DM2: Probabilistic Graphical Modeles

4 Implementation - Gaussian Mixture

In [1]:
```
%matplotlib inline
from __future__ import division

import pandas as pd
import numpy as np
from scipy.stats import norm, multivariate_normal
import math
import random

import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
import matplotlib.cm as cm

plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = 12,12

em_train = pd.read_csv('EMGaussian.data',header=None,sep=' ')
em_test = pd.read_csv('EMGaussian.test',header=None,sep=' ')


# Inspired from http://stackoverflow.com/questions/12301071/multidimen
sional-confidence-intervals
def plot_cov_ellipse(cov, pos, nstd=1.645, ax=None, fill=False, **kwar
gs):
    """
    Plots an `nstd` sigma error ellipse based on the specified covaria
nce
    matrix (`cov`). Additional keyword arguments are passed on to the
    ellipse patch artist.

    Parameters
    ----------
        cov : The 2x2 covariance matrix to base the ellipse on
        pos : The location of the center of the ellipse. Expects a 2-e
lement
            sequence of [x0, y0].
        nstd : The radius of the ellipse in numbers of standard deviat
ions.
            Defaults to 1.645 standard deviations (for 90% of the mass
of the
            Gaussian distribution)
        ax : The axis that the ellipse will be plotted on. Defaults to
the
            current axis.
        Additional keyword arguments are pass on to the ellipse patch.

    Returns
    -------
        A matplotlib ellipse artist
    """
    def eigsorted(cov):
        vals, vecs = np.linalg.eigh(cov)
        order = vals.argsort()[::-1]
        return vals[order], vecs[:,order]

    if ax is None:
        ax = plt.gca()

    vals, vecs = eigsorted(cov)
    theta = np.degrees(np.arctan2(*vecs[:,0][::-1]))

    # Width and height are "full" widths, not radius
    width, height = 2 * nstd * np.sqrt(vals)
    ellip = Ellipse(xy=pos, width=width, height=height, angle=theta, f
ill=False, **kwargs)

    ax.add_artist(ellip)
    return ellip
```

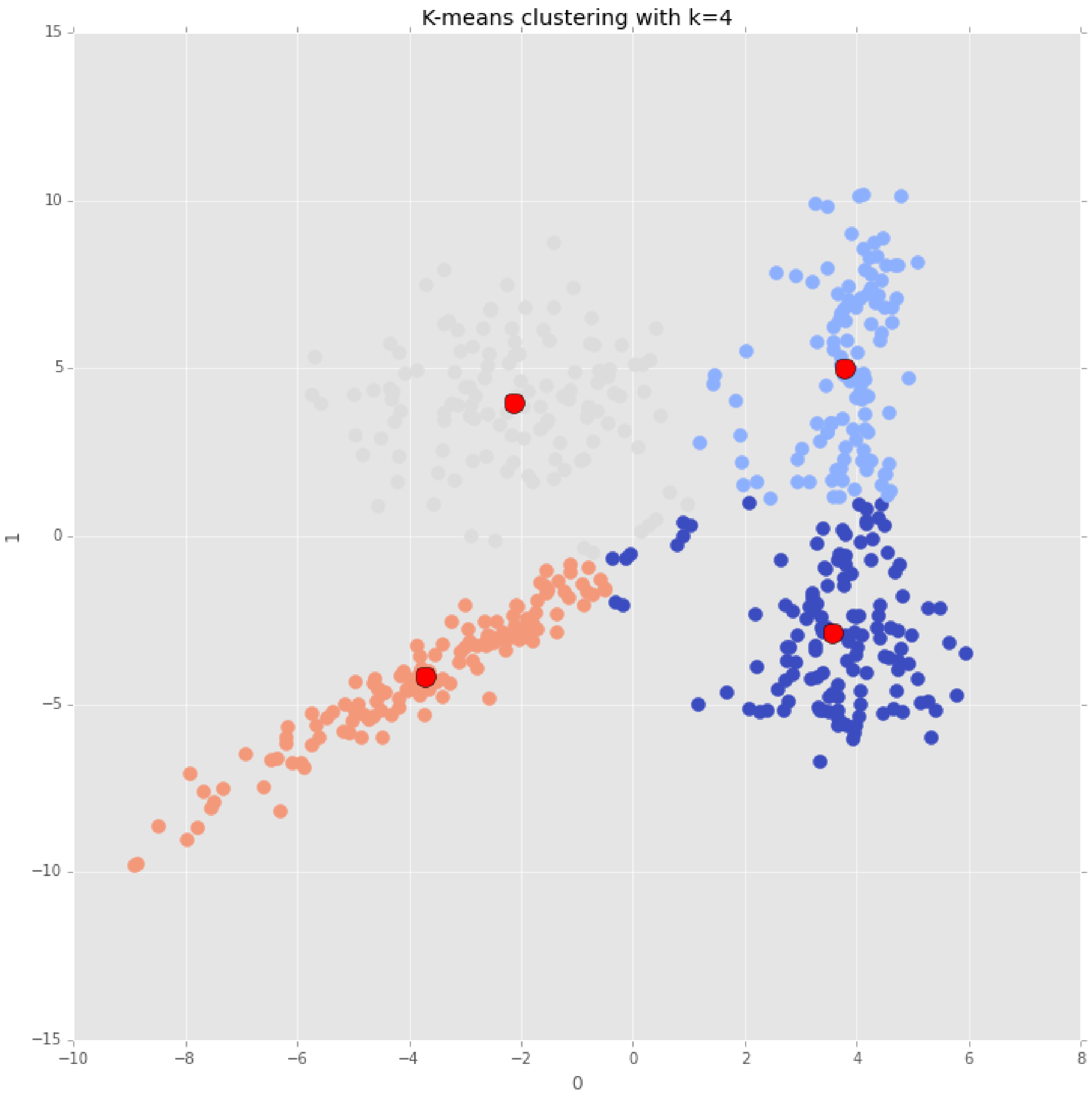
(a). Implementation of kmeans

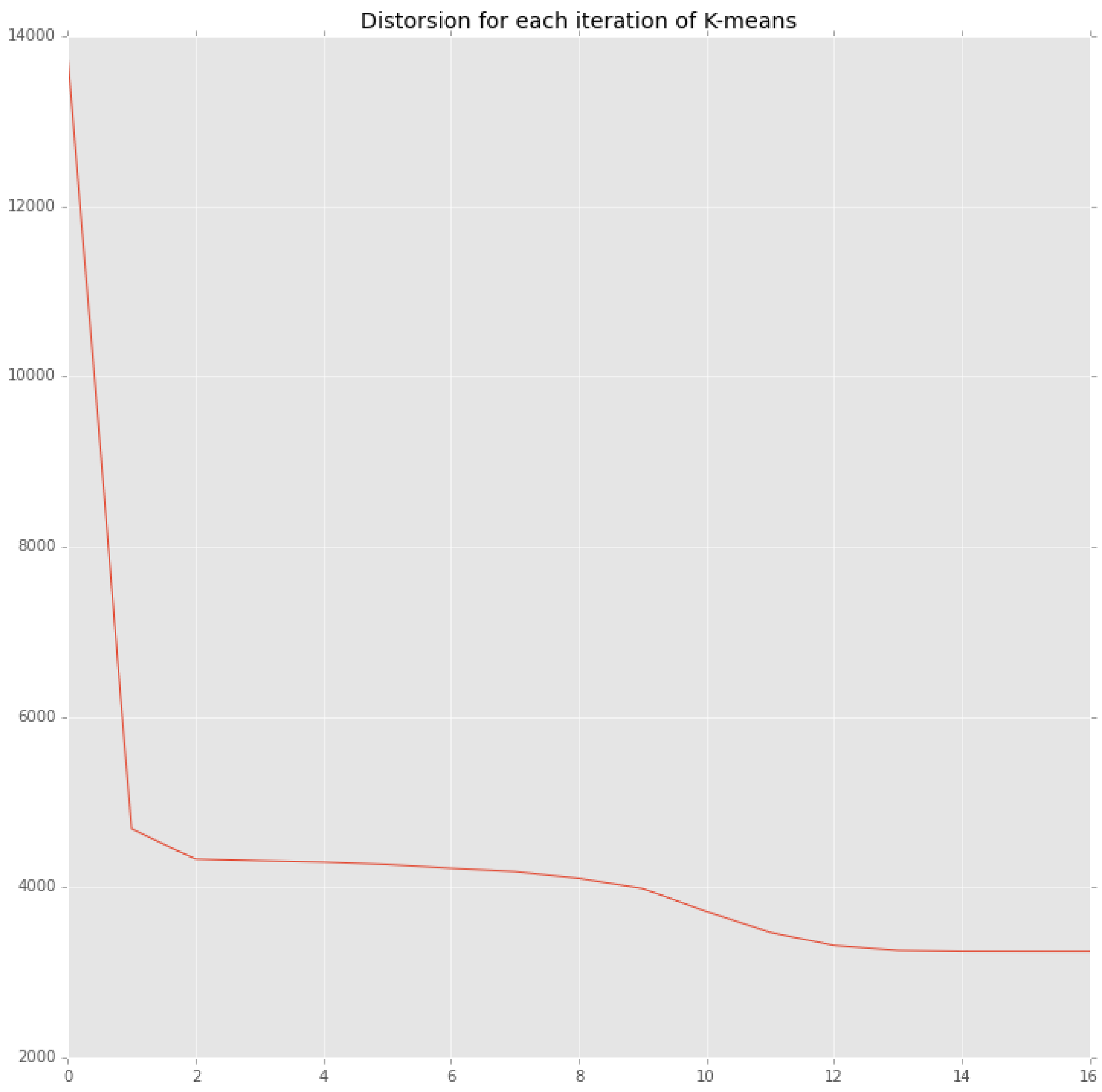
In [2]:
```
def closest_center(point, centers):
    centers = np.asarray(centers)
    dist_2 = np.sum((centers - point)**2, axis=1)
    return np.argmin(dist_2), np.min(dist_2)

def kmeans(df, k):
    n = len(df)
    indexes = random.sample(range(n),k)
    new_centers = np.array([df.iloc[i].values for i in indexes])
    centers = np.array([])
    distorsion = []

    while not np.array_equal(centers, new_centers):
        centers = new_centers

        # Assign each point to the closest center
        closest = np.array([list(closest_center(x.values, centers)) fo
r _, x in df.iterrows()])
        labels = closest[:,0]
        distorsions = closest[:,1]
        distorsion.append(distorsions.sum())

        # Compute new centers
        new_centers = np.array([df[labels == i].mean(axis=0) for i in
range(k)])
    return new_centers, labels, distorsion
```

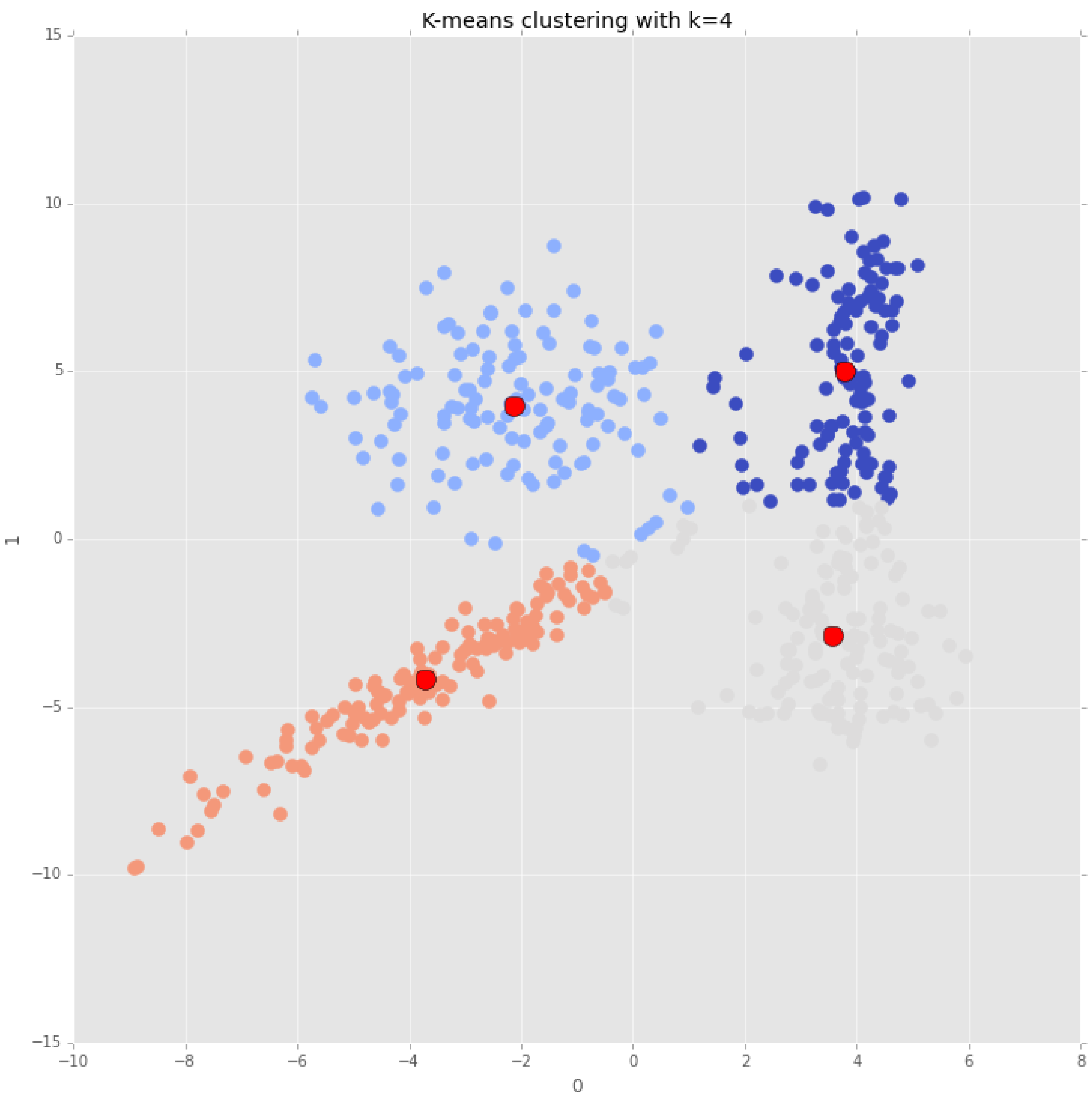
K-means is applied several times with randoms initialisation, the clustering with the center is displayed along with the distorsion at each iteration

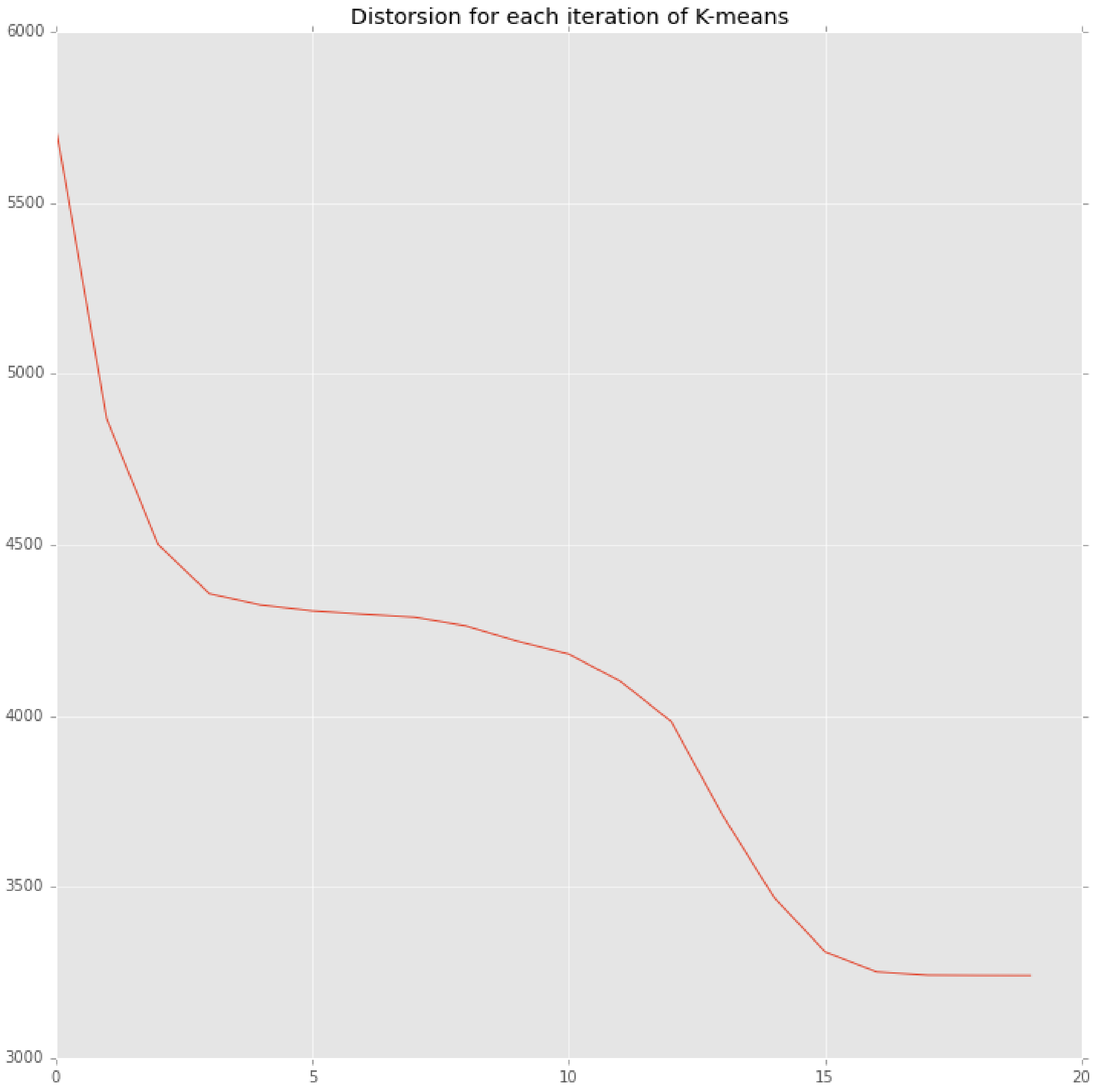
In [3]:
```
k = 4
color_map = cm.get_cmap('coolwarm')
colors = color_map([x/float(k) for x in range(k)])
for i in range(0,3):
    centers, labels, distorsion = kmeans(em_train, k)

    ax = em_train[labels == 0].plot(kind='scatter', x=0, y=1, color=co
lors[0], s=80)
    for j in range(1,k):
        em_train[labels == j].plot(kind='scatter', x=0, y=1, ax=ax, co
lor=colors[j], s=80)
    plt.plot(centers[:,0], centers[:,1], 'o', ms=13, color='red')
    plt.title('K-means clustering with k={}'.format(k))
    plt.show()
    plt.plot(distorsion)
    plt.title('Distorsion for each iteration of K-means')
    plt.show()
```

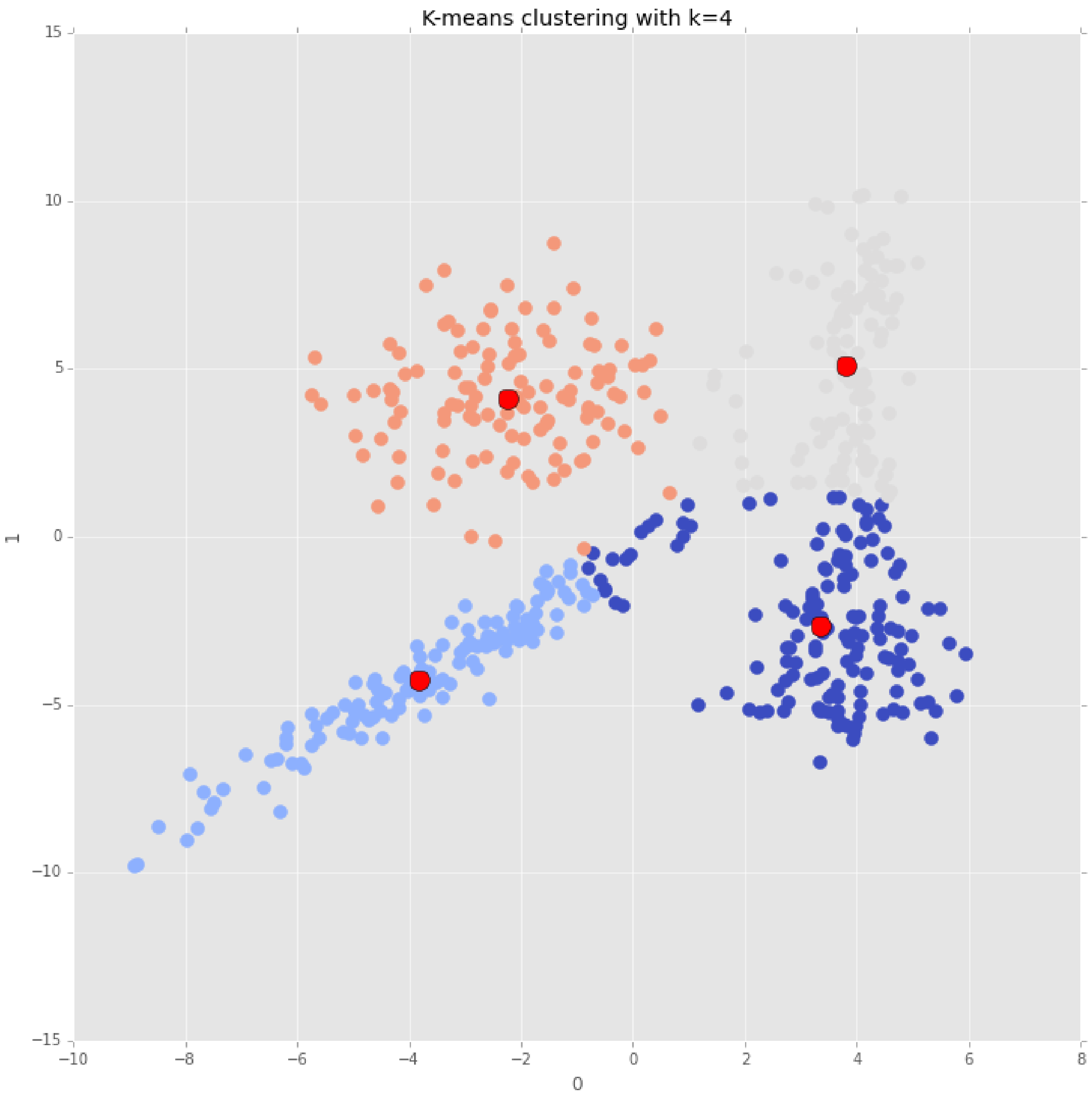
```
/Users/Adrien/anaconda/lib/python2.7/site-packages/matplotlib/collecti
ons.py:571: FutureWarning: elementwise comparison failed; returning sc
alar instead, but in the future will perform elementwise comparison
  if self._edgecolors == str('face'):
```

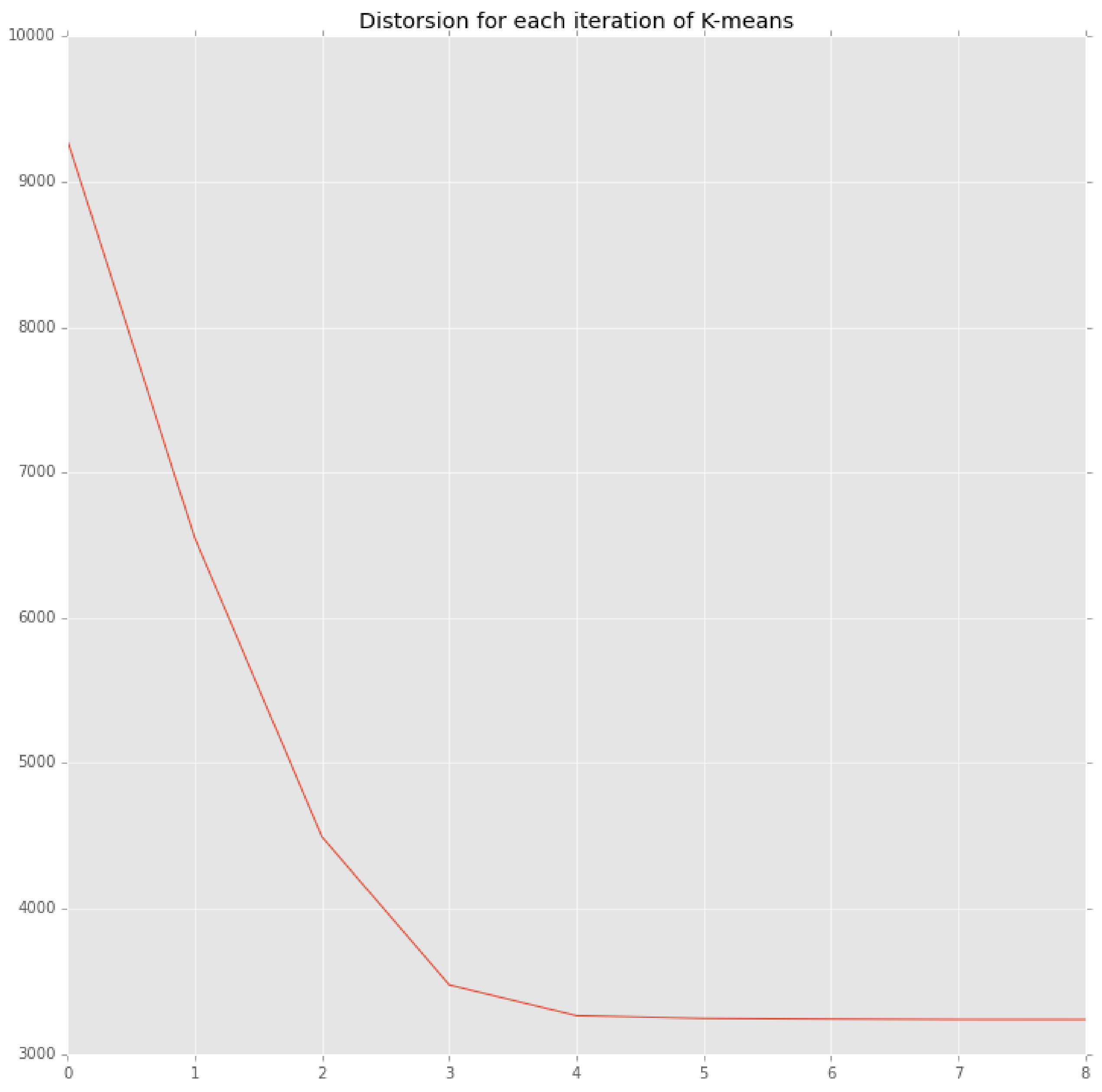



Distorsion for each iteration of K-means
14000
12000
10000
8000
6000
4000
2000
0
2
4
6
8
10
12
14
16


K-means clustering with k=4
15
10
5
0
−5
−10
−15
1
−10
−8
−6
−4
−2
0
2
4
6
8
0


Distorsion for each iteration of K-means
6000
5500
5000
4500
4000
3500
3000
0
5
10
15
20


K-means clustering with k=4
15
10
5
0
−5
−10
−15
1
−10
−8
−6
−4
−2
0
2
4
6
8
0

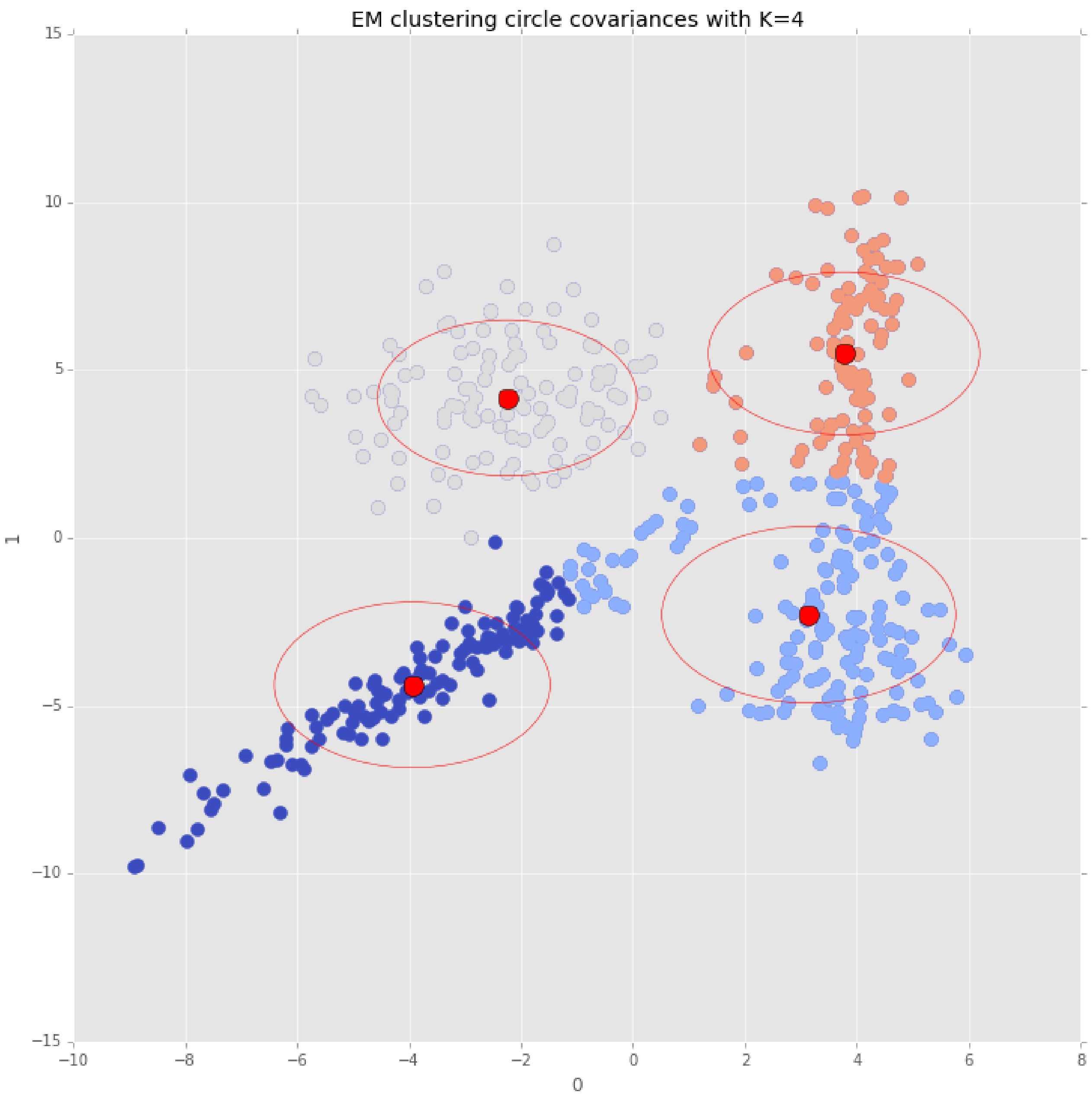


The clustering is efficient with four cluster and doesn't seem to suffer from random initialization

(c). Gaussian mixture

In [4]:
```
# The covariance matrices are proportional the identity matrix
def em_identity(df, K, iter=100):
    array = df.values
    n = len(df)
    p = np.empty(K); p.fill(1/n)
    sigma = [1] * 4

    # Initialize centers with kmeans
    mu, _, _ = kmeans(df, K)
    mu = list(mu)
    res = np.empty((K, n))

    for j in xrange(iter):
        # E step
        for k in range(K):
            res[k,:] = p[k] * multivariate_normal.pdf(array,mu[k],sigm
a[k])
        res = res / res.sum(axis=0)

        # M step
        p = res.mean(axis=1)
        # The responsabilities are normalized to avoid dividing by the
ir sum for the next computations
        res = (res.T / res.sum(axis=1)).T
        mu = np.dot(res,array)
        for k in range(K):
            sigma[k] = res[k,:].dot(np.linalg.norm(array - mu[k],axi
s=1))
    return p, mu, sigma

def predict(df, p, mu, sigma):
    array = df.values
    n = len(df)
    k = len(mu)
    proba = np.zeros((k,n))
    for i in xrange(k):
        proba[i,:] = p[i] * multivariate_normal.pdf(array, mu[i], sigm
a[i])
    return np.argmax(proba, axis=0)
```

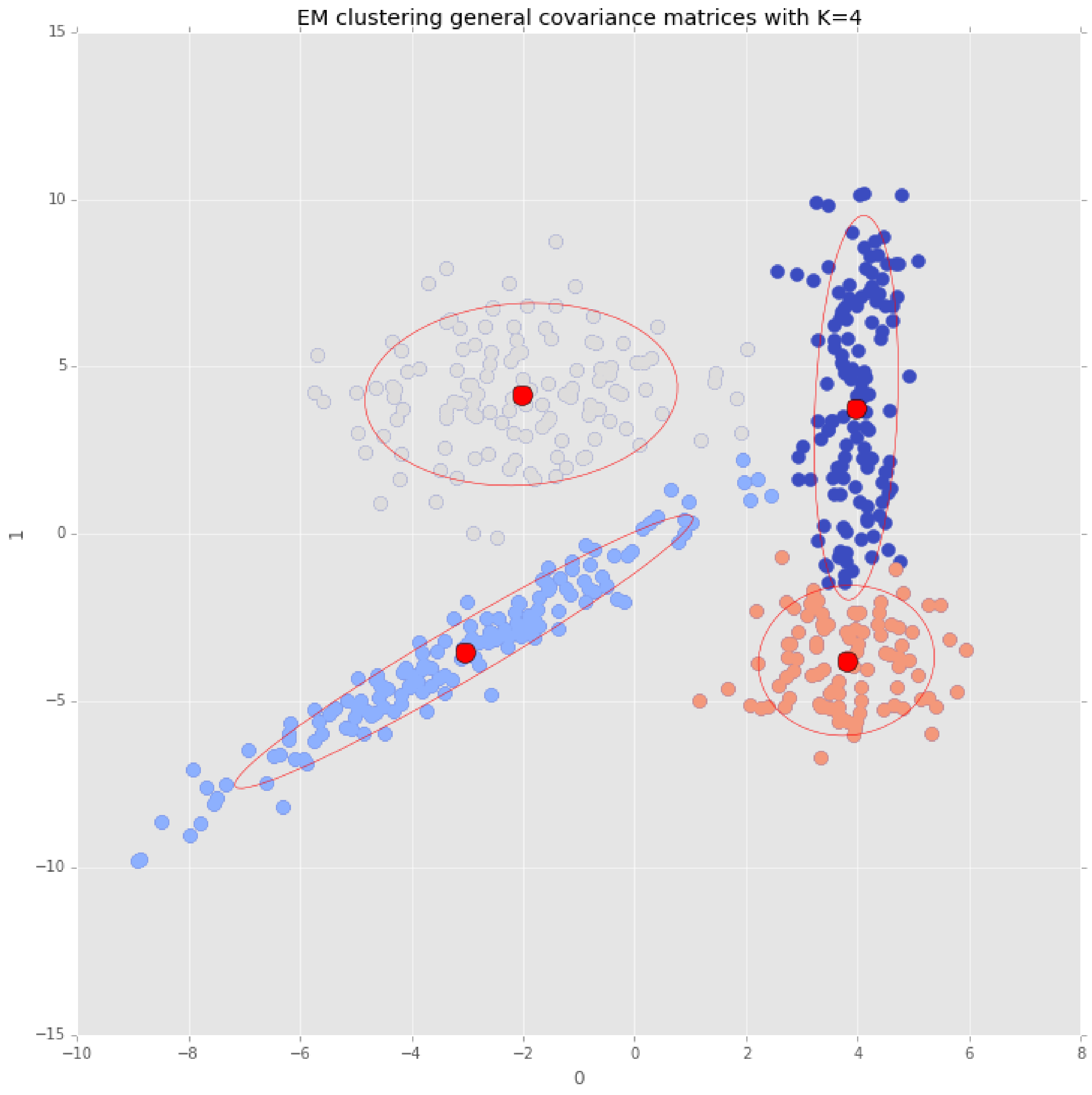
```
In [5]: k = 4
        data = em_train
        color_map = cm.get_cmap('coolwarm')
        colors = color_map([x/float(k) for x in range(k)])

        p1, mu1, sigma1 = em_identity(em_train, k)
        sigma1 = [np.diag(np.repeat(s,2)) for s in sigma1]
        labels = predict(em_train, p1, mu1, sigma1)

        ax = data.plot(kind='scatter', x=0, y=1, color=colors[0], s=80)
        plot_cov_ellipse(sigma1[0], mu1[0], color='red',fill=False)
        for j in range(1,k):
            data[labels == j].plot(kind='scatter', x=0, y=1, ax=ax, color=colo
        rs[j], s=80)
            plot_cov_ellipse(sigma1[j], mu1[j], color='red', fill=False)
        plt.plot(mu1[:,0], mu1[:,1], 'o', ms=13, color='red')
        plt.title('EM clustering circle covariances with K={}'.format(k))
        # plt.gca().set_aspect('equal', adjustable='box') # used to ensure axi
        s have same scale
        plt.show()
```



Here cluster are circles (you can uncomment the last line to have equal axis on the graph), we can observe that it doesn't seem to be right model to be fit to our data

In [6]:
```
def em(df, K, iter=100):
    array = df.values
    n = len(df)
    p = np.empty(K); p.fill(1/n)
    sigma = [df.cov().values] * K

    # Initialize centers with kmeans
    mu, _, _ = kmeans(df, K)
    mu = list(mu)
    res = np.empty((K, n))

    for j in xrange(iter):
        # E step
        for k in range(K):
            res[k,:] = p[k] * multivariate_normal.pdf(array,mu[k],sigm
a[k])
        res = res / res.sum(axis=0)

        # M step
        p = res.mean(axis=1)
        # The responsabilities are normalized to avoid dividing by the
ir sum for the next computations
        res = (res.T / res.sum(axis=1)).T
        mu = np.dot(res,array)
        for k in range(K):
            sigma[k] = (res[k,:] * (array - mu[k]).T).dot(array - m
u[k])
    return p, mu, sigma
```

In [7]:
```
k = 4
data = em_train
color_map = cm.get_cmap('coolwarm')
colors = color_map([x/float(k) for x in range(k)])

p2, mu2, sigma2 = em(em_train, k)
labels = predict(em_train, p2, mu2, sigma2)

ax = data.plot(kind='scatter', x=0, y=1, color=colors[0], s=80)
plot_cov_ellipse(sigma2[0], mu2[0], color='red',fill=False)
for j in range(1,k):
    data[labels == j].plot(kind='scatter', x=0, y=1, ax=ax, color=colo
rs[j], s=80)
    plot_cov_ellipse(sigma2[j], mu2[j], color='red', fill=False)
plt.plot(mu2[:,0], mu2[:,1], 'o', ms=13, color='red')
plt.title('EM clustering general covariance matrices with K={}'.forma
t(k))
plt.show()
```



This time with general covariance matrix the model fit better the different shapes of our data (d). Loglikelihood of the model on train data and test on test data

As expected from the plot of the data and the shape of the clusters the second model with general covariance matrices is performing better for clustering.

In [8]:
```
def loglikelihood(df, mu, sigma, p):
    K = len(p)
    ll = np.zeros((len(df), K))
    array = df.values
    for k in xrange(0,K):
        ll[:,k] = multivariate_normal.pdf(data, mu[k,:], sigma[k])
    ll = np.dot(ll,p)
    return np.log(ll).sum()
```

In [11]:
```
print "Log-likelihood with model 1 (circle covariance matrices) on tra
in data: {}".format(loglikelihood(em_train, mu1, sigma1,p1))
print "Log-likelihood with model 2 (general covariance matrices) on tr
ain data: {}\n".format(loglikelihood(em_train, mu2, sigma2,p2))
print "Log-likelihood with model 1 on test data: {}".format(loglikelih
ood(em_test, mu1, sigma1,p1))
print "Log-likelihood with model 2 on test data: {}".format(loglikelih
ood(em_test, mu2, sigma2,p2))
```

```
Log-likelihood with model 1 (circle covariance matrices) on train dat
a: -2703.77401346
Log-likelihood with model 2 (general covariance matrices) on train dat
a: -2327.71567492

Log-likelihood with model 1 on test data: -2703.77401346
Log-likelihood with model 2 on test data: -2327.71567492
```

As expected the Log-likelihod is higher with the second model