```
#generate sample data
#From last assignment

from sklearn.datasets import make_moons
from random import uniform

def moons(samples = 1000, d=0, r=10, w=6, n=None):

    points, moon = make_moons(n_samples=samples, shuffle=False, noise=n, random_state=
        None)

    #scale with r
    for p in points:
        p[0] = p[0] *r
        p[1] = p[1] *r


    #move: with d
    for i in range(len(points)):
        if i < len(points)/2:
            points[i][1] = points[i][1]+0.5*w
        else:
            points[i][1] = points[i][1]-d-w-0.5*w


    #widen with w
    for p in points:
        p[0] = p[0] + uniform(-w/2, w/2)
        p[1] = p[1] + uniform(-w/2, w/2)

    return points, moon


def case(d, n):

    print 'd = ', d
    print 'noise = ', n

    train, m = moons(1000, d, 10, 6, n)
    scatter([p[0] for p in train], [p[1] for p in train], c = m)
    show()

    test, _ = moons(3000, d, 10, 6, n)

    rbf = RBF(2, 2, 2)
    #clf.fit(train, m)
    #m = clf.predict(test)

    scatter([p[0] for p in test], [p[1] for p in test], c = m)
    show()
```

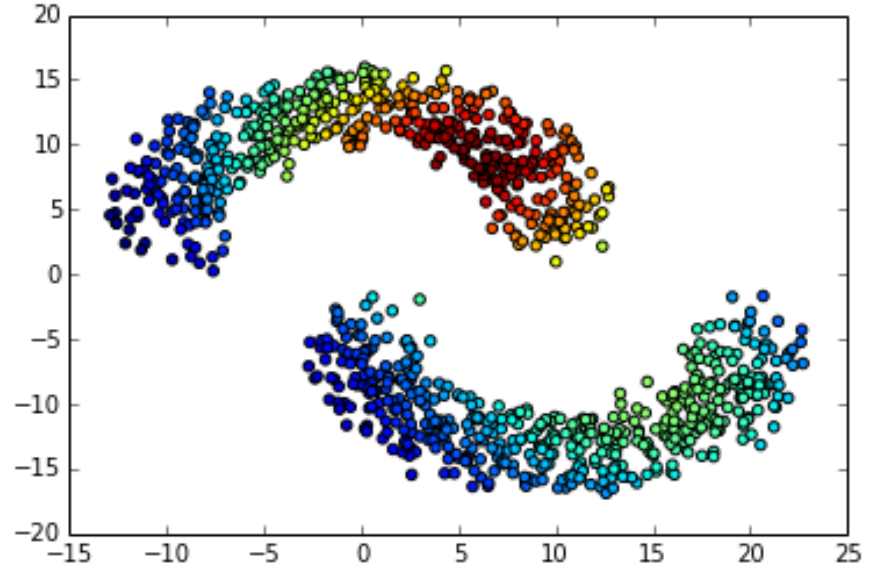
```
from scipy.cluster.vq import kmeans2
from math import sqrt, exp
from numpy.linalg import norm
from itertools import combinations

class RBF:

    def __init__(self, input_layers, hidden_layers, output_layers):
        self.input_layers = input_layers
        self.hidden_layers = hidden_layers
        self.output_layers = output_layers
        self.hidden_means = []
        self.classificatin= []
        self.ws=[]


    def train(self, data, reg_parm):

        #K-MEANS
        self.hidden_means, self.classification = kmeans2(data, self.hidden_layers)
        self.ws = [random.random() for _ in range(len(self.hidden_means)+1)]

        #Define Gaussian functions
        hidden_gaussians = []

        combs = [p for p in combinations(self.hidden_means, 2)]
        dmax = max(map(norm, combs))

        K = len(self.hidden_means)
        sigma = dmax/(sqrt(2*K))

        for p in self.hidden_means:
            #xo = p[] BECAUSE: CLOSURE
            def gauss(x, y, xo = p[0], yo = p[1]):
                return exp(-(
                        ((x-xo)**2)/(2*(sigma**2))
                            +
                            ((y-yo)**2)/(2*(sigma**2))
                            )
                        )
            hidden_gaussians.append(gauss)
        self.hidden_gaussians = hidden_gaussians



        #train perceptron
        unit_step = lambda x: 0 if x < 0 else 1


        for i in xrange(len(data)):
```

```
            x = []
            for g in self.hidden_gaussians:
                eta = 0.5*(norm(reg_parm *g(data[i][0], data[i][1])))
                x.append(eta)
            #apply gaussians to input
            #self.data[i]
            x = append(x, 1)
            expected = self.classification[i]
            result = dot(self.ws, x)
            error = expected - unit_step(result)
            self.ws += 0.1 * error * x

    def test(self, p):
        x = p[0]
        y = p[1]

        s = []
        #apply gaussians
        for g in self.hidden_gaussians:
            s.append(g(x, y))

        #apply weights
        s = dot(self.ws, (append(s, 1)))
        return s
```

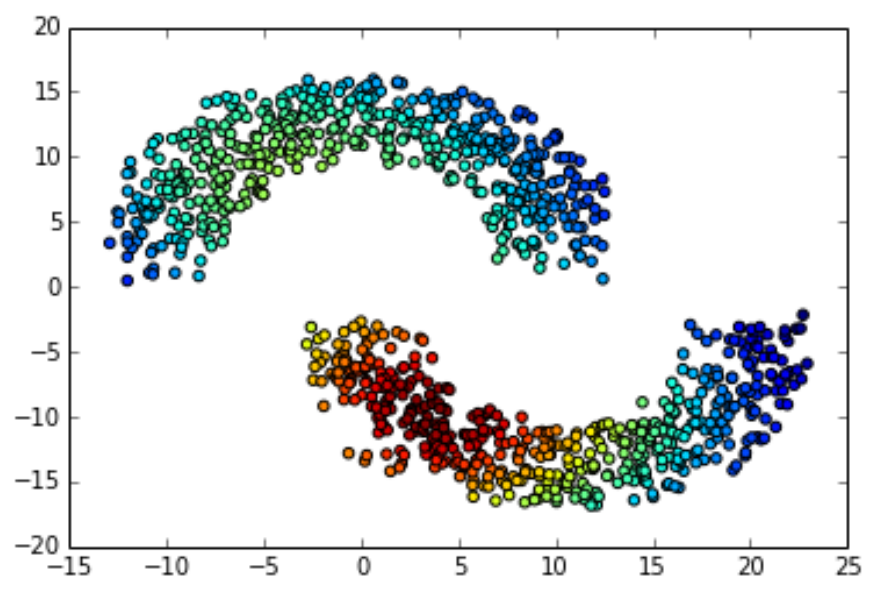
```
#define case
def case(dist, dim, reg_parm):
    train, _ = moons(1000, dist, 10, 6, None)
    test, _ = moons(1000, dist, 10, 6, None)
    rbf = RBF(2, dim, 2)
    rbf.train(train, reg_parm)

    m = []
    for t in test:
        m.append(rbf.test(t))

    scatter([p[0] for p in test], [p[1] for p in test], c = m)
    show()
```

```
for i in arange(0, 1, 0.2):
    print "lambda = ", i
    case(0.0, 4, i)
    #case(-5.0, 4, i)
```

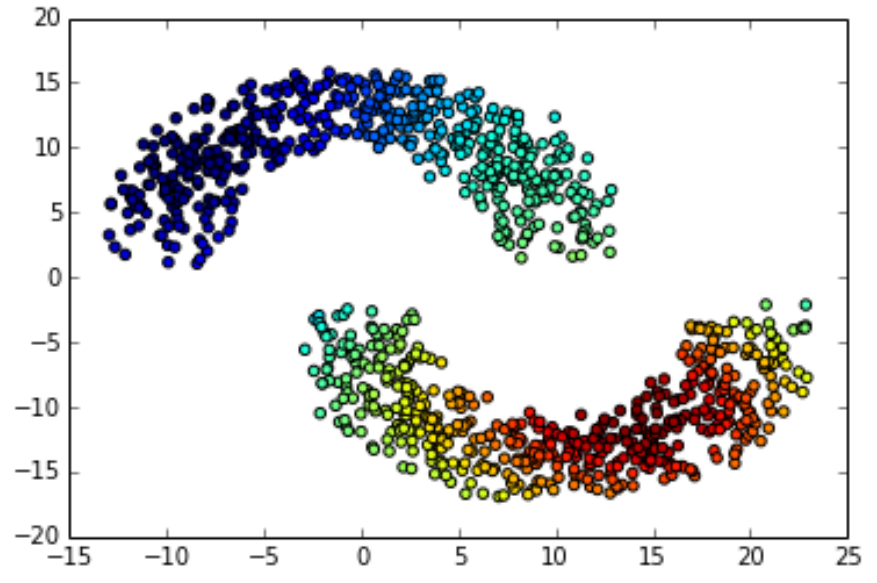
```
lambda =  0.0
```



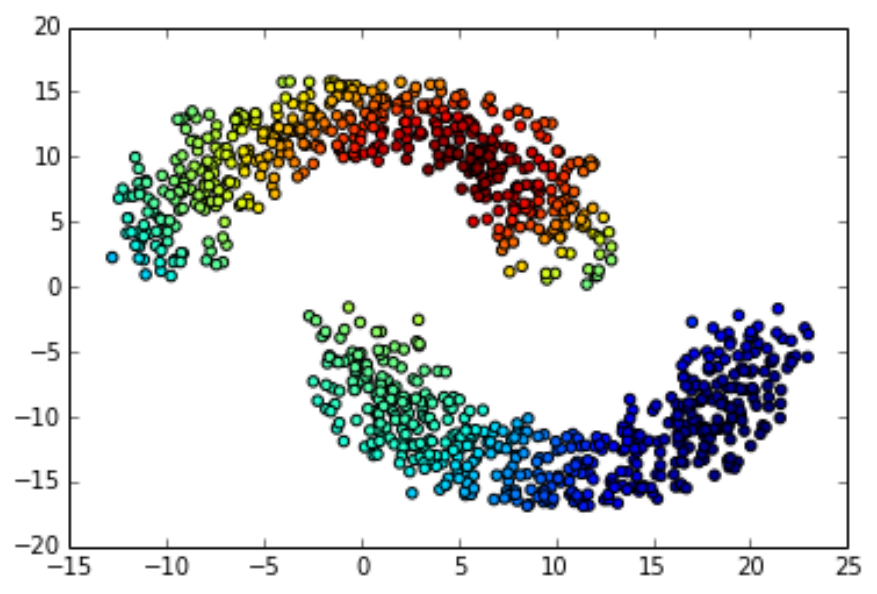

```
lambda =  0.2
```



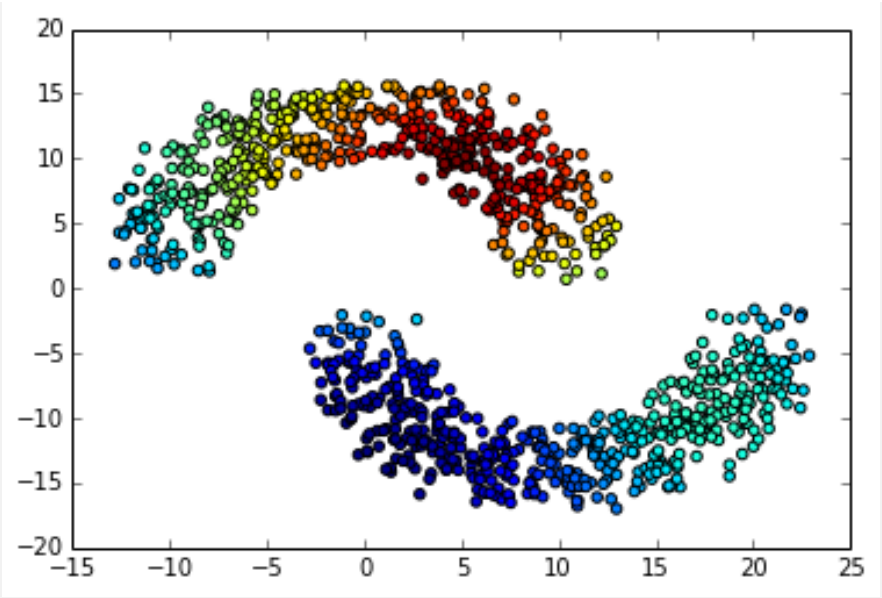

```
lambda =  0.4
```




```
lambda =  0.6
```

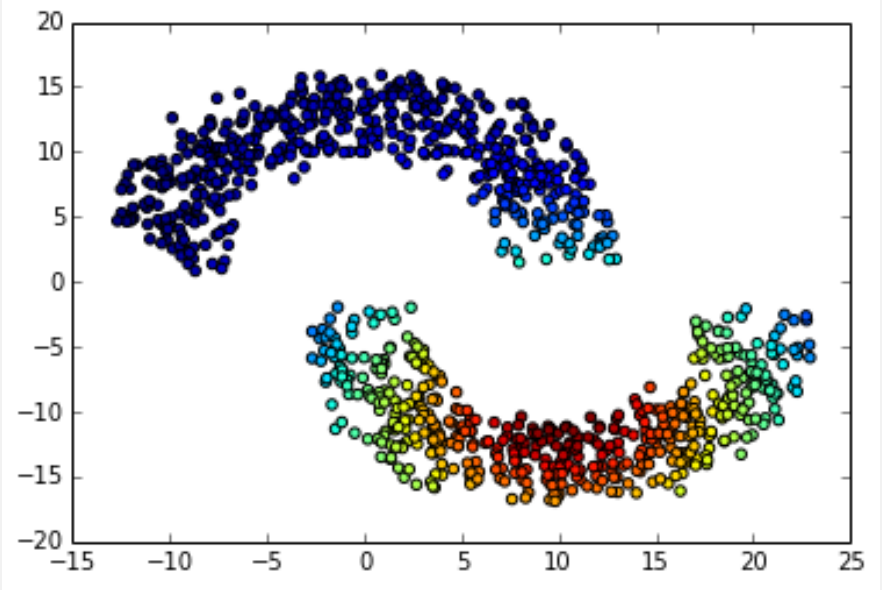


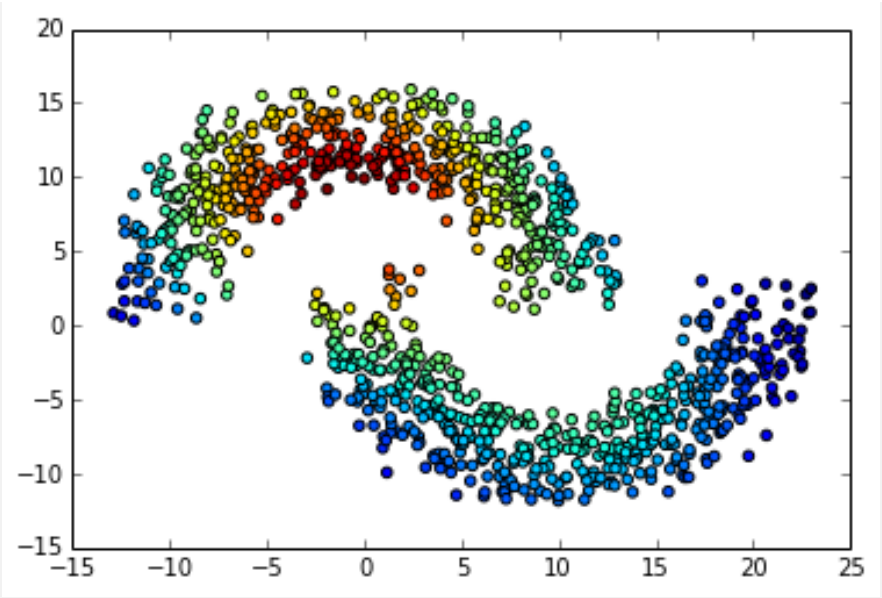

```
lambda =  0.8
```



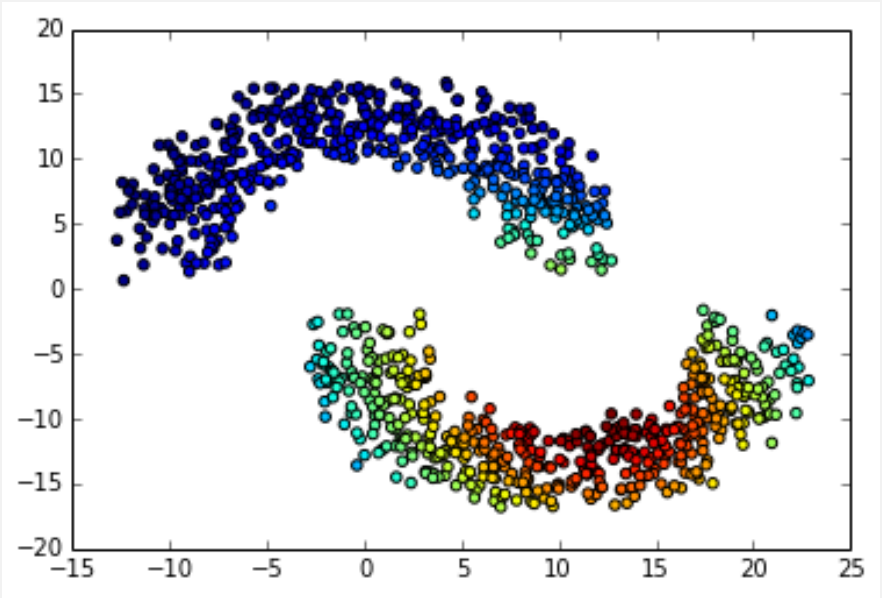

```
for i in range(6):
    print "K = ", i+2
    case(0.0, i+2, 0.2)
    case(-5.0, i+2, 0.2)
```

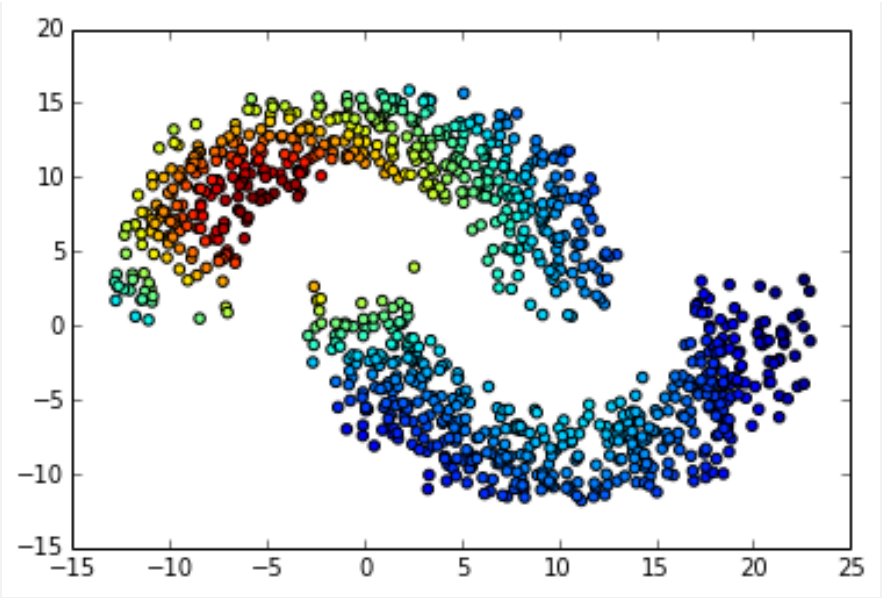
```
K =  2
```



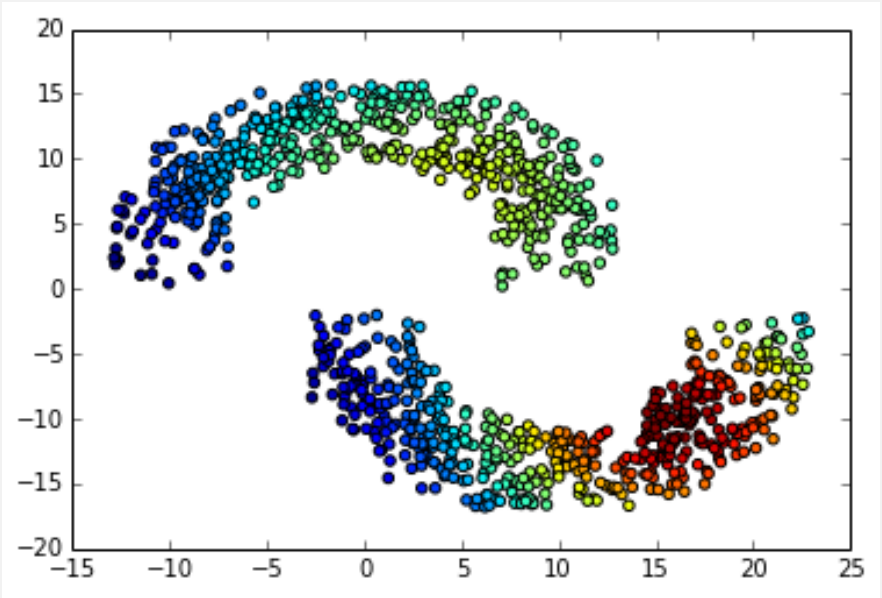

20
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15
20
25

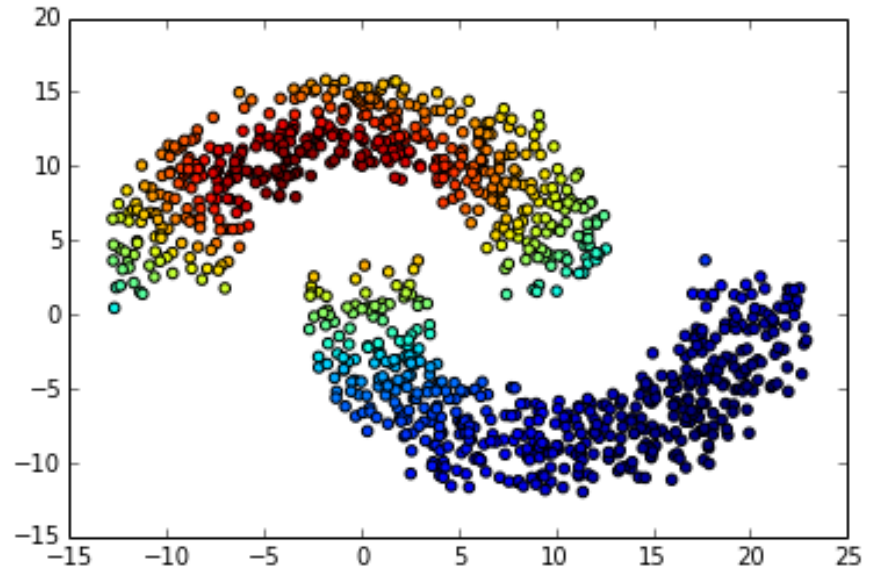
```
K =  3
```


20
15
10
5
0
−5
−10
−15
−20
−15
−10
−5
0
5
10
15
20
25

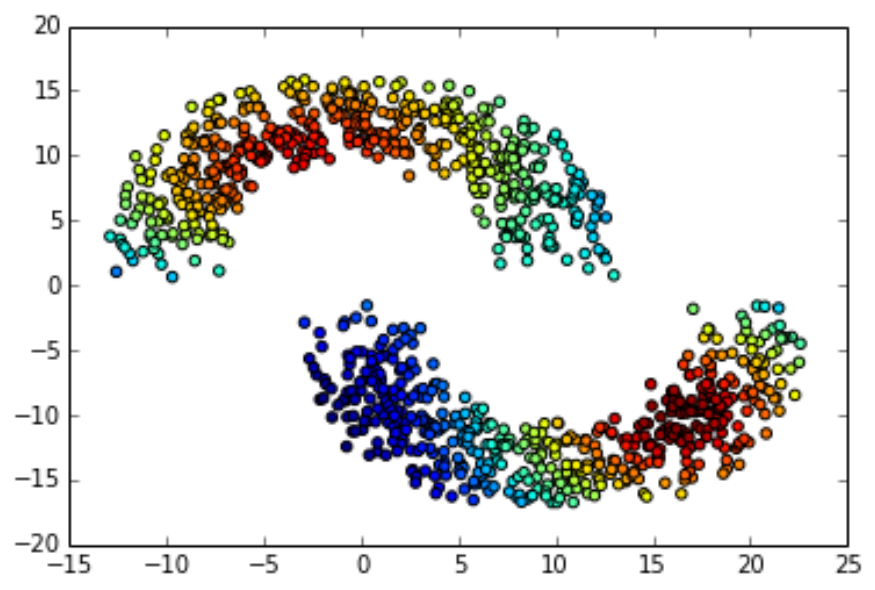

20
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15
20
25

K = 4

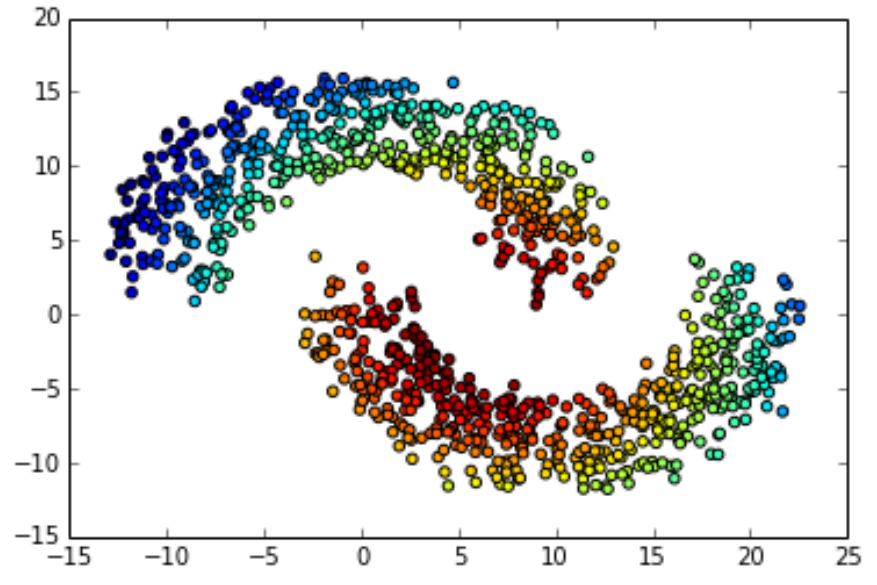

20
15
10
5
0
−5
−10
−15
−20
−15
−10
−5
0
5
10
15
20
25



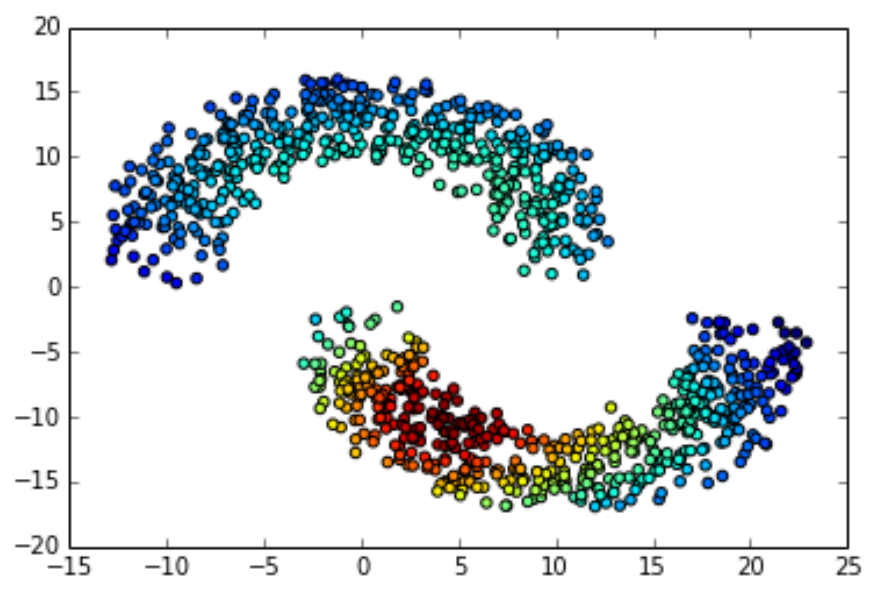

```
K =  5
```

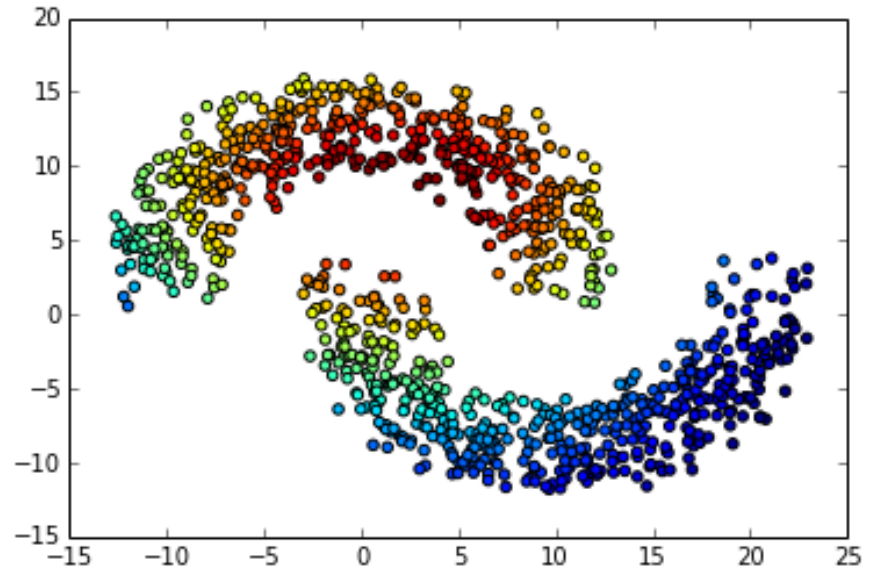



```
/home/quignon/anaconda/lib/python2.7/site-packages/scipy/cluster/vq.py:588: UserWarning: One of
  warnings.warn("One of the clusters is empty. "
```

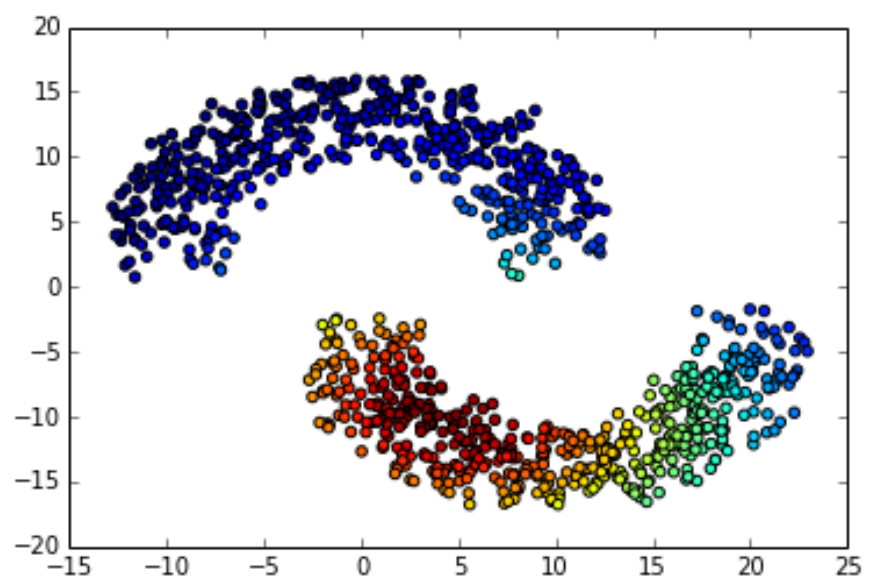

20
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15
20
25

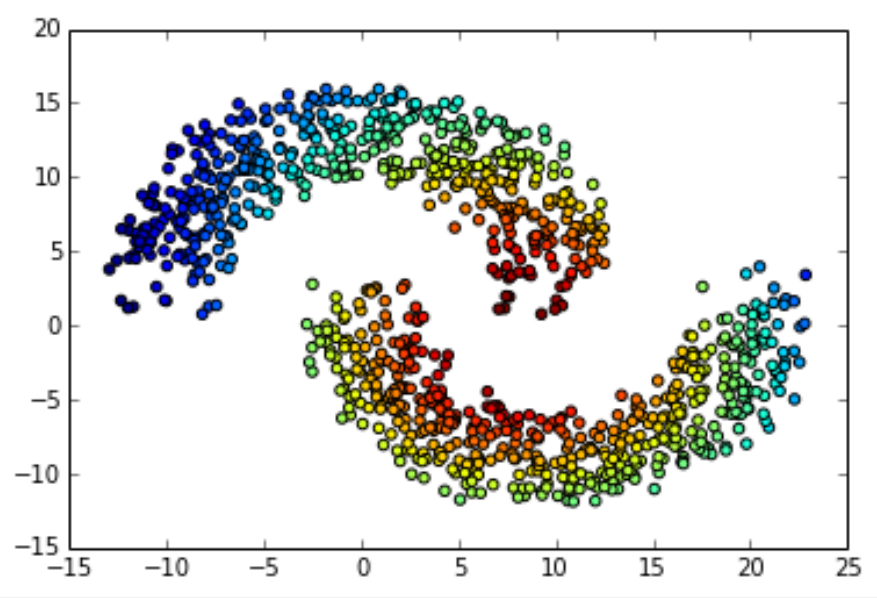
```
K =  6
```


20
15
10
5
0
−5
−10
−15
−20
−15
−10
−5
0
5
10
15
20
25

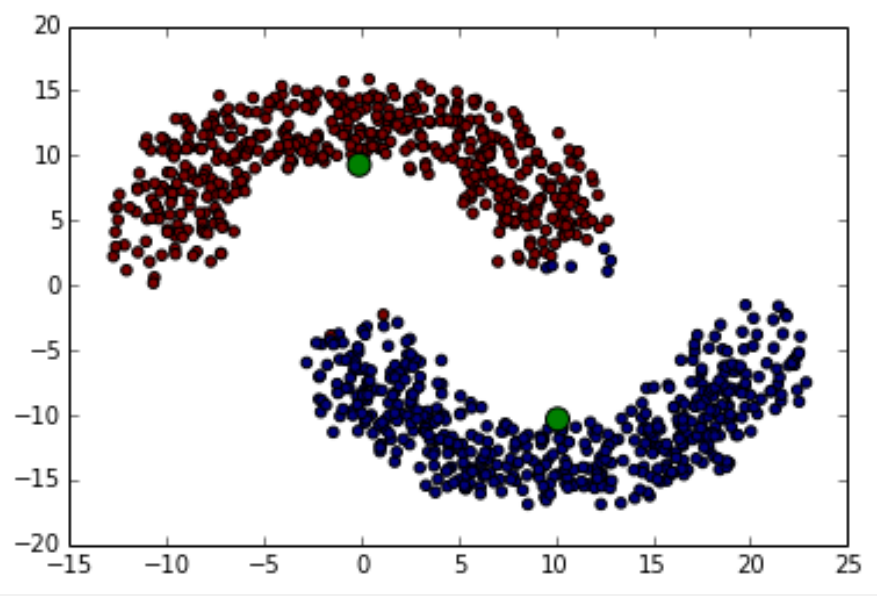

20
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15
20
25

```
K =  7
```


20
15
10
5
0
−5
−10
−15
−20
−15
−10
−5
0
5
10
15
20
25




```
#K-Mean
train, m = moons(1000, 0.0, 10, 6, None)

means, classification = kmeans2(train, 2)

scatter(train[:,0], train[:,1], c=classification)
scatter(means[:,0], means[:,1], c='green', s = 100)
show()
```



```
#K-Mean
train, m = moons(1000, -5.0, 10, 6, None)

means, classification = kmeans2(train, 2)

scatter(train[:,0], train[:,1], c=classification)
scatter(means[:,0], means[:,1], c='green', s = 100)
show()
```