Lecture 16 - K-Means continued

In [1]:
```
import numpy as np
import numpy.random as npr

import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-colorblind')

from IPython.display import Image
```

Last class we introduced the **k-Means**, a clustering algorithm which lands as an unsupervised Machine Learning technique.

In [2]:
```
Image('figures/KMeans.png', width=800)
```

Out[2]:

Algorithm 1 K-Means Algorithm

1: Define number of clusters, K
2: Initialize cluster representatives
3: **repeat**
4: **for** i = 1 to N **do**
5: Determine the closest centroid representative, θ_k, for $\mathbf{x}_i$
6: Set label of data point i to the cluster whose centroid θ_k is closest
7: **end for**
8: **for** j = 1 to K **do**
9: Update cluster centroid representative θ_k to the mean of the points with cluster label k
10: **end for**
11: **until** Change in cluster centers is small

The K-Means algorithm uses **Expectation-Maximization (EM) as the optimization approach**:

- The E-Step is assigning that data points to the closest cluster
- The M-step is computing the centroid of each cluster

As we learned before, optimization with EM is called **Alternating Optimization** and therefore the final solution will be **dependent** on the initialization (of cluster centroids). Therefore the final solution may not be the *optimal* (also referred to as *global*) solution.

- The objective function for the K-Means algorithm is:

$$
\begin{aligned}
J(\Theta, U) &= \sum_{i=1}^{N} \sum_{k=1}^{K} u_{ik} d^2 \left(x_i, \theta_k\right) \\
&= \sum_{i=1}^{N} \sum_{k=1}^{K} u_{ik} \|x_i - \theta_k\|_2^2 \\
\text{such that } u_{ik} \in \{0, 1\} \text{ and } &\sum_{k=1}^{K} u_{ik} = 1
\end{aligned}
$$

where u_{ij} are cluster assignments, θ_j is the j^{th} cluster representative and $d\left(x_i, \theta_k\right)$ is the distance between data point x_i and cluster centroid θ_k.

- In K-Means, we want to optimize:

$$\arg_{\Theta, U} \min J(\Theta, U)$$

- If we use the Euclidean distance, does the k-means algorithm make any assumptions on cluster shape?
- Given a data set with an unknown number of clusters, can you come up with a strategy for determining the *right* number of clusters?
- Can we use other distance metrics in objective function $J(\Theta, U)$?

Observations

- Since K-Means uses distance-based measurements to determine the *similarity* between data points, it's recommended to **scale the data** since almost always the features in any data set would have different units of measurements (e.g. age vs income).
- Given K-Means Alternating Optimization approach, different initializations may lead to different clusters, as K-Means algorithm may be stuck in a *local optima* and not converge to the *global optima*. Therefore, it's recommended to run the algorithm using different initializations of centroids and pick the results of the run that yielded the lower sum of squared distance.
- One of the convergence criteria is to check whether the assignment of data points has not changed from one iteration to the next. This criteria of "assignment of points not changing" is the same as observing no change in the *within-cluster variation*:

$$\frac{1}{N_k} \sum_{i \in c_k} \|x_i - \theta_k\|_2^2$$

K-Means Applications

K-Means is a very popular algorithm and is commonly used in a variety of applications, such as: market segmentation, document clustering, image segmentation, image compression, etc.

The *goal* usually when we undergo a cluster analysis is either:

1. Get a meaningful intuition of the structure of the data we are dealing with.
2. Cluster-then-predict where different models will be built for different subgroups. An example of that is clustering patients into different subgroups (based on some feature map) and build a model for each subgroup to predict the probability of the risk of having a heart attack.

Let's take a look at two case applications for K-Means: Image Compression and Data Segmentation.

Example: K-Means as Data Segmentation

In [1]:

```
from scipy import spatial
from scipy.spatial.distance import cdist
from numpy.random import permutation

def KMeans(X, K, MaxIter = 10000, Thresh = 1e-5, Visualization = True):
    '''This function implements K-Means algorithm.
    Where the data X will be partitioned into K clusters.
    Data matrix X is of size NxD, where N is the number of points and D the di
mension of the data.'''

    #Initialize Cluster Centers by drawing Randomly from Data (can use other
    # methods for initialization...)
    N       = X.shape[0] # number of data points
    rp      = npr.permutation(N); # shuffle the data (random permutation)
    centers = X[rp[0:K],:]; # select first K data points as cluster centroids

    if Visualization:
        fig = plt.figure()
        plt.subplot(1,2,1)
        plt.scatter(X[:,0],X[:,1],c='b')
        plt.plot(centers[:,0],centers[:,1],'xr',markersize=20)
        plt.title('Randomly select K='+str(K)+' data points as Centroids')
        plt.show()

    diff    = np.inf;
    iter    = 0;
    while((diff > Thresh) & (iter < MaxIter)):
        D   = spatial.distance.cdist(X, centers) # compute the distance of eve
ry point to every cluster centroid
        L   = np.argmin(D, axis=1) # Assign data to closest cluster representa
tive (using Euclidean distance)

        # Visualization
        if Visualization:
            plt.subplot(1,2,1)
            plt.scatter(X[:,0],X[:,1],c=L)
            plt.plot(centers[:,0],centers[:,1],'xr',markersize=20)
            plt.title('Membership Assignment')

        #Update cluster centers
        centersPrev = centers.copy()
        for i in range(K):
            centers[i,:] = np.mean(X[L == i,:], axis=0) # New cluster centroid
s will be the average of all points assigned to it

        # Visualization
        if Visualization:
            plt.subplot(1,2,2)
            plt.scatter(X[:,0],X[:,1],c=L)
            plt.plot(centers[:,0],centers[:,1],'xr',markersize=20)
            plt.title('Update Cluster Centroids')
            plt.pause(1)

        #Update diff & iteration count for stopping criteria
        diff = np.linalg.norm(centersPrev - centers)
        iter = iter+1
```

```
    return centers, L

# Plotting function for synthetically generated data
def Plotting(X, true_labels, L, name):
    if len(true_labels)>0:
        fig = plt.figure(figsize=(12,4))
        fig.add_subplot(1,2,1)
        plt.scatter(X[:,0],X[:,1],c=true_labels)
        plt.title(name)
    else:
        fig = plt.figure(figsize=(12,4))
        fig.add_subplot(1,2,1)
        plt.scatter(X[:,0],X[:,1],c='b')
        plt.title(name)
    fig.add_subplot(1,2,2)
    plt.scatter(X[:, 0], X[:, 1], c=L)
    plt.title("K-Means Clustering Results")
```

In [2]:
```
from sklearn import datasets

# Create synthetically generate data
n_samples = 1500
X1, T1 = datasets.make_blobs(n_samples=n_samples,centers=3,cluster_std=1)
# Blobs data
X2, T2 = datasets.make_blobs(n_samples=n_samples,cluster_std=[1.0, 2.5, 0.5],c
enters=3) # Different Variance Blobs data
X3, T3 = datasets.make_moons(n_samples=n_samples, noise=.05)
# Moons data
X4, T4 = datasets.make_circles(n_samples, noise=.05, factor=0.5)
# Circles data
X5     = np.dot(X1, [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]])
# Anisotropicly distributed data
X6     = np.vstack((X1[T1 == 0][:500], X1[T1 == 1][:100], X1[T1 == 2][:10]))
# Unevenly sized Blobs data
T6     = np.hstack((np.zeros(500),np.ones(100),2*np.ones(10)))
```

In [ ]:
```
NoClusters = 3
C1, L1 = KMeans(X1, NoClusters, Visualization = True)
Plotting(X1, T1, L1, 'Blobs')
```

In [ ]:
```
NoClusters = 3
C2, L2 = KMeans(X2, NoClusters, Visualization = True)
Plotting(X2, T2, L2, 'Different Variance Blobs')
```

In [ ]:
```
NoClusters = 2
C3, L3 = KMeans(X3, NoClusters, Visualization = False)
Plotting(X3, T3, L3, 'Moons')
```

In [ ]:
```
NoClusters = 2
C4, L4 = KMeans(X4, NoClusters, Visualization = False)
Plotting(X4, T4, L4, 'Circles')
```

In [ ]:
```
NoClusters = 3
C5, L5 = KMeans(X5, NoClusters, Visualization = False)
Plotting(X5, T1, L5, 'Anisotropicly distributed data')
```

In [ ]:
```
NoClusters = 3
C6, L6 = KMeans(X6, NoClusters, Visualization = False)
Plotting(X6, T6, L6, 'Unevenly sized Blobs')
```

In [ ]:
```

```