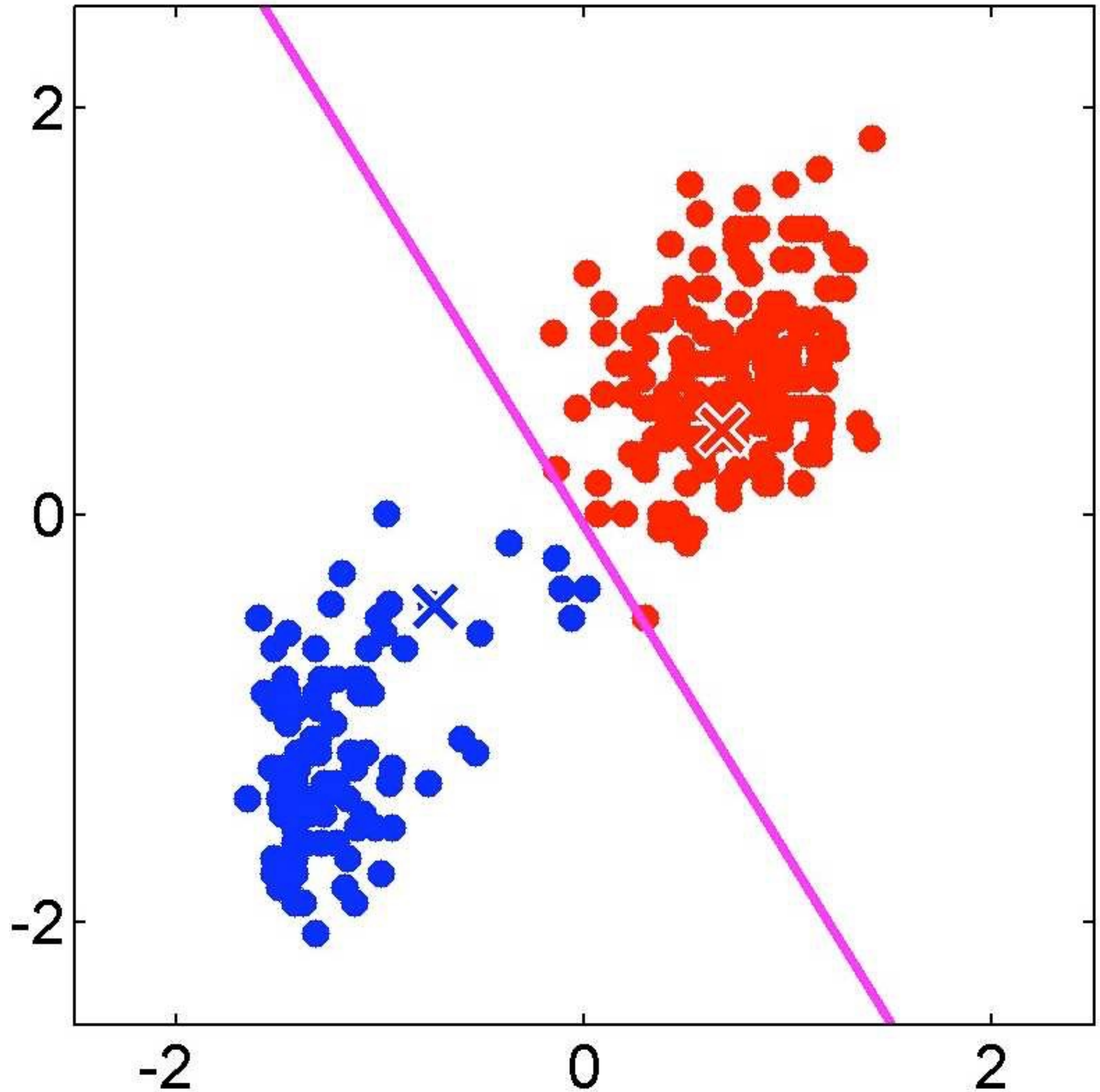
Assigning Points



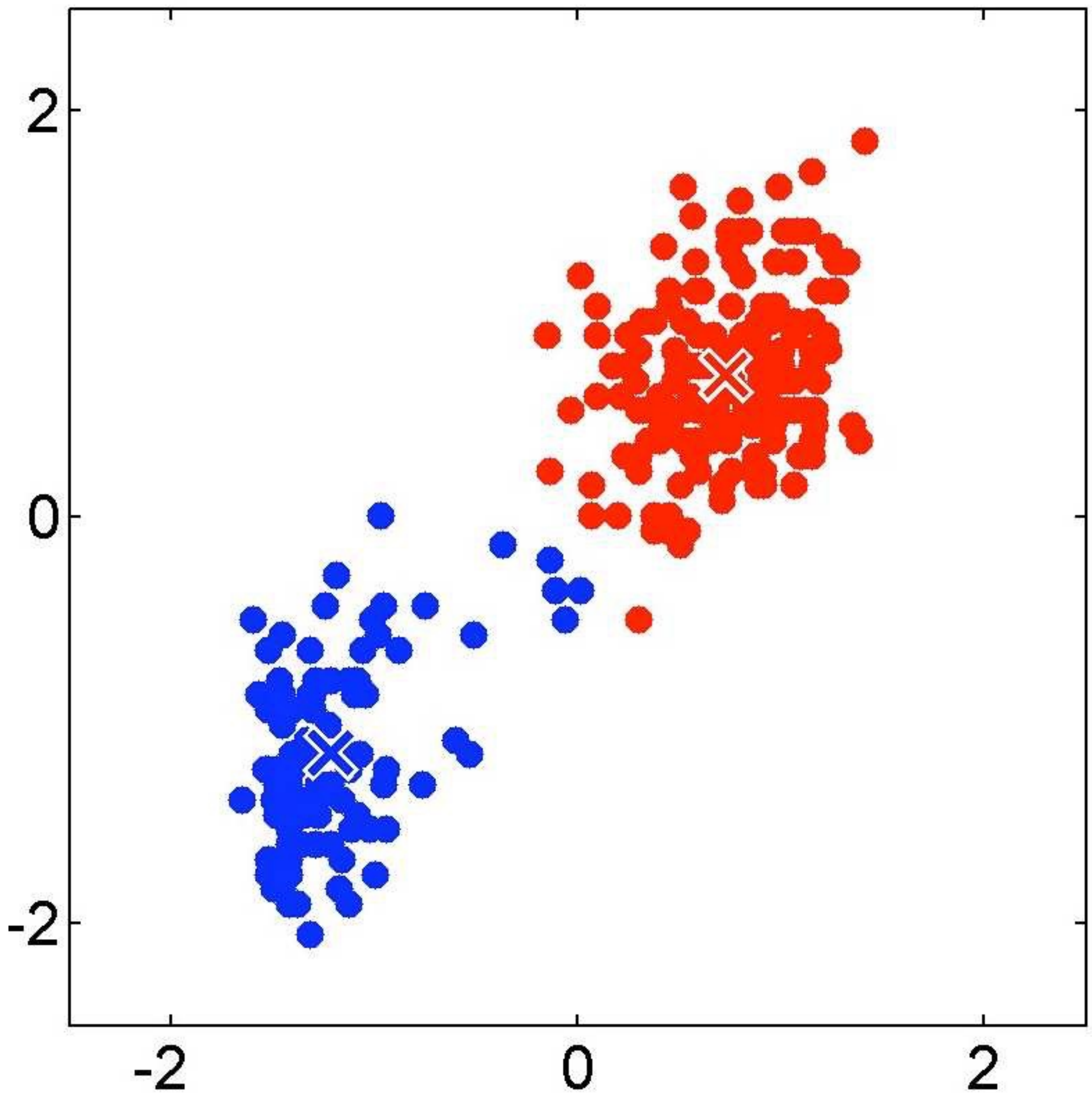
Recomputing the Cluster Centers



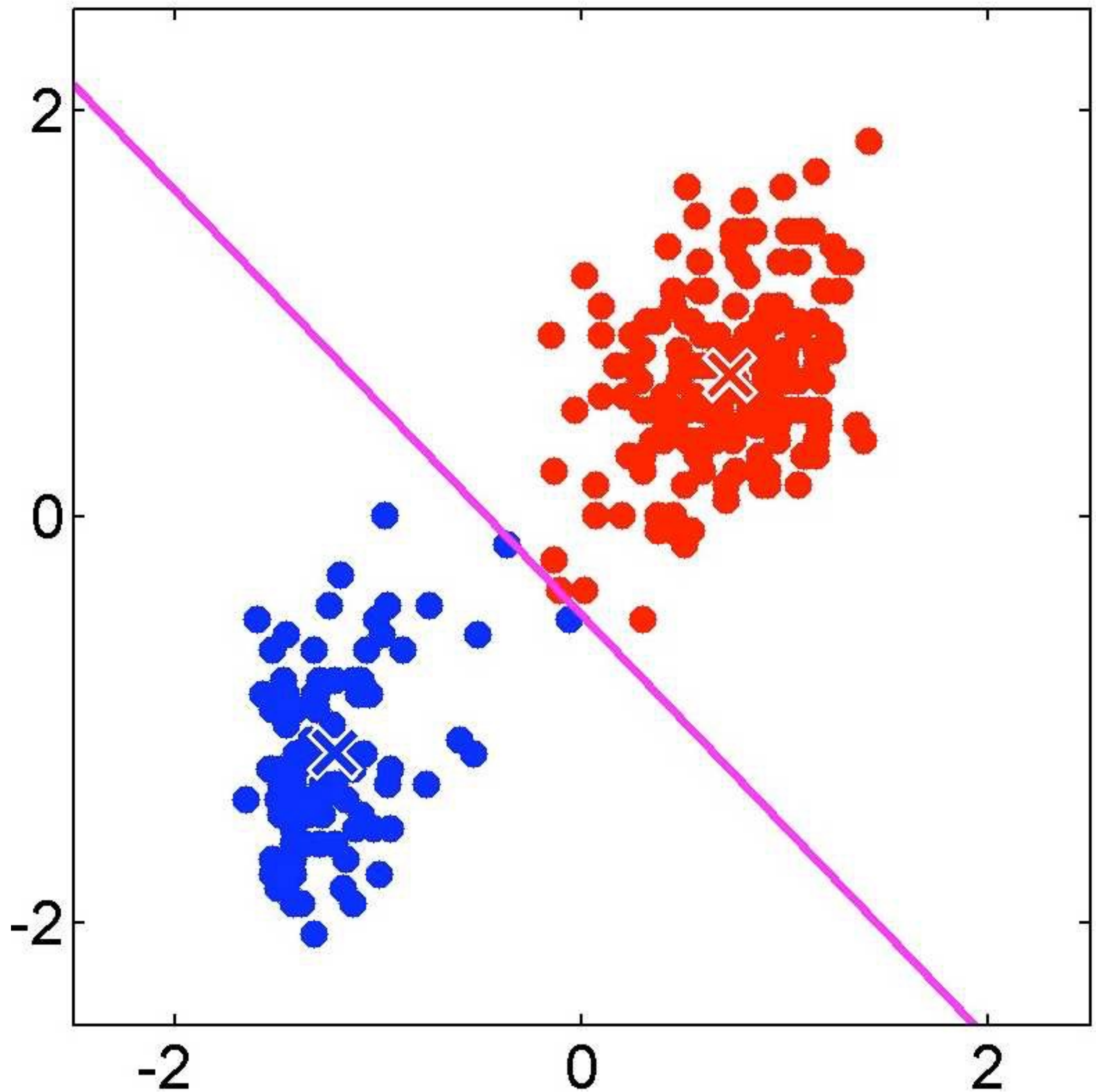
Assigning Points



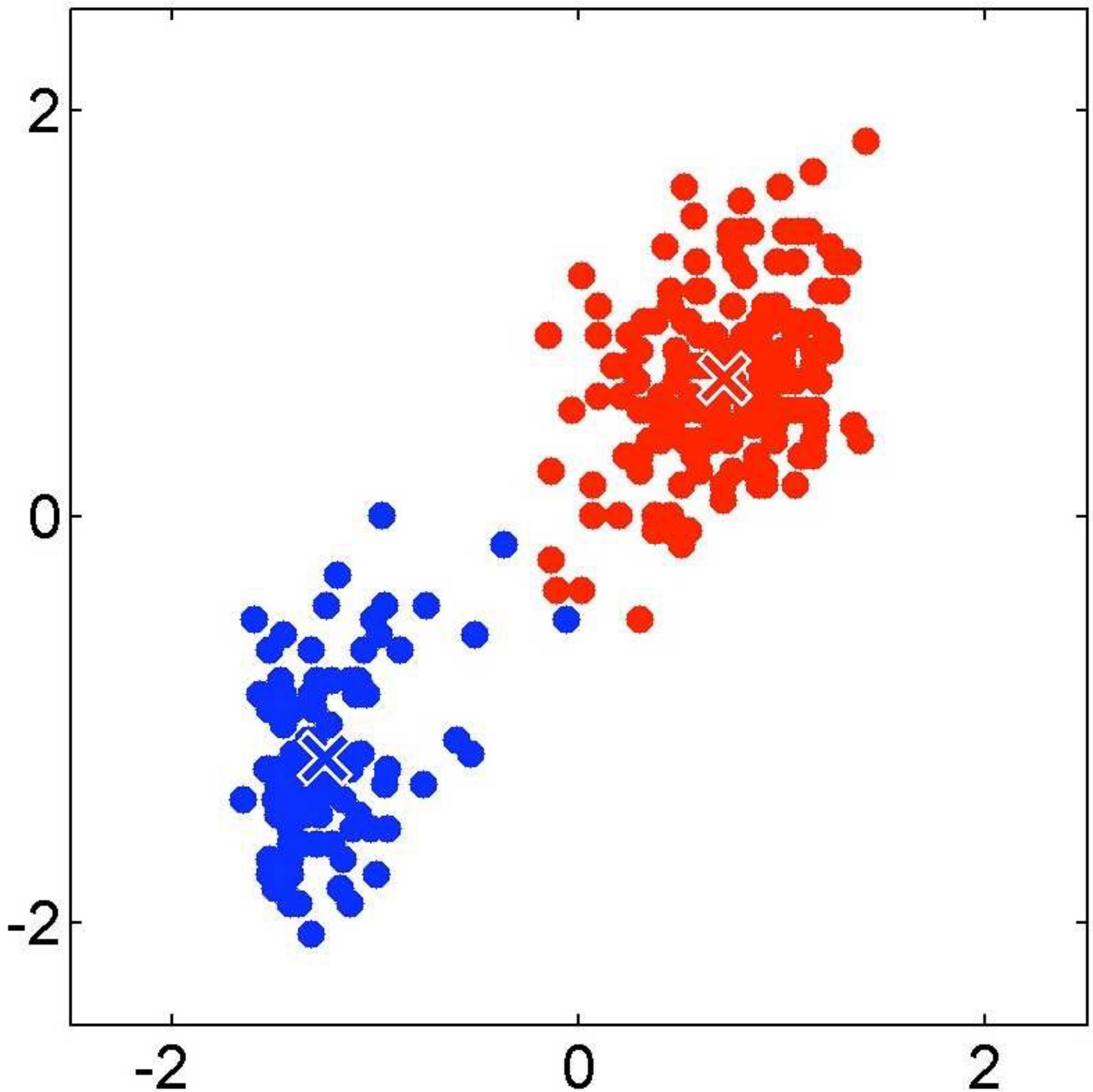
Recomputing the Cluster Centers



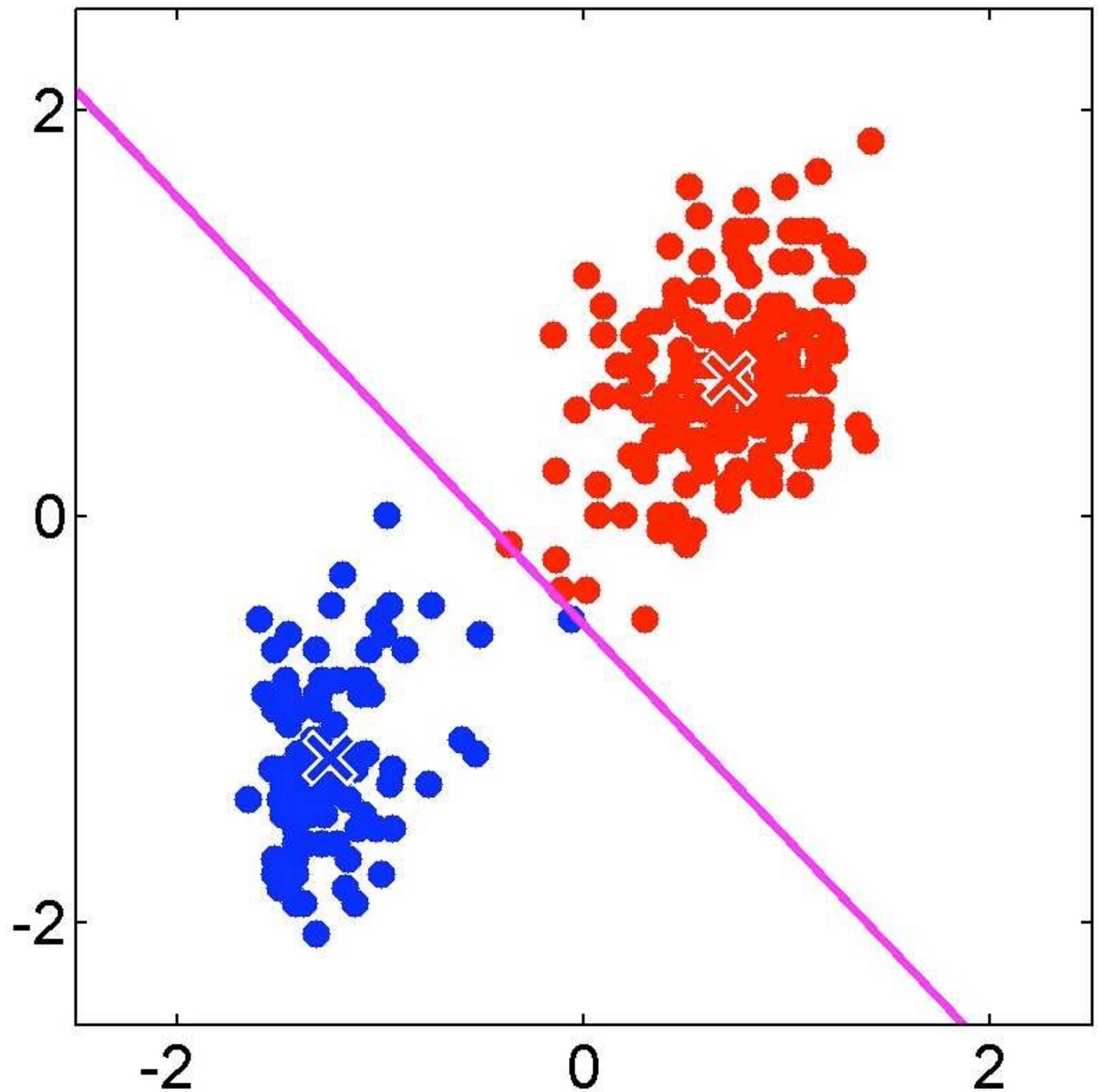
Assigning Points



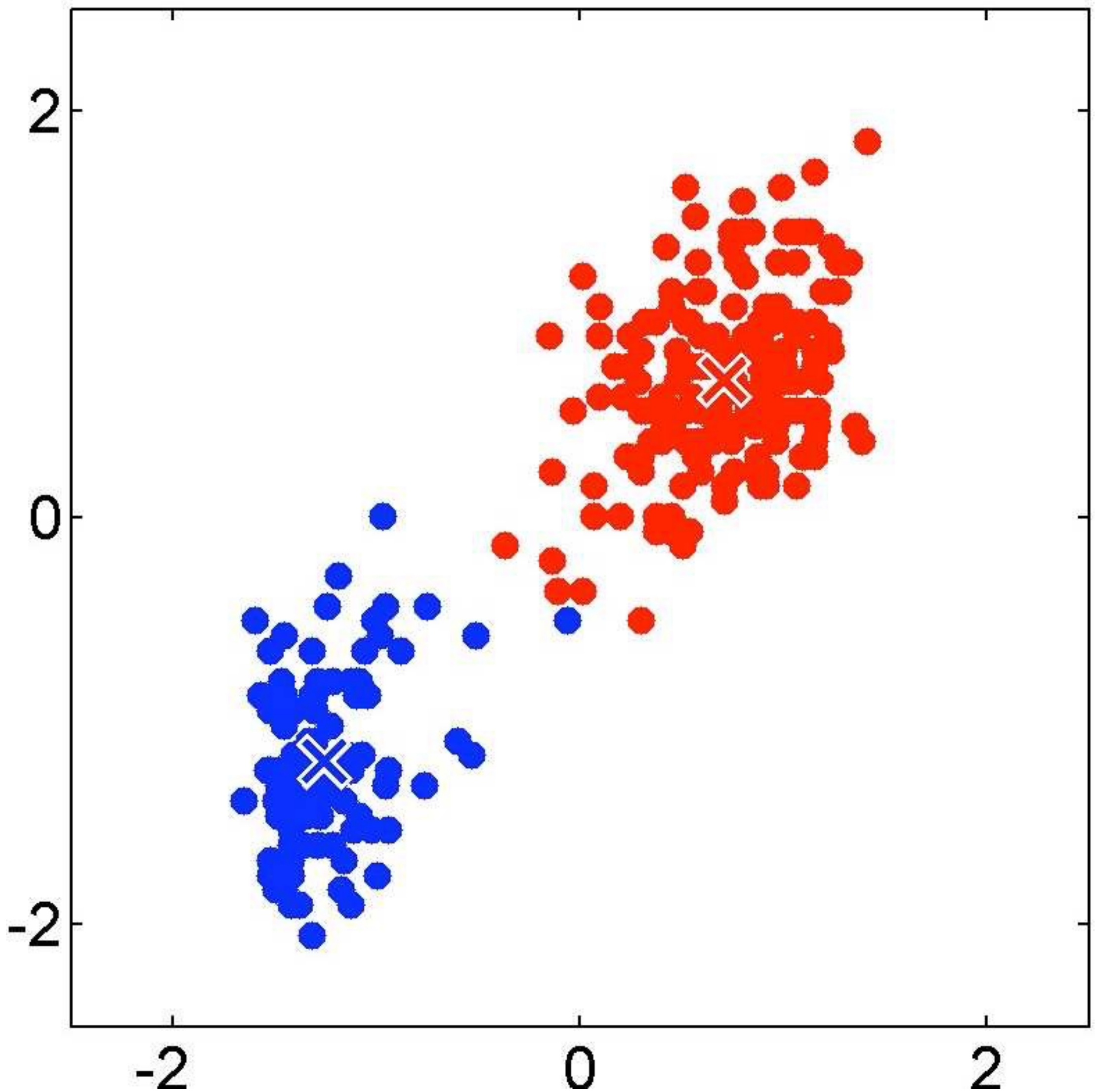
Recomputing the Cluster Centers



Assigning Points



Recomputing the Cluster Centers



Summary: K-means Clustering

- (Iterative) Algorithm

Randomly initialize k cluster centroids $\mu_1, \mu_2, \cdots, \mu_k \in \mathbb{R}^n$

Repeat{

for $i = 1$ to m

$c_i :=$ index (from 1 to k) of cluster centroid closest to $x^{(i)}$

for $k = 1$ to k

$\mu_k :=$ average (mean) of points assigned to cluster k

}

K-means: Optimization Point of View (optional)

- c_i = index of cluster $(1, 2, \cdots, k)$ to which example $x^{(i)}$ is currently assigned
- μ_k = cluster centroid
- μ_{c_i} = cluster centroid of cluster to which example $x^{(i)}$ has been assigned
- Optimization objective:

$$J(c_1, \cdots, c_m, \mu_1, \cdots, \mu_k) = \frac{1}{m} \sum_{i=1}^{m} \|x^{(i)} - \mu_{c_i}\|^2$$

$$\min_{c_1, \cdots, c_m, \ \mu_1, \cdots, \mu_k} J(c_1, \cdots, c_m, \mu_1, \cdots, \mu_k)$$

Expectation Maximization Algorithm

- It is a "chicken and egg" problem (dilemma)
  - Q: if we knew c_is, how would we determine which points to associate with each cluster center?
  - A: for each point $x^{(i)}$, choose closest c_i

  - Q: if we knew the cluster memberships, how do we get the centers?
  - A: choose c_i to be the mean of all points in the cluster

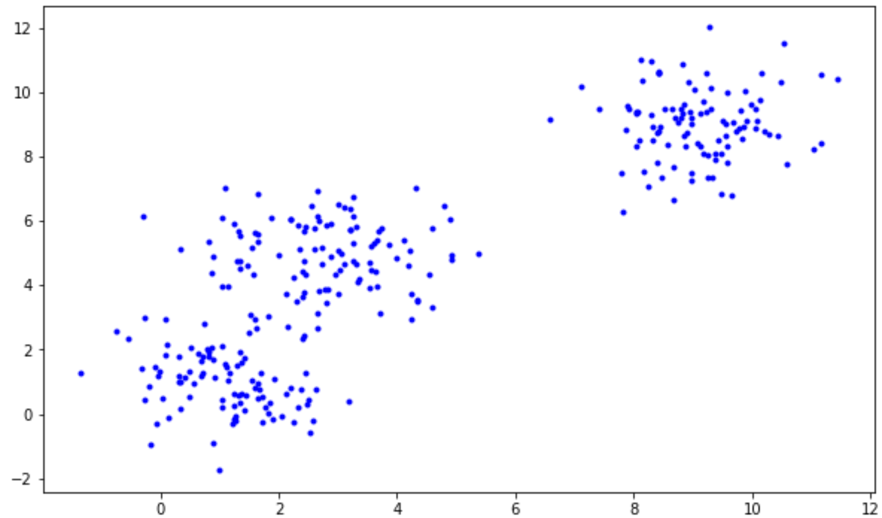
- Extension of K-means algorithm
  - A special case of Expectation Maximization (EM) algorithm
  - A special case of Gaussian Mixture Model (GMM)
  - Won't be discussed in this course

Python: Data Generation

```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

```
# data generation
G0 = np.random.multivariate_normal([1, 1], np.eye(2), 100)
G1 = np.random.multivariate_normal([3, 5], np.eye(2), 100)
G2 = np.random.multivariate_normal([9, 9], np.eye(2), 100)

X = np.vstack([G0, G1, G2])
X = np.asmatrix(X)
print(X.shape)

plt.figure(figsize=(10, 6))
plt.plot(X[:,0], X[:,1], 'b.')
plt.show()
```

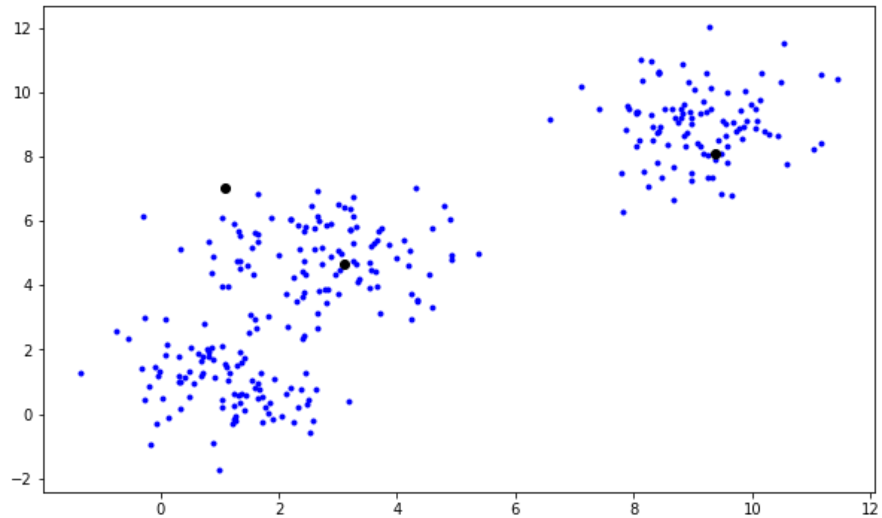
```
(300, 2)
```



Python: Random Initialization

```
# The number of clusters and data
k = 3
m = X.shape[0]

# ramdomly initialize mean points
mu = X[np.random.randint(0,300,k),:]
pre_mu = mu.copy()
print(mu)

plt.figure(figsize=(10, 6))
plt.plot(X[:,0], X[:,1], 'b.')
plt.plot(mu[:,0], mu[:,1], 'ko')
plt.show()
```

```
[[ 1.08723812  7.02505378]
 [ 9.37822573  8.07630763]
 [ 3.09510078  4.684617  ]]
```



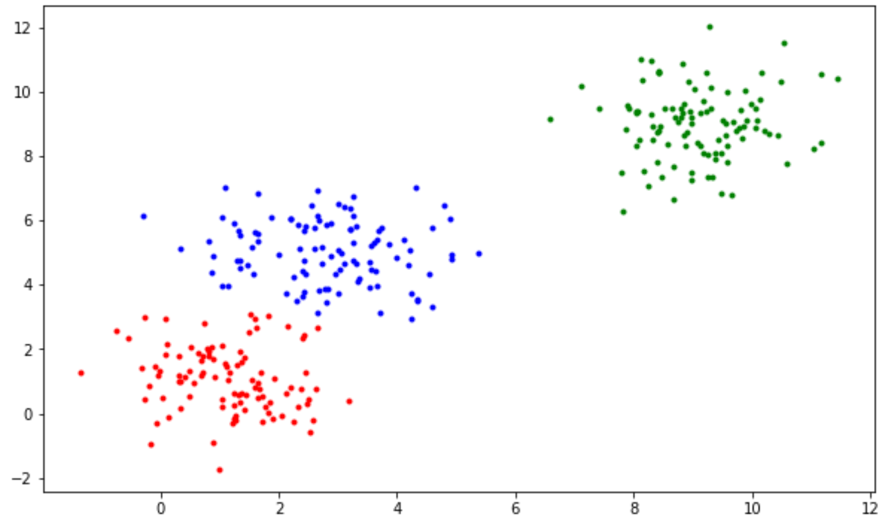
Python: K-Means

```
y = np.empty([m,1])

# Run K-means
for n_iter in range(500):
    for i in range(m):
        d0 = np.linalg.norm(X[i,:] - mu[0,:],2)
        d1 = np.linalg.norm(X[i,:] - mu[1,:],2)
        d2 = np.linalg.norm(X[i,:] - mu[2,:],2)

        y[i] = np.argmin([d0, d1, d2])

    err = 0
    for i in range(k):
        mu[i,:] = np.mean(X[np.where(y == i)[0]], axis=0)
        err += np.linalg.norm(pre_mu[i,:] - mu[i,:],2)

    pre_mu = mu.copy()

    if err < 1e-10:
        print("Iteration:", n_iter)
        break
```

```
Iteration: 4
```

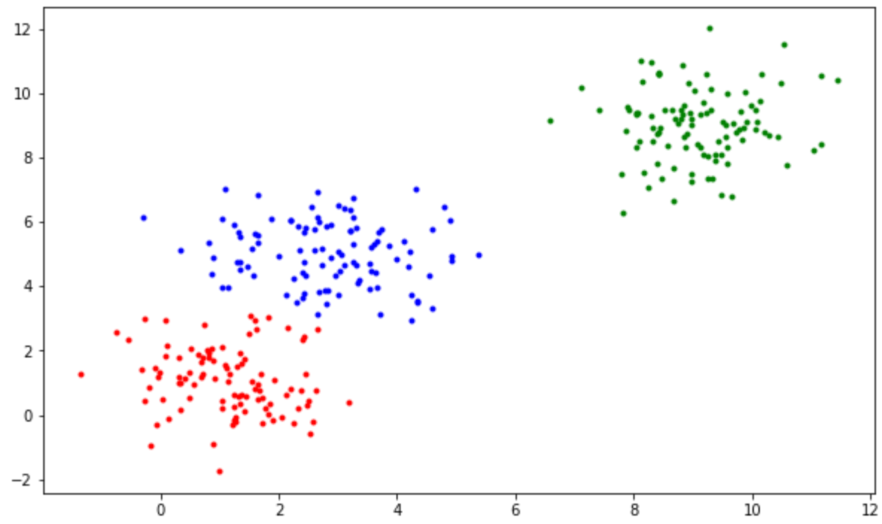
Python: Result

```
X0 = X[np.where(y==0)[0]]
X1 = X[np.where(y==1)[0]]
X2 = X[np.where(y==2)[0]]

plt.figure(figsize=(10, 6))
plt.plot(X0[:,0], X0[:,1], 'b.')
plt.plot(X1[:,0], X1[:,1], 'g.')
plt.plot(X2[:,0], X2[:,1], 'r.')
plt.show()
```



Python: K-Means in Scikit-learn

```
# use kmeans from the scikit-learn module

from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters = 3, random_state = 0)
kmeans.fit(X)

plt.figure(figsize=(10,6))
plt.plot(X[kmeans.labels_ == 0,0],X[kmeans.labels_ == 0,1],'b.')
plt.plot(X[kmeans.labels_ == 1,0],X[kmeans.labels_ == 1,1],'g.')
plt.plot(X[kmeans.labels_ == 2,0],X[kmeans.labels_ == 2,1],'r.')
plt.show()
```



Initialization Issues

- k-means is extremely sensitive to cluster center initialization

- Bad initialization can lead to
  - Poor convergence speed
  - Bad overall clustering

- Safeguarding measures:
  - Choose first center as one of the examples, second which is the farthest from the first, third which is the farthest from both, and so on.
  - Try multiple initialization and choose the best result

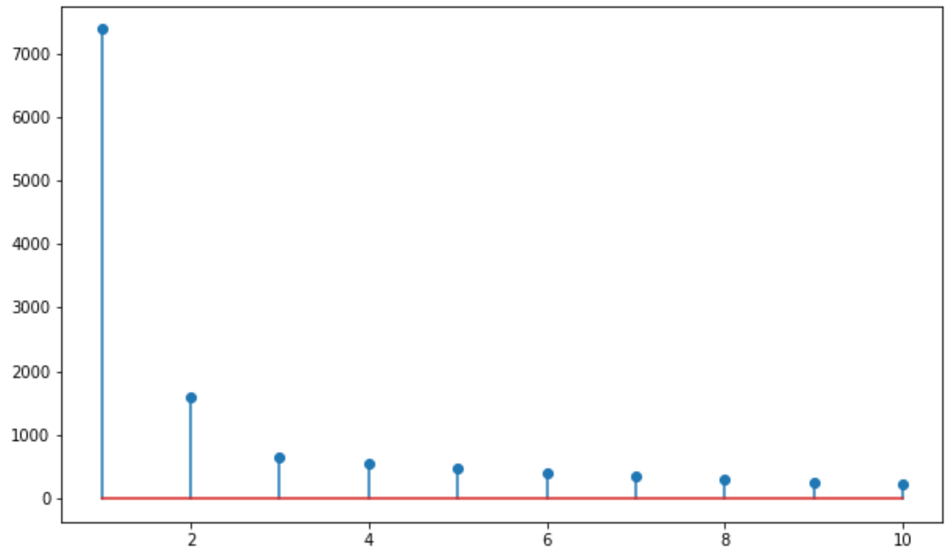
Choosing the Number of Clusters

- Idea: when adding another cluster does not give much better modeling of the data

- One way to select k for the K-means algorithm is to try different values of k, plot the K-means objective versus k, and look at the 'elbow-point' in the plot

Choosing the Number of Clusters

```
cost = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, random_state=0).fit(X)
    cost.append(abs(kmeans.score(X)))

plt.figure(figsize=(10,6))
plt.stem(range(1,11),cost)
plt.show()
```



K-means: Limitations

- Make hard assignments of points to clusters
  - A point either completely belongs to a cluster or not belongs at all
  - No notion of a soft assignment (*i.e.*, probability of being assigned to each cluster)
  - Gaussian mixture model (we will study later) and Fuzzy K-means allow soft assignments

- Sensitive to outlier examples
  - K-medians algorithm is a more robust alternative for data with outliers

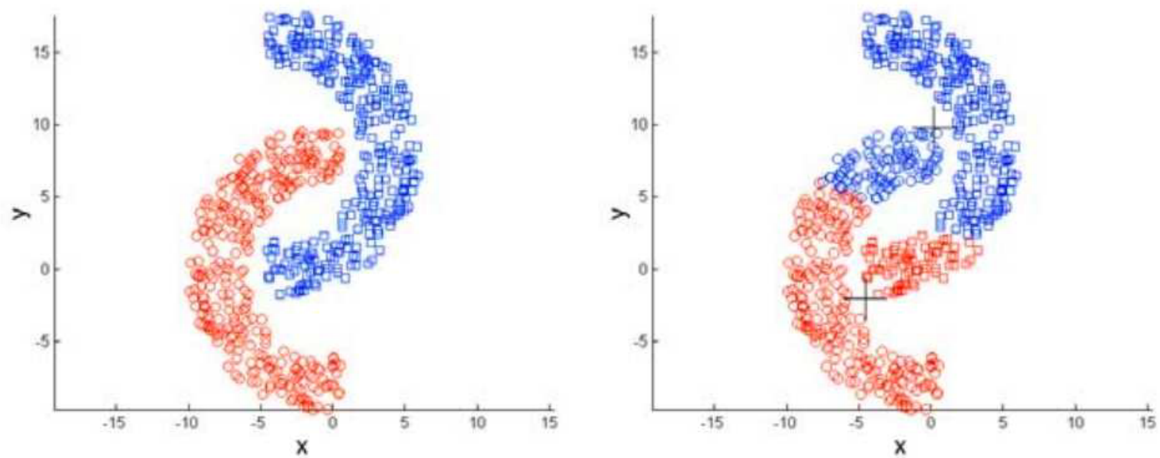
K-means: Limitations

- Works well only for round shaped, and of roughly equal sizes/density cluster

- Does badly if the cluster have non-convex shapes
  - Spectral clustering (we will study later) and Kernelized K-means can be an alternative

K-means: Limitations

- Non-convex/non-round-shaped cluster: standard K-means fails !



- (optional) Connectivity → networks → spectral partitioning

K-means: Limitations

- Clusters with different densities