

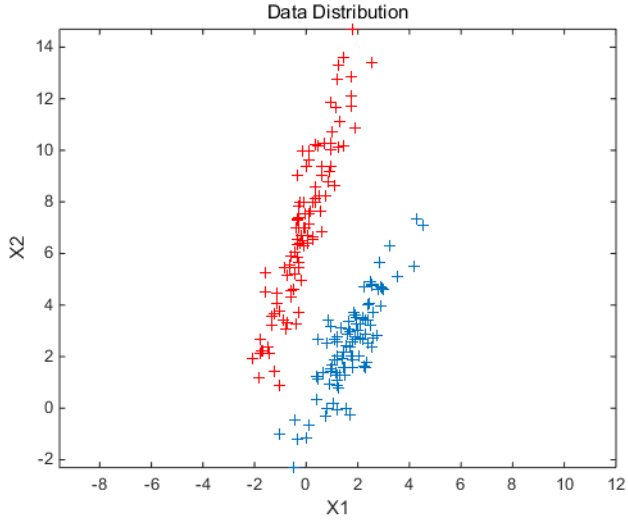
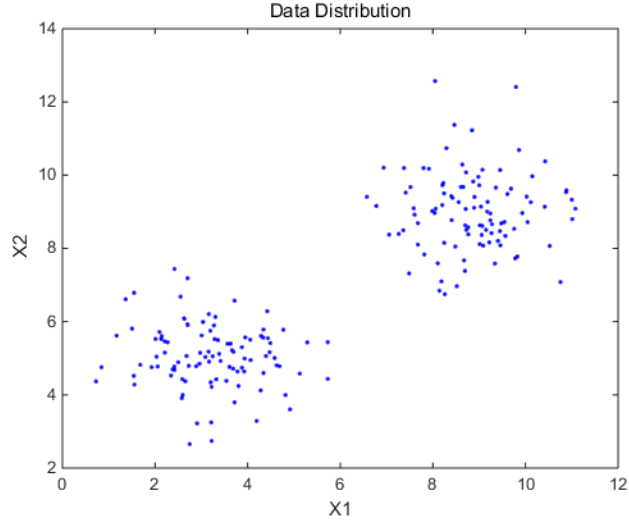


Clustering: K-means

Industrial AI Lab.

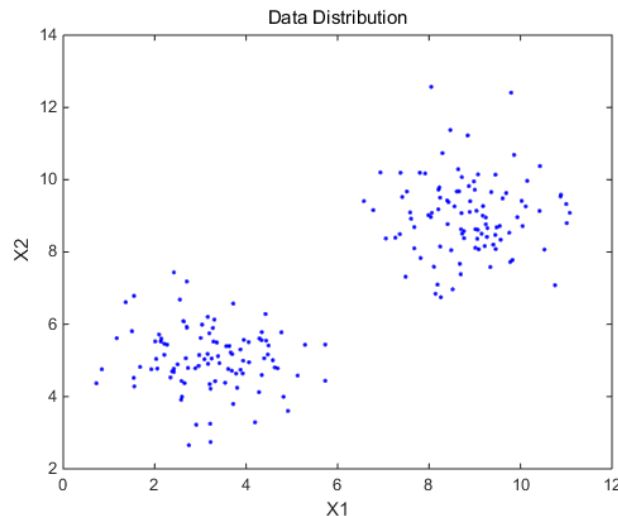
Prof. Seungchul Lee

Supervised vs. Unsupervised Learning

Supervised Learning	Unsupervised Learning
Building a model from labeled data	Clustering from unlabeled data
	
$\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ $\{y^{(1)}, y^{(2)}, \dots, y^{(m)}\} \Rightarrow$ Classification	$\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\} \Rightarrow$ Clustering

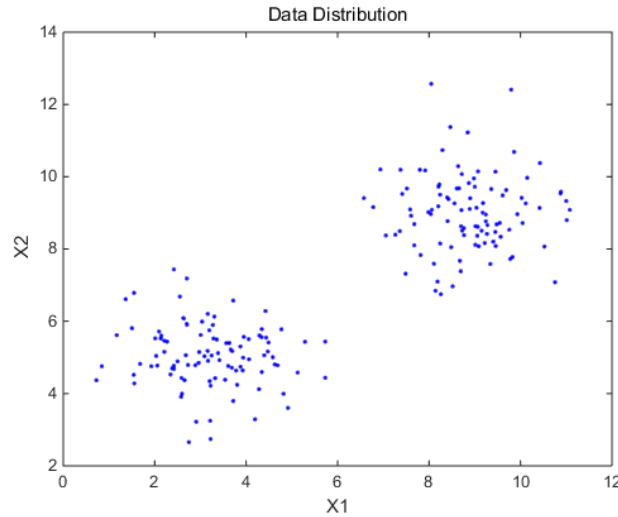
Data Clustering

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



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

Data Clustering: Similarity



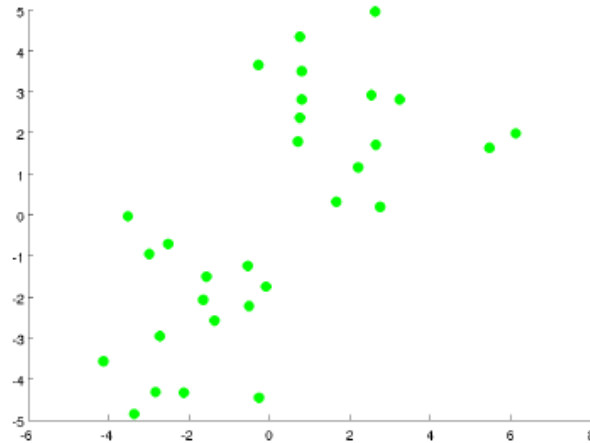
$$\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\} \Rightarrow \text{Clustering}$$

- 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

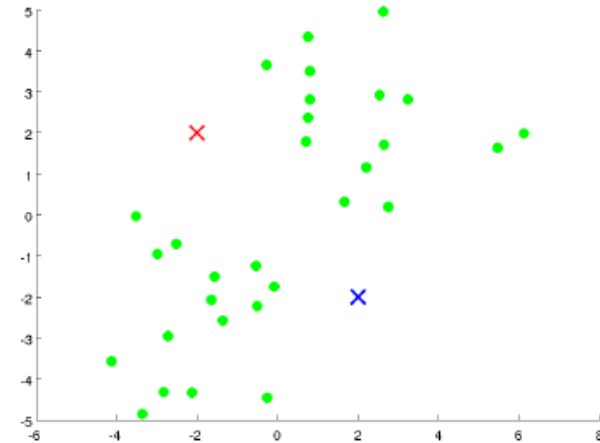
K-means: (Iterative) Algorithm

1) Initialization

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



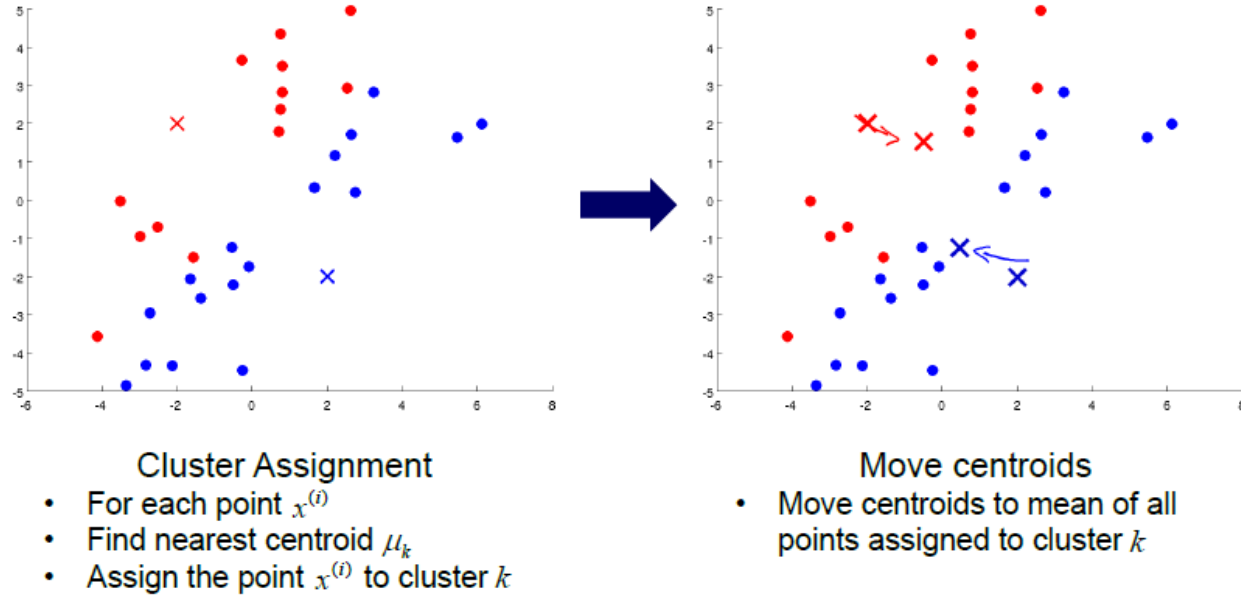
Training set data



Place centroids at random locations ($K=2$)

K-means: (Iterative) Algorithm

2) Iteration

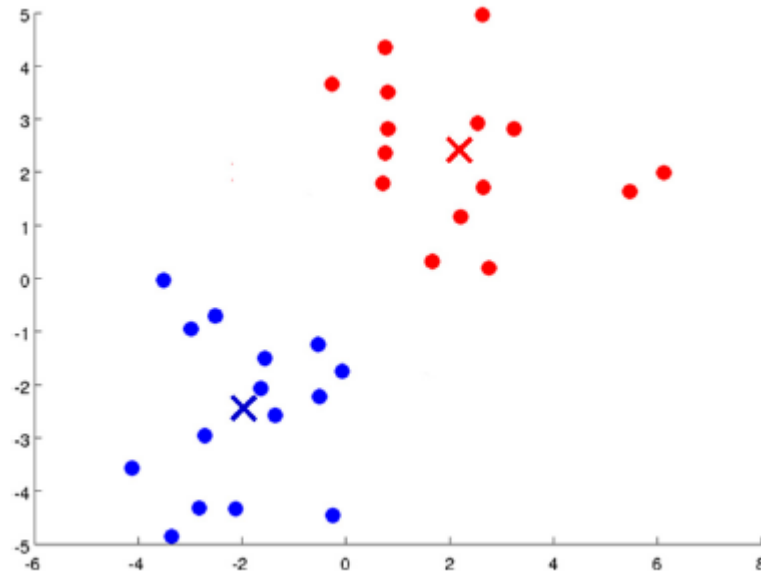


- Repeat until convergence
 - A possible convergence criteria: cluster centers do not change anymore

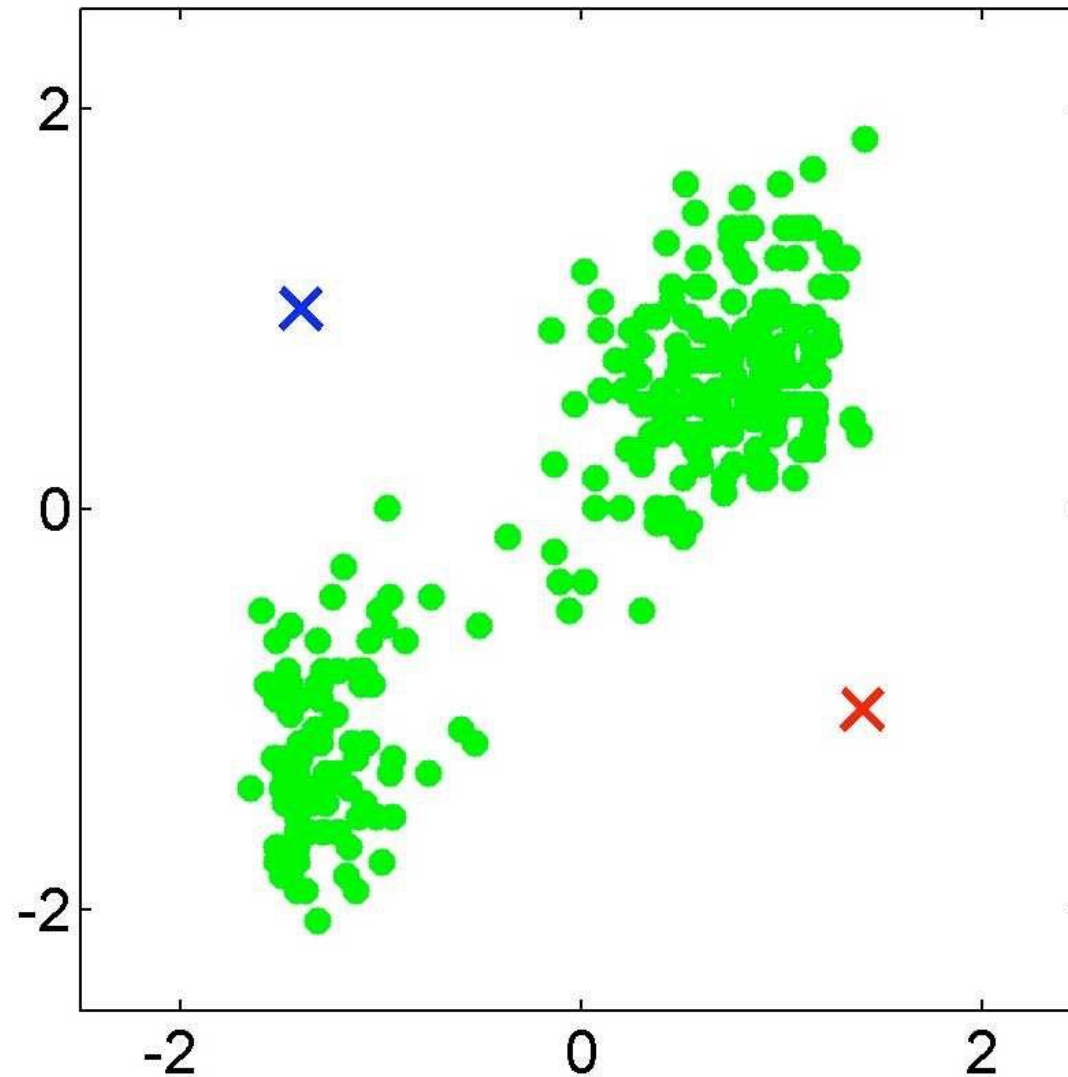
K-means: (Iterative) Algorithm

3) Output

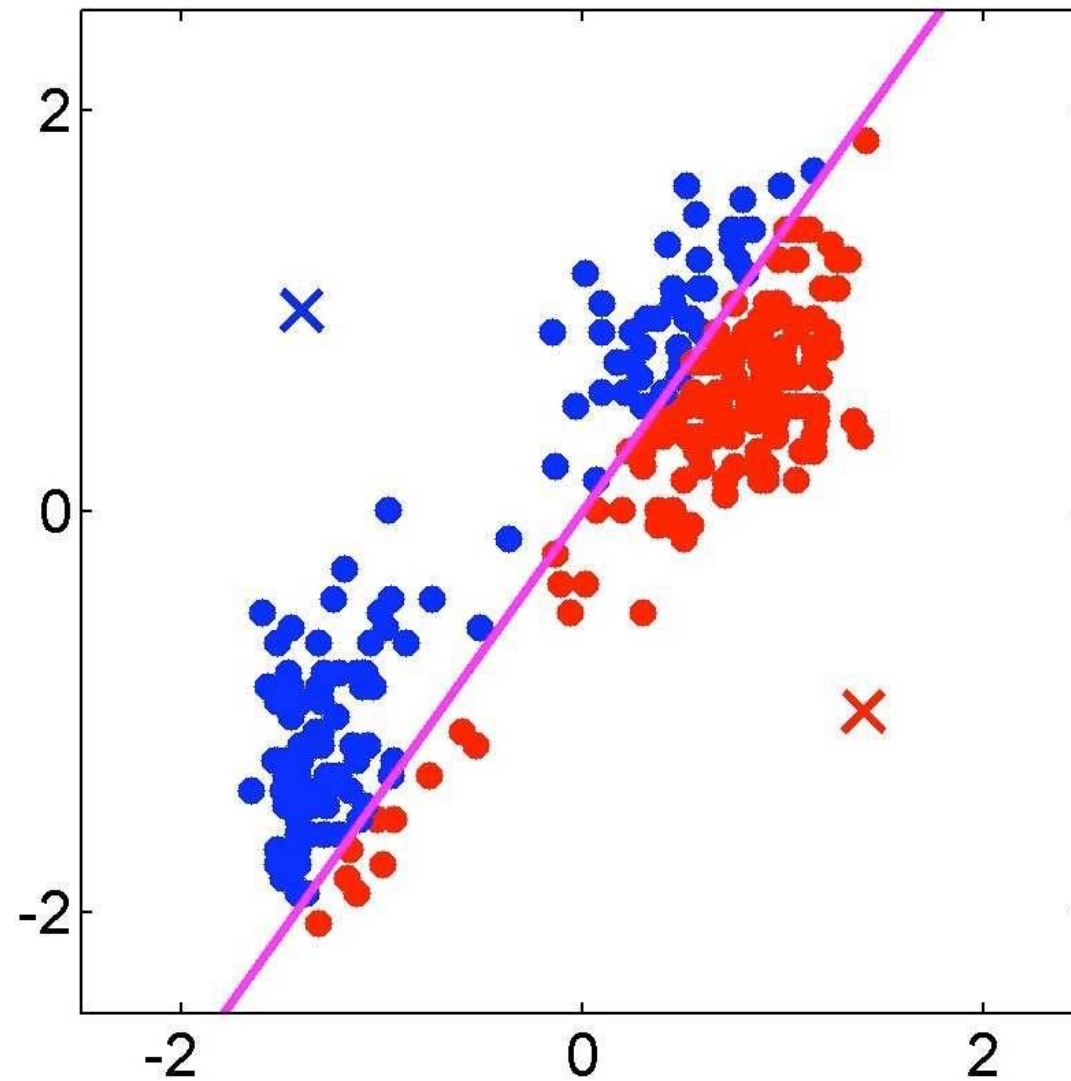
- c (label) : index (1 to k) of cluster centroid (centers)
- μ : averages (mean) of points assigned to cluster $\{\mu_1, \mu_2, \dots, \mu_k\}$



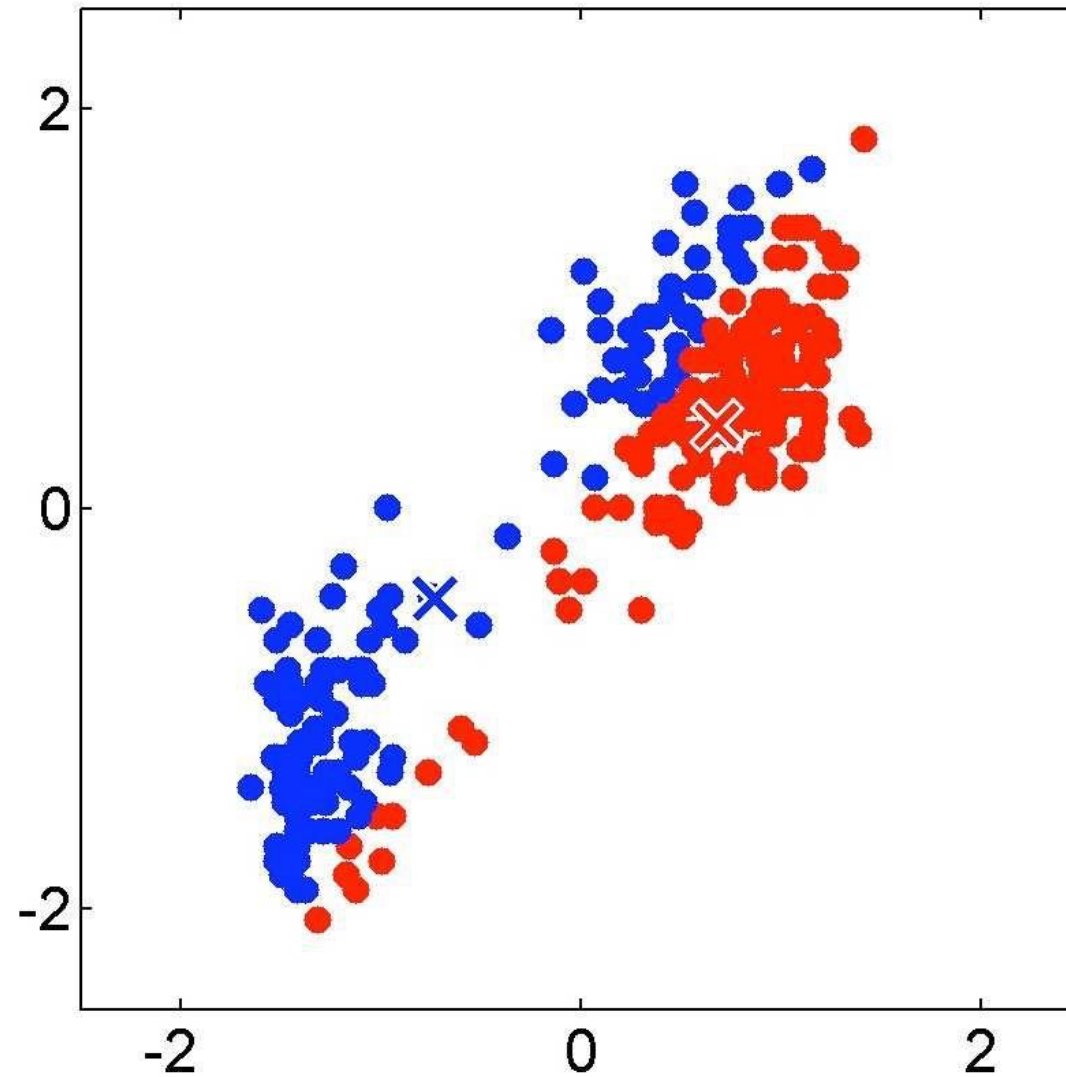
Initialization ($k = 2$)



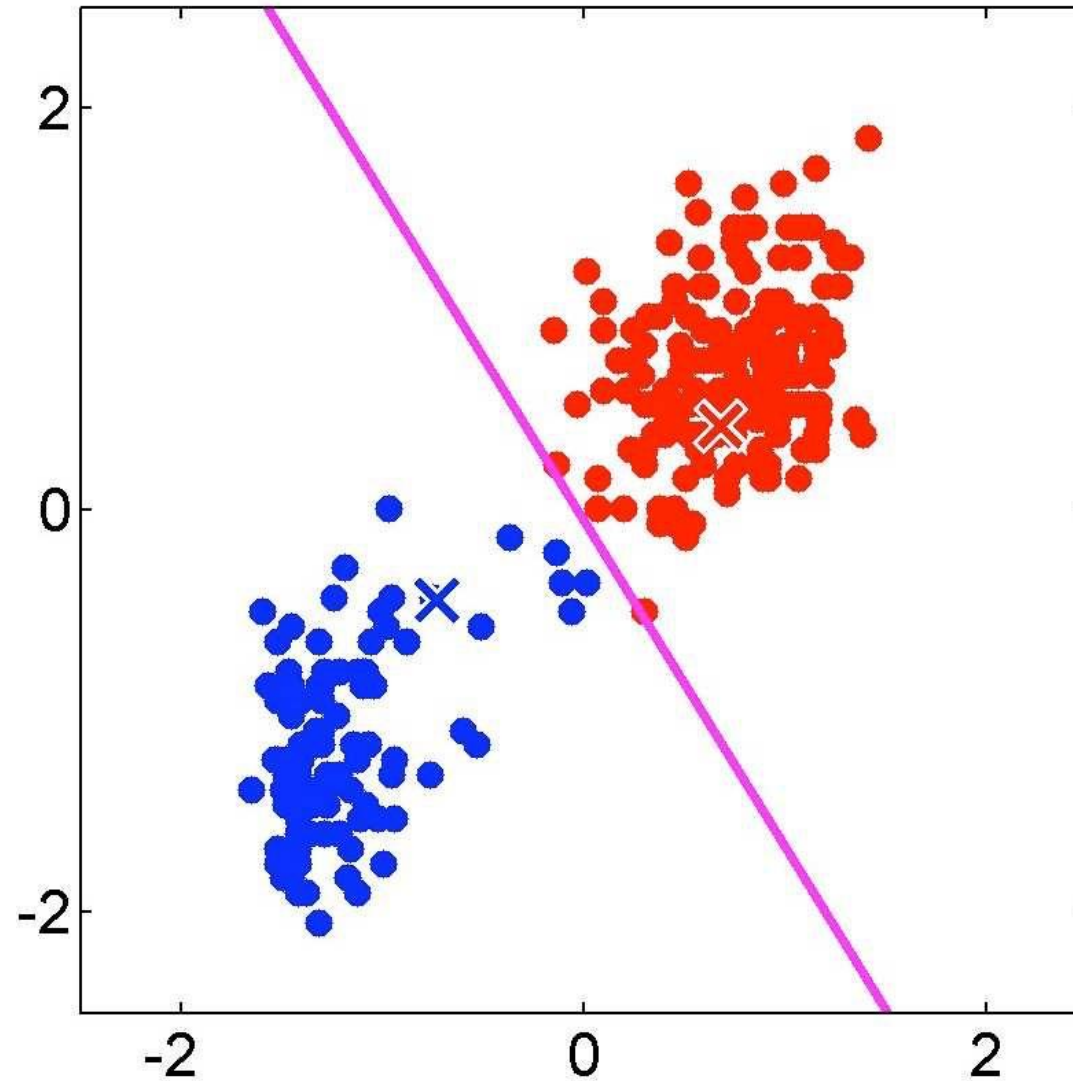
Assigning Points



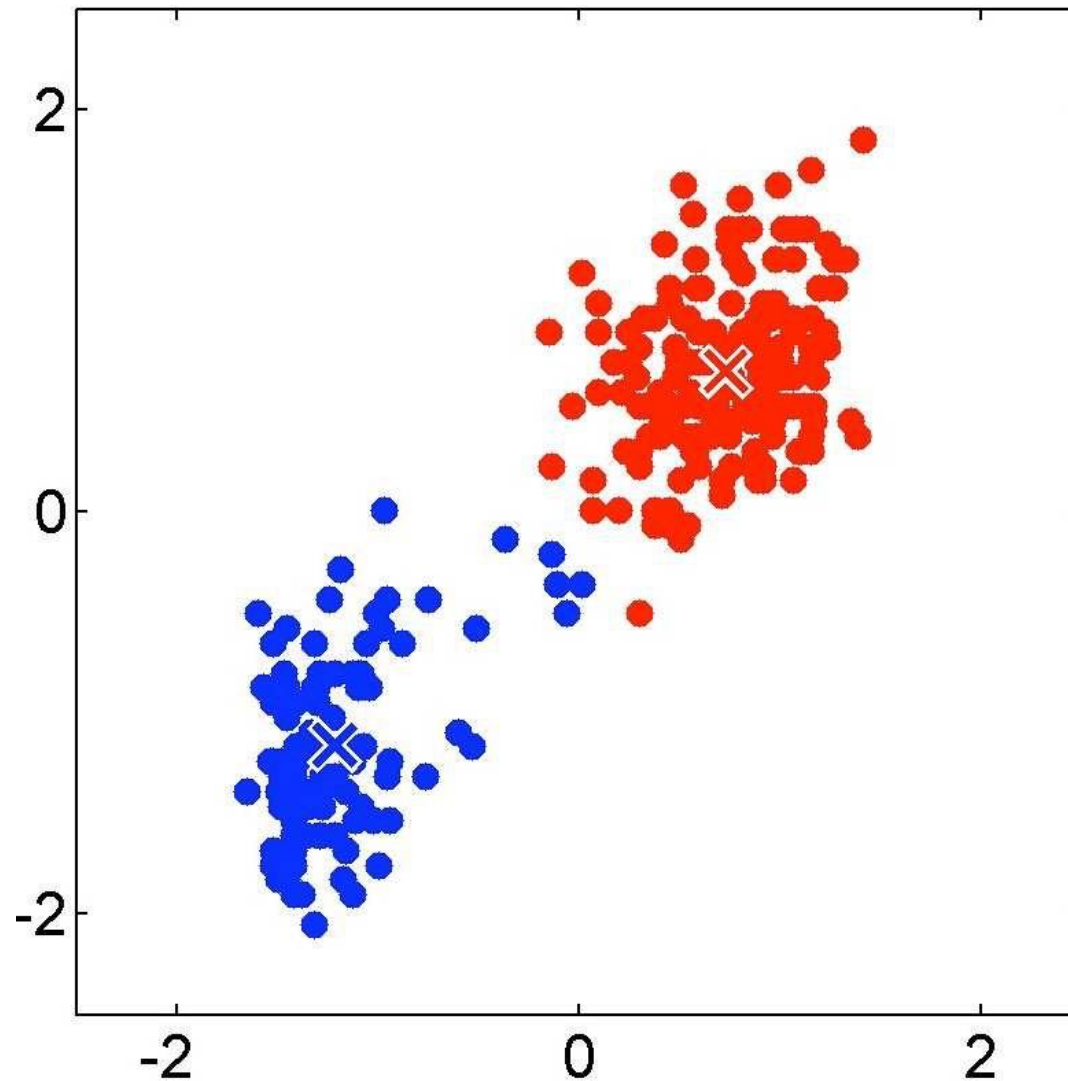
Recomputing the Cluster Centers



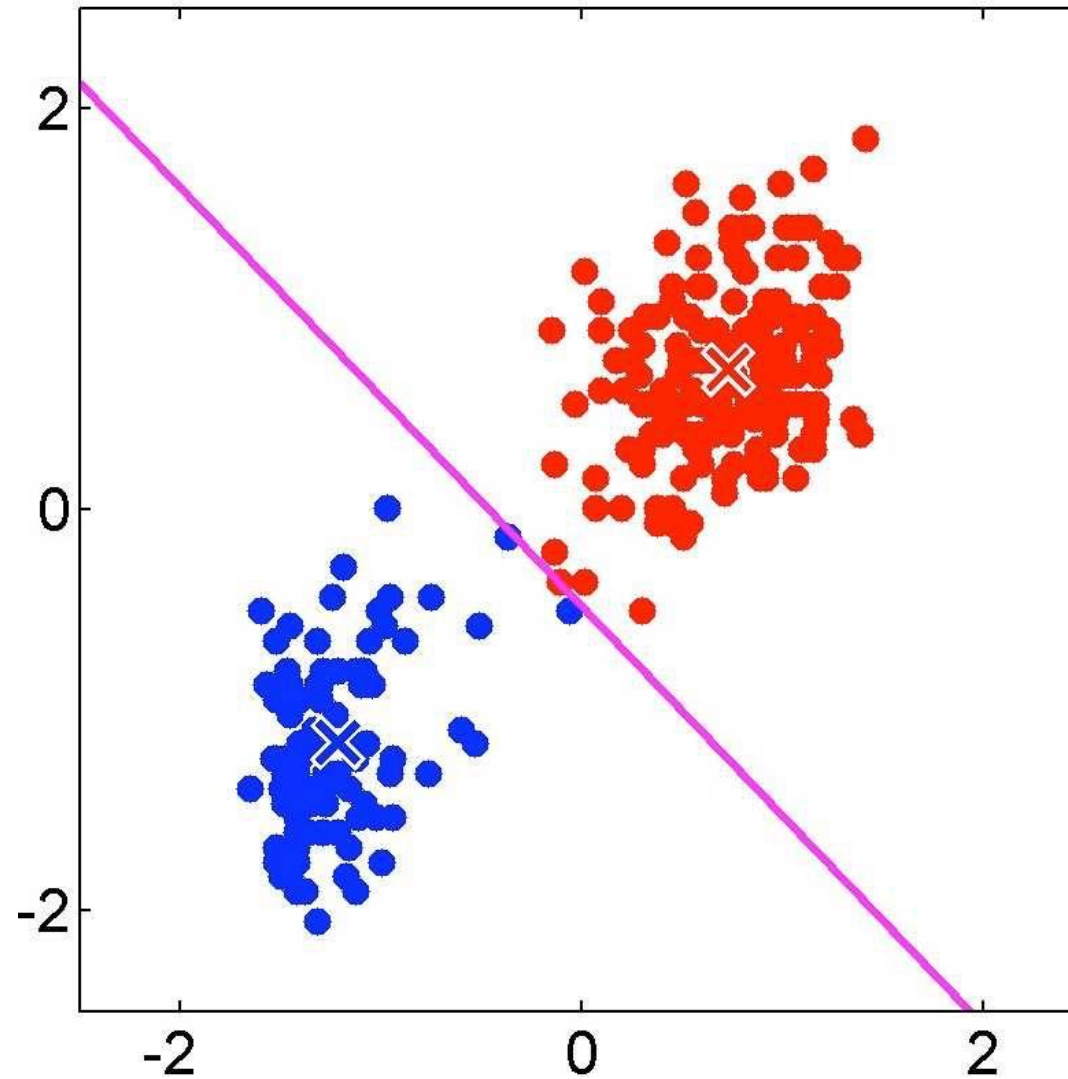
Assigning Points



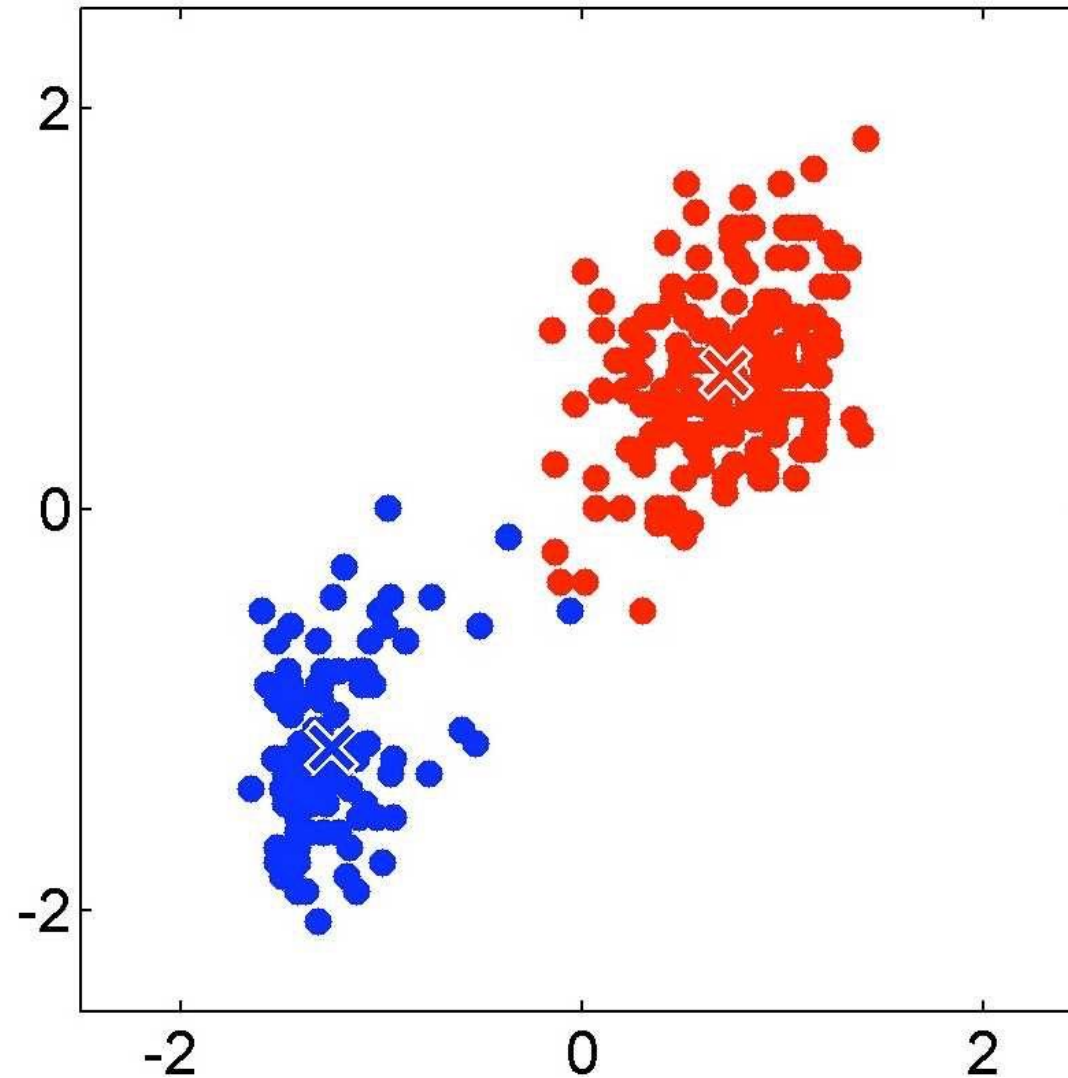
Recomputing the Cluster Centers



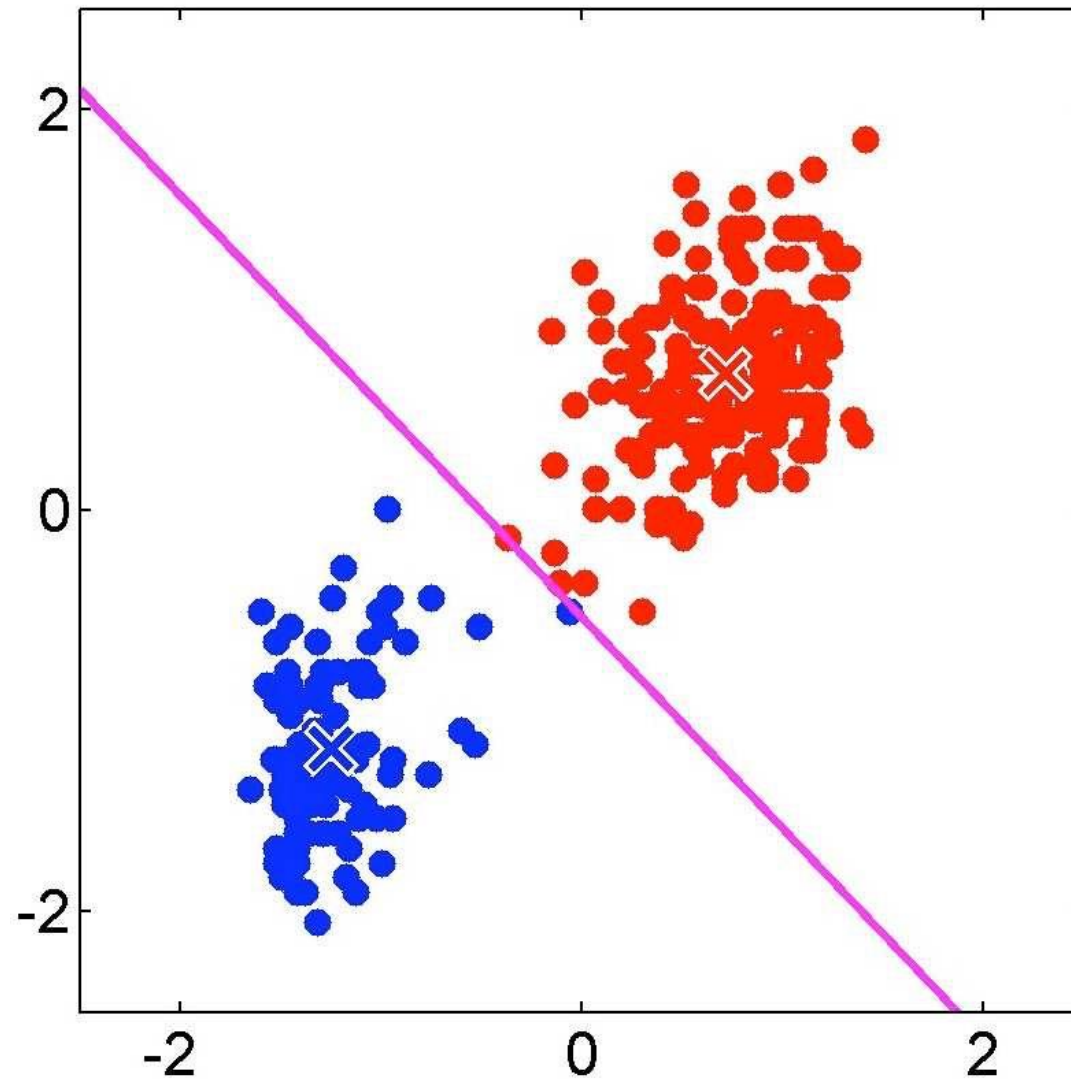
Assigning Points



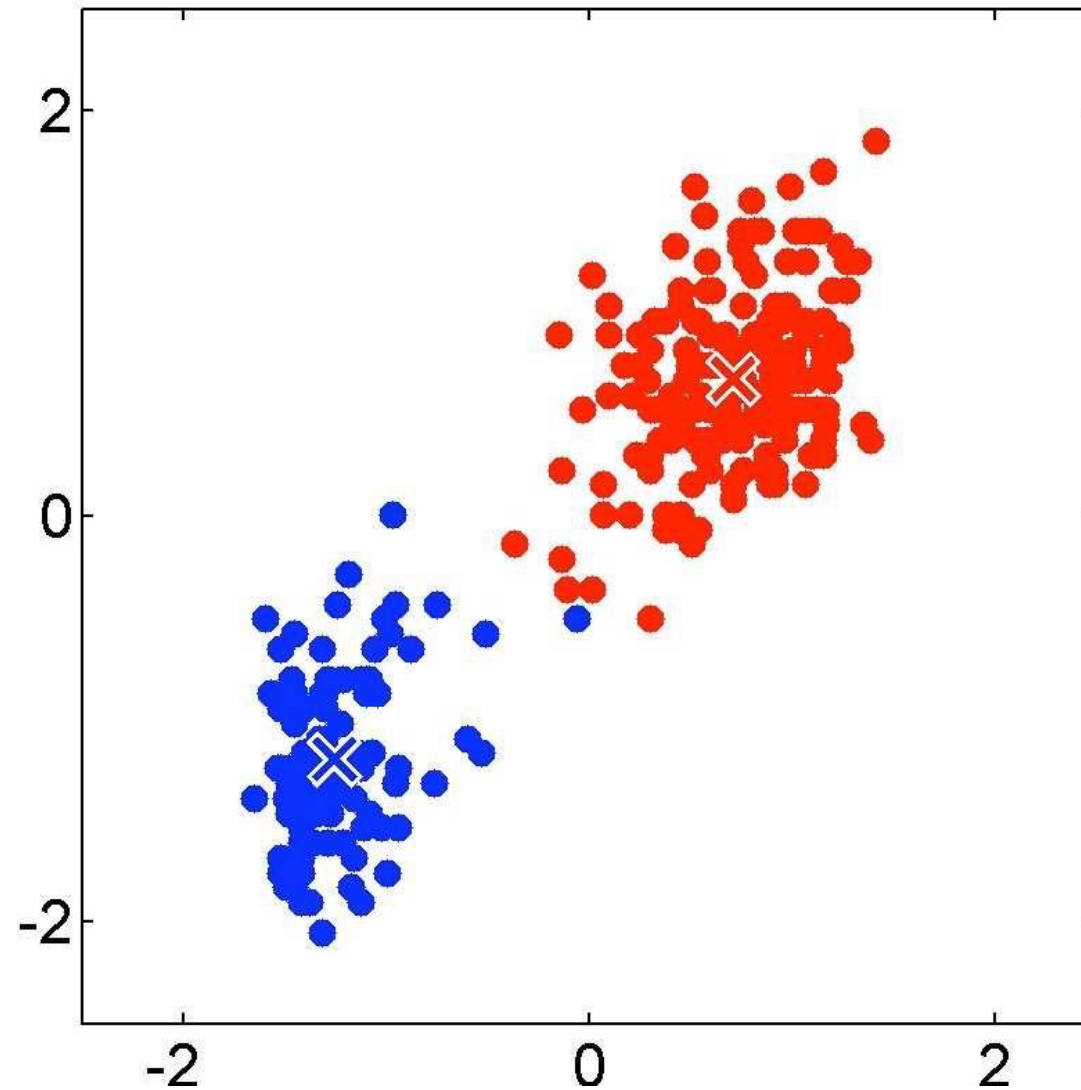
Recomputing the Cluster Centers



Assigning Points



Recomputing the Cluster Centers



Summary: K-means Clustering

- (Iterative) Algorithm

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

```
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 $(1, 2, \dots, k)$ 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:

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

Expectation Maximization (EM) Algorithm

- It is a "chicken and egg" problem (dilemma)
 - Q: if we knew c_i s, 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

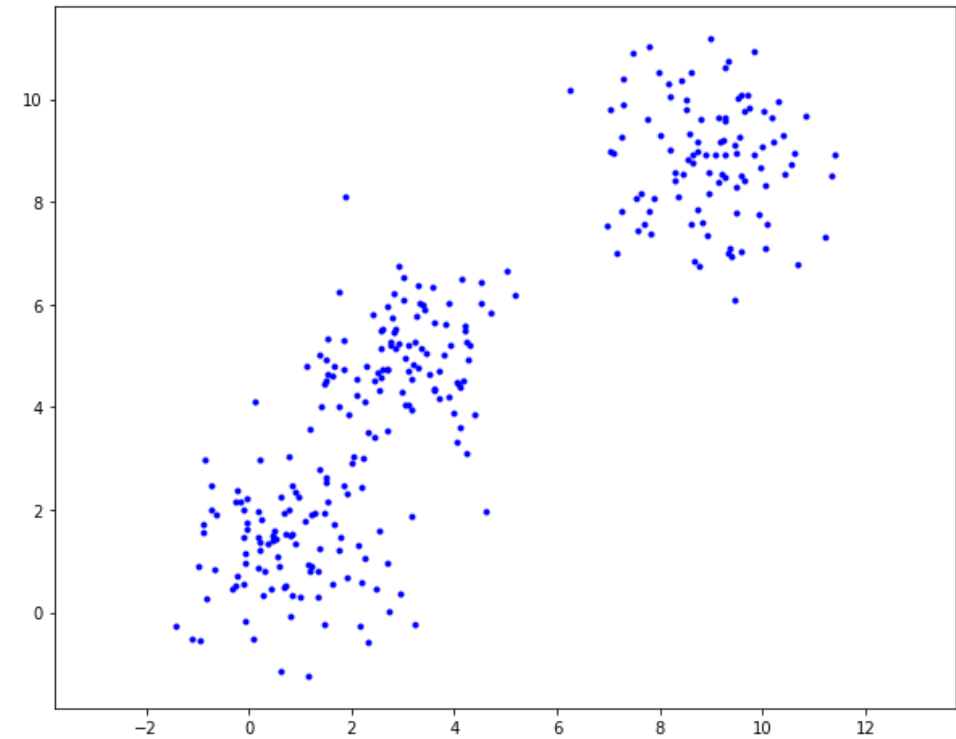
```
# data generation
```

```
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])
```



Python: Data Generation and Random Initialization

```
# 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])
```

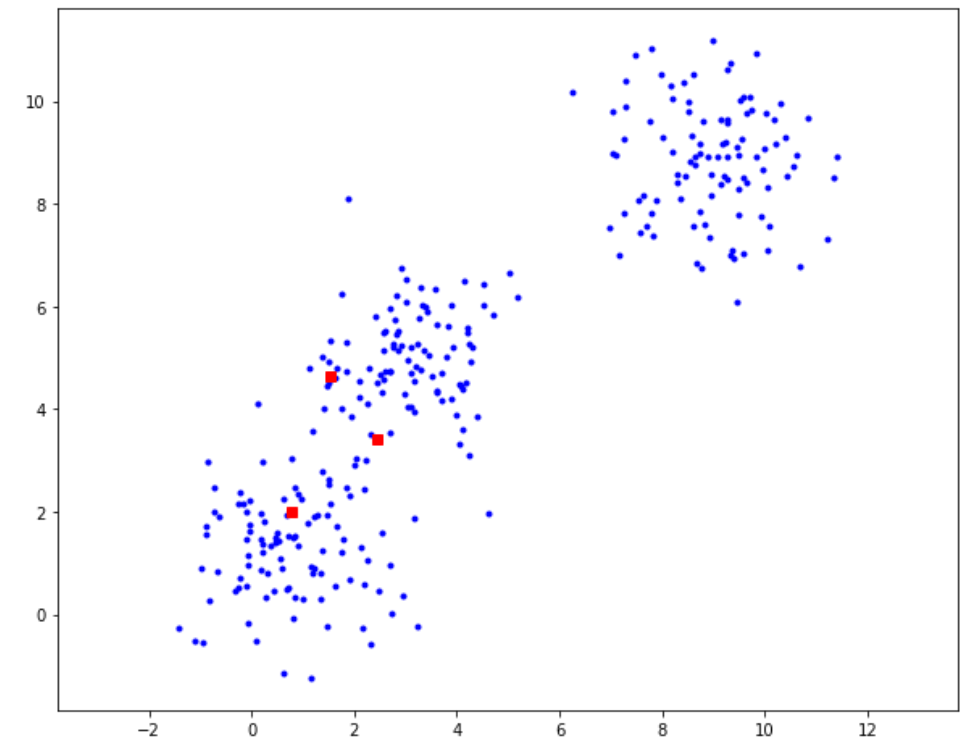
```
# The number of clusters and data
```

```
k = 3
```

```
m = X.shape[0]
```

```
# randomly initialize mean points
```

```
mu = X[np.random.randint(0, m, k), :]
```

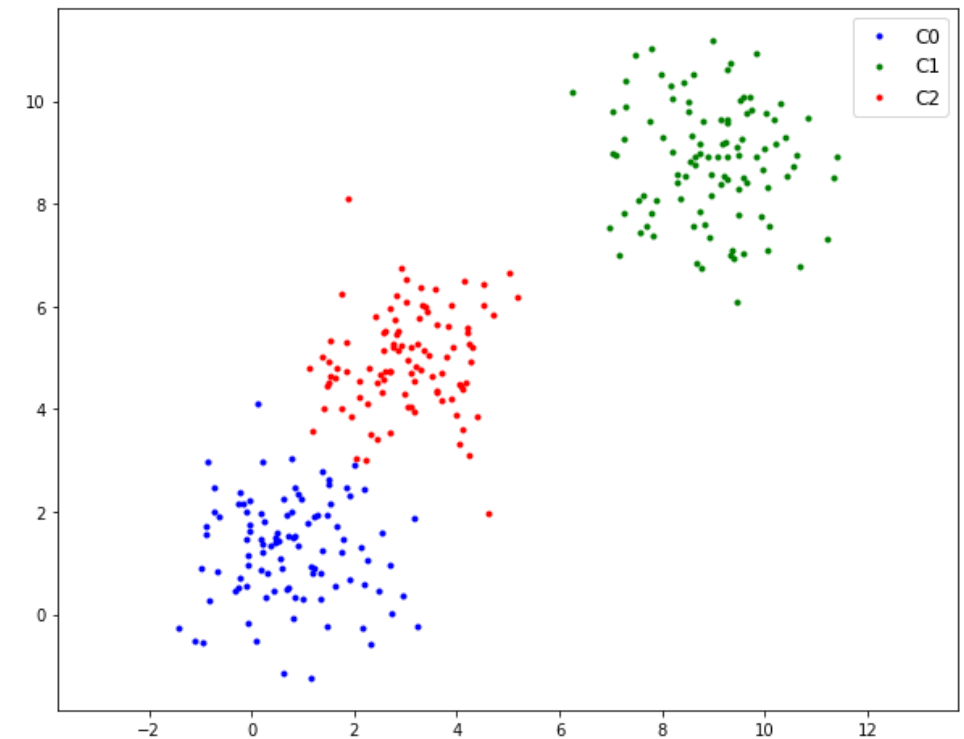


Python: K-Means

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

```
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  $k$   
}
```

```
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
```



Python: K-Means in Scikit-learn

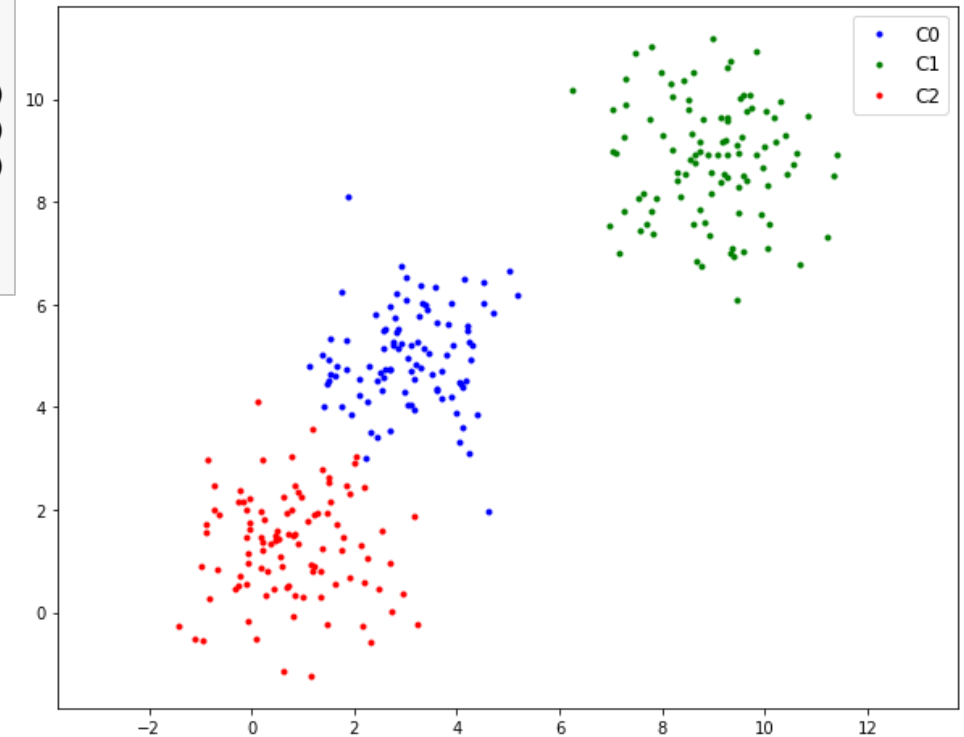


```
# use kmeans from the scikit-learn module

from sklearn.cluster import KMeans

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

plt.figure(figsize = (10,8))
plt.plot(X[kmeans.labels_ == 0, 0],X[kmeans.labels_ == 0, 1], 'b.', label = 'C0')
plt.plot(X[kmeans.labels_ == 1, 0],X[kmeans.labels_ == 1, 1], 'g.', label = 'C1')
plt.plot(X[kmeans.labels_ == 2, 0],X[kmeans.labels_ == 2, 1], 'r.', label = 'C2')
plt.axis('equal')
plt.legend(fontsize = 12)
plt.show()
```



Initialization Issues

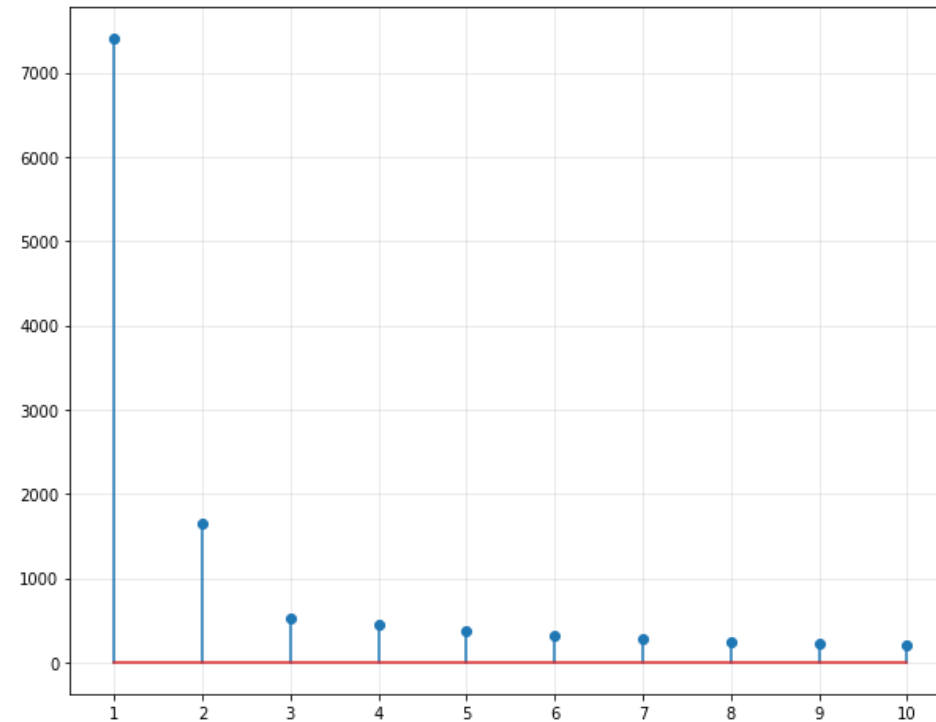
- k-means is extremely sensitive to cluster center initialization
- Bad initialization can lead to
 - Poor convergence speed
 - Bad overall clustering
- Safeguarding measures:
 - Choose first center as one of the examples, second which is the farthest from the first, third which is the farthest from 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)))
```



K-means: Limitations

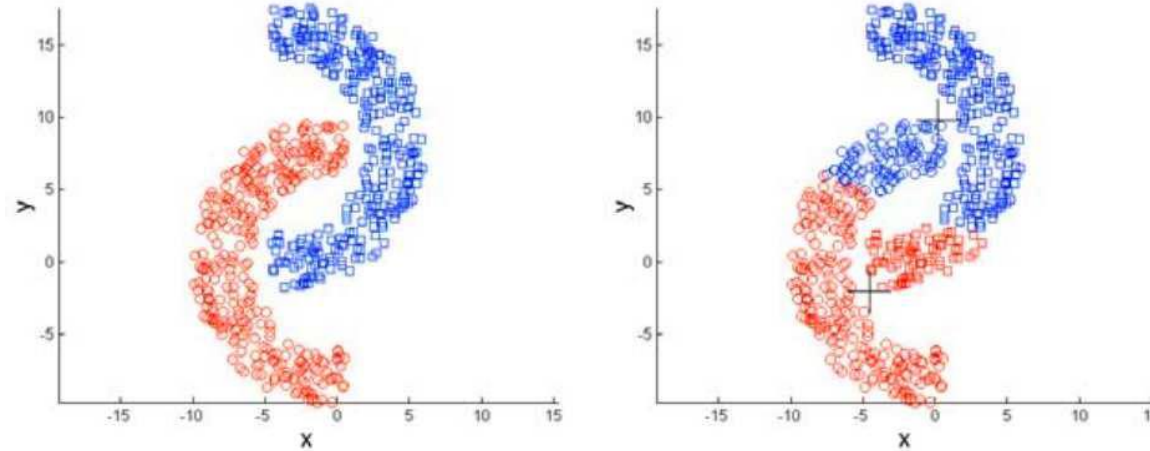
- Make hard assignments of points to clusters
 - A point either completely belongs to a cluster or not belongs at all
 - No notion of a soft assignment (*i.e.*, probability of being assigned to each cluster)
 - Gaussian mixture model (we will study later) and Fuzzy K-means allow soft assignments
- Sensitive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

