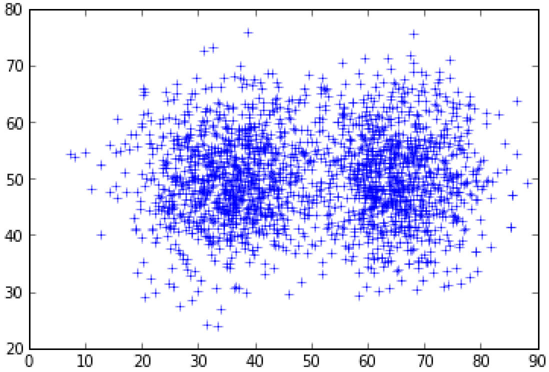
```python
from scipy.spatial.distance import cdist
from numpy import sort as asort
from collections import Counter
```

```
data = r_[8*randn(1000,2)+array([65,50]),
          8*randn(1000,2)+array([35,50])]
classes = array([0]*1000+[1]*1000)
plot(data[:,0],data[:,1],'+')
```



```
test_data = r_[8*randn(1000,2)+array([65,50]),
              8*randn(1000,2)+array([35,50])]
test_classes = array([0]*1000+[1]*1000)
```

k-Nearest Neighbor Prediction / Error

```python
def knnerr(data,classes,test_data,test_classes,k=1):
    dists = cdist(test_data,data)
    nearest = argsort(dists,axis=0)[:,:k]
    nclasses = take(classes,nearest,axis=0)
    pred = array([Counter(c).most_common(3)[0][0] for c in nclasses
        ])
    return sum(pred!=test_classes)*1.0/len(test_classes)

for k in range(1,21):
    print k,knnerr(data,classes,test_data,test_classes,k=k)
```

```
1 0.063
2 0.0465
3 0.039
4 0.053
5 0.031
...
17 0.021
18 0.0165
19 0.014
20 0.013
```

Density Estimation and k-Nearest Neighbor

Nearest-neighbor methods are related to non-parametric density estimators.

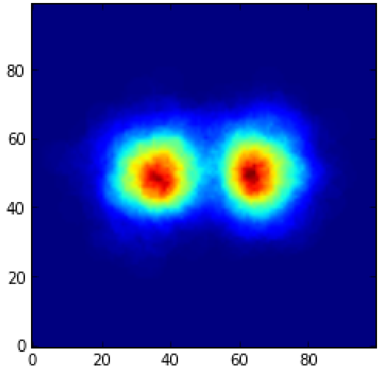
The simplest non-parametric density estimator is to count the number of samples in a volume and divide by the volume (or not divide if we don't care about normalization).

```
def rcounts(test_data,data,r):
    dists = cdist(test_data,data)
    counts = sum(dists<r,axis=1)
    return counts
```

Here is a non-parametric density estimate.

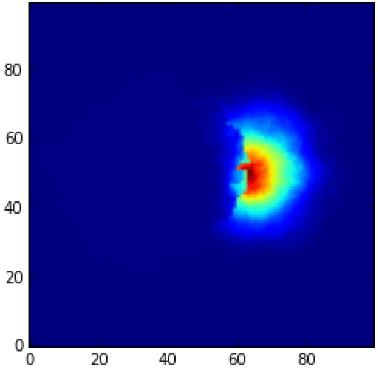
```python
xs,ys = mgrid[:100,:100]
coords = c_[xs.ravel(),ys.ravel()]
counts = rcounts(coords,data,5.0).reshape(100,100)
imshow(counts.T,origin='lower')
```

```
(10000, 2)
```



For classification, we need to estimate $P(c|x)/P(x)$. We can use non-parametric estimates for both of these.

```
posterior = (rcounts(coords,data,5.0)/maximum(1,rcounts(coords,data
    [classes==1],5.0))).reshape(100,100)
imshow(posterior.T,origin='lower')
```

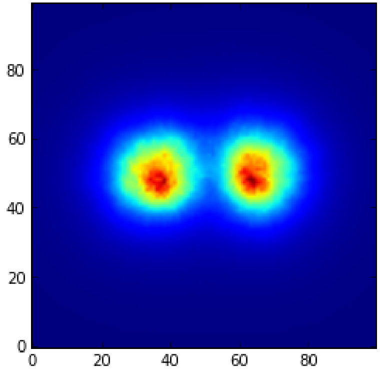


The problem with estimating densities by balls is that we don't necessarily know the scale of the data. For density estimation, is often easier to keep the number of neighbors we look for constant and let the volume vary.

```python
def kdists(test_data,data,k=1):
    dists = cdist(test_data,data)
    d = asort(dists,axis=1)[:,k]
    return d
def kvols(test_data,data,k=1):
    assert data.shape[1]==2
    return kdists(test_data,data,k=k)**2*pi
```

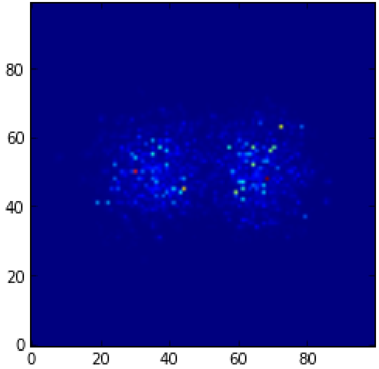
This also gives us a good non-parametric density estimate.

```
vols = kvols(coords,data,k=100).reshape(100,100)
imshow(1.0/vols.T,origin='lower')
```



For 1-nearest neighbor, the result looks a little less convincing, but the estimates still work well for classification.

```
vols = kvols(coords,data,k=1).reshape(100,100)
imshow(1.0/vols.T,origin='lower')
```



k-nearest neighbor classification

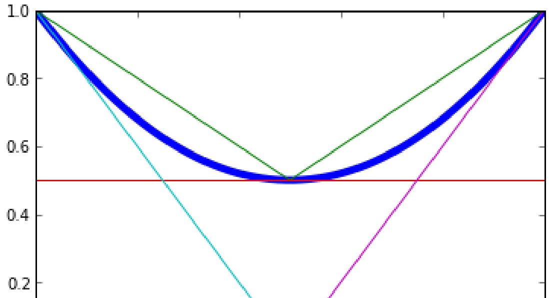
- adaptively pick a volume by picking the k -th nearest neighbor
- within that volume, estimate the posterior probability by counting the number of samples in each class and dividing by the total number of samples (i.e., k)

1-nearest neighbor classification:

- works just like k -NN, but worth thinking about it differently too
- two class case
- posterior probability is $P(c|x)$
- unknown sample x' , nearest training set sample x
- for large enough test set $P(c'|x') \approx P(c|x)$
- $P(c = c') = P(c|x)^2 + (1 - P(c|x))^2 = 2P(c|x)^2 - 2P(c|x) + 1$

The above equation gives us a bound on the nearest neighbor classification error in terms of the Bayes error. We can derive this bound mathematically, or simply look at a graph. For *k* -nearest neighbors, there are better bounds. This is much better than what you might suppose given the noisy estimate from the nonparametric density estimator.

```
p = linspace(0,1,1000)
ylim((0,1))
plot(p,2*p**2-2*p+1,linewidth=5)
plot(p,maximum(p,1-p)); plot(p,0.5*ones(len(p))); plot(p,1-2*p);
    plot(p,-1+2*p)
```


1.0
0.8
0.6
0.4
0.2