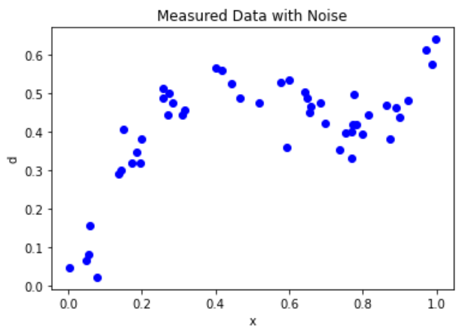
1a)

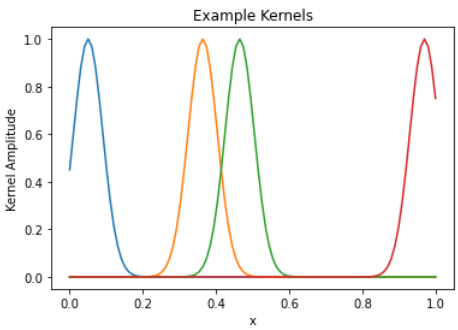
lamda = 0.01, sigma = 0.04

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(1024)  # ensure same noise for each run

# number of training points
n = 50

# sample n random points between 0 and 1
x = np.random.rand(n,1)

# set d = x^2 + .4 sin(1.5 pi x) + noise
d = x*x + 0.4*np.sin(1.5*np.pi*x) +0.04*np.random.randn(n,1)

# plot result
plt.plot(x,d,'bo')
plt.xlabel('x')
plt.ylabel('d')
plt.title('Measured Data with Noise')
plt.show()
```



In [2]:
```
sigma = 0.04 #defines Gaussian kernel width
p = 100 #number of points on x-axis

# Display examples of the kernels
x_test = np.linspace(0,1.00,p)  # uniformly sample interval [0,1]
j_list = [5, 36, 46, 96]  #list of indices for example kernels

Kdisplay = np.zeros((p,len(j_list)),dtype=float)

for i in range(p):
    for j in range(len(j_list)):
        Kdisplay[i,j]= np.exp(-(x_test[i]-x_test[j_list[j]])**2/(2*sigma**2))

print('Sigma = ',sigma)
plt.plot(x_test, Kdisplay)
plt.title('Example Kernels')
plt.xlabel('x')
plt.ylabel('Kernel Amplitude')
plt.show()
```

```
Sigma =  0.04
```

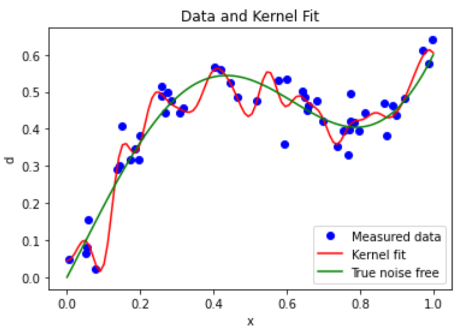


In [3]:
```
# Kernel fitting to data

lam = 0.01 #ridge regression parameter

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        distsq[i,j]=(x[i]-x[j])**2

K = np.exp(-distsq/(2*sigma**2))

alpha = np.linalg.inv(K+lam*np.identity(n))@d
```

In [4]:

```python
# Generate smooth curve corresponding to data fit

distsq_xtest = np.zeros((p,n),dtype=float)
for i in range(0,p):
    for j in range(0,n):
        distsq_xtest[i,j] = (x_test[i]-x[j])**2

dtest = np.exp(-distsq_xtest/(2*sigma**2))@alpha

dtrue = x_test*x_test + 0.4*np.sin(1.5*np.pi*x_test)  # noise free data for comparison

print('Sigma = ',sigma)
print('Lambda = ',lam)
plt.plot(x,d,'bo',label='Measured data')
plt.plot(x_test,dtest,'r',label='Kernel fit')
plt.plot(x_test,dtrue,'g',label='True noise free')
plt.title('Data and Kernel Fit')
plt.legend(loc='lower right')
plt.xlabel('x')
plt.ylabel('d')
plt.show()
```

```
Sigma =  0.04
Lambda =  0.01
```



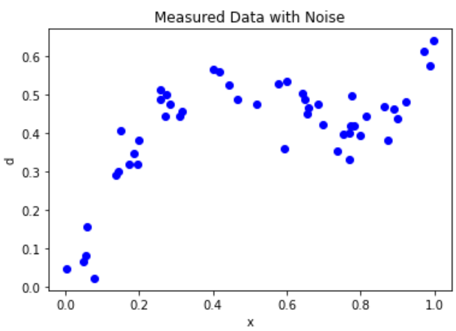
The third peak from the left has an xi value of roughly 0.45. The peak is located at the value of xi and the curve drops down on either side this xi value. The sigma determines the smoothness of the curve. The higher the sigma the smoother the cuve is i.e. the less the curve factors infiner details about the dataset in question. Conversely, the smaller sigma is, the more the curve tends to overfit the data.

1b)

lamda = 0.01, sigma = 0.2

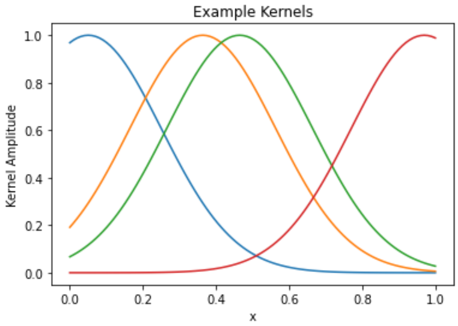
In [5]:

```python
np.random.seed(1024)  # ensure same noise for each run

# number of training points
n = 50

# sample n random points between 0 and 1
x = np.random.rand(n,1)

# set d = x^2 + .4 sin(1.5 pi x) + noise
d = x*x + 0.4*np.sin(1.5*np.pi*x) +0.04*np.random.randn(n,1)

# plot result
plt.plot(x,d,'bo')
plt.xlabel('x')
plt.ylabel('d')
plt.title('Measured Data with Noise')
plt.show()
```



In [6]:
```
sigma = 0.2 #defines Gaussian kernel width
p = 100 #number of points on x-axis

# Display examples of the kernels
x_test = np.linspace(0,1.00,p)  # uniformly sample interval [0,1]
j_list = [5, 36, 46, 96]  #list of indices for example kernels

Kdisplay = np.zeros((p,len(j_list)),dtype=float)

for i in range(p):
    for j in range(len(j_list)):
        Kdisplay[i,j]= np.exp(-(x_test[i]-x_test[j_list[j]])**2/(2*sigma**2))

print('Sigma = ',sigma)
plt.plot(x_test, Kdisplay)
plt.title('Example Kernels')
plt.xlabel('x')
plt.ylabel('Kernel Amplitude')
plt.show()
```

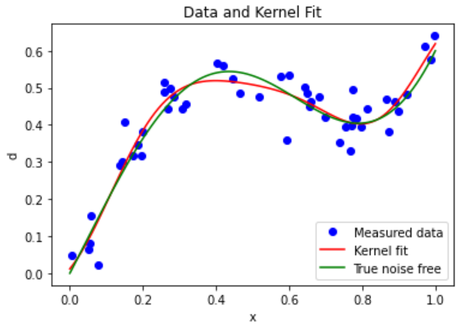
```
Sigma =  0.2
```



In [7]:
```
# Kernel fitting to data

lam = 0.01 #ridge regression parameter

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        distsq[i,j]=(x[i]-x[j])**2

K = np.exp(-distsq/(2*sigma**2))

alpha = np.linalg.inv(K+lam*np.identity(n))@d
```

In [8]:
```
# Generate smooth curve corresponding to data fit

distsq_xtest = np.zeros((p,n),dtype=float)
for i in range(0,p):
    for j in range(0,n):
        distsq_xtest[i,j] = (x_test[i]-x[j])**2

dtest = np.exp(-distsq_xtest/(2*sigma**2))@alpha

dtrue = x_test*x_test + 0.4*np.sin(1.5*np.pi*x_test)  # noise free data for comparison

print('Sigma = ',sigma)
print('Lambda = ',lam)
plt.plot(x,d,'bo',label='Measured data')
plt.plot(x_test,dtest,'r',label='Kernel fit')
plt.plot(x_test,dtrue,'g',label='True noise free')
plt.title('Data and Kernel Fit')
plt.legend(loc='lower right')
plt.xlabel('x')
plt.ylabel('d')
plt.show()
```

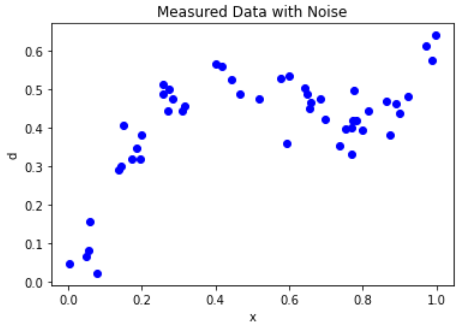
```
Sigma =  0.2
Lambda =  0.01
```



In [ ]:

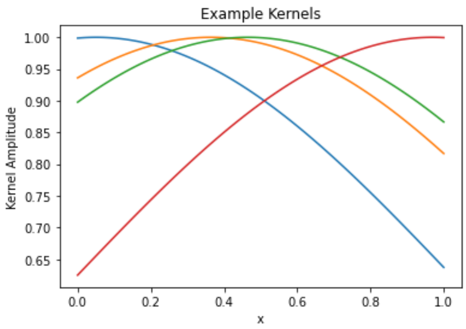
lamda = 0.01, sigma = 1

In [9]:
```
np.random.seed(1024)  # ensure same noise for each run

# number of training points
n = 50

# sample n random points between 0 and 1
x = np.random.rand(n,1)

# set d = x^2 + .4 sin(1.5 pi x) + noise
d = x*x + 0.4*np.sin(1.5*np.pi*x) +0.04*np.random.randn(n,1)

# plot result
plt.plot(x,d,'bo')
plt.xlabel('x')
plt.ylabel('d')
plt.title('Measured Data with Noise')
plt.show()
```



In [10]:
```
sigma = 1 #defines Gaussian kernel width
p = 100 #number of points on x-axis

# Display examples of the kernels
x_test = np.linspace(0,1.00,p)  # uniformly sample interval [0,1]
j_list = [5, 36, 46, 96]  #list of indices for example kernels

Kdisplay = np.zeros((p,len(j_list)),dtype=float)

for i in range(p):
    for j in range(len(j_list)):
        Kdisplay[i,j]= np.exp(-(x_test[i]-x_test[j_list[j]])**2/(2*sigma**2))

print('Sigma = ',sigma)
plt.plot(x_test, Kdisplay)
plt.title('Example Kernels')
plt.xlabel('x')
plt.ylabel('Kernel Amplitude')
plt.show()
```

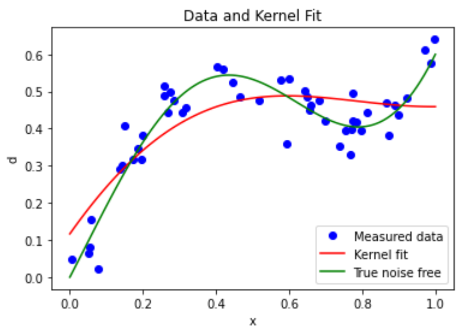
```
Sigma =  1
```



In [11]:
```
# Kernel fitting to data

lam = 0.01 #ridge regression parameter

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        distsq[i,j]=(x[i]-x[j])**2

K = np.exp(-distsq/(2*sigma**2))

alpha = np.linalg.inv(K+lam*np.identity(n))@d
```

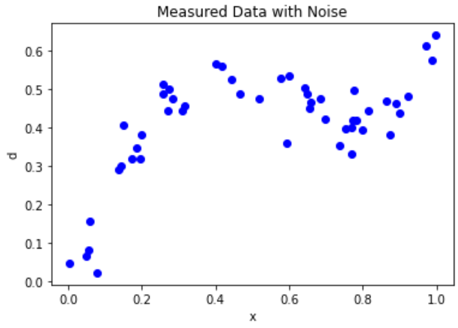
In [12]:
```
# Generate smooth curve corresponding to data fit

distsq_xtest = np.zeros((p,n),dtype=float)
for i in range(0,p):
    for j in range(0,n):
        distsq_xtest[i,j] = (x_test[i]-x[j])**2

dtest = np.exp(-distsq_xtest/(2*sigma**2))@alpha

dtrue = x_test*x_test + 0.4*np.sin(1.5*np.pi*x_test)  # noise free data for comparison

print('Sigma = ',sigma)
print('Lambda = ',lam)
plt.plot(x,d,'bo',label='Measured data')
plt.plot(x_test,dtest,'r',label='Kernel fit')
plt.plot(x_test,dtrue,'g',label='True noise free')
plt.title('Data and Kernel Fit')
plt.legend(loc='lower right')
plt.xlabel('x')
plt.ylabel('d')
plt.show()
```

```
Sigma =  1
Lambda =  0.01
```



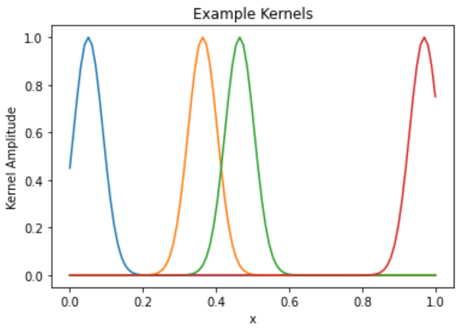
lamda = 1, sigma = 0.04

In [13]:
```
np.random.seed(1024)  # ensure same noise for each run

# number of training points
n = 50

# sample n random points between 0 and 1
x = np.random.rand(n,1)

# set d = x^2 + .4 sin(1.5 pi x) + noise
d = x*x + 0.4*np.sin(1.5*np.pi*x) +0.04*np.random.randn(n,1)

# plot result
plt.plot(x,d,'bo')
plt.xlabel('x')
plt.ylabel('d')
plt.title('Measured Data with Noise')
plt.show()
```



In [14]:
```
sigma = 0.04 #defines Gaussian kernel width
p = 100 #number of points on x-axis

# Display examples of the kernels
x_test = np.linspace(0,1.00,p)  # uniformly sample interval [0,1]
j_list = [5, 36, 46, 96]  #list of indices for example kernels

Kdisplay = np.zeros((p,len(j_list)),dtype=float)

for i in range(p):
    for j in range(len(j_list)):
        Kdisplay[i,j]= np.exp(-(x_test[i]-x_test[j_list[j]])**2/(2*sigma**2))

print('Sigma = ',sigma)
plt.plot(x_test, Kdisplay)
plt.title('Example Kernels')
plt.xlabel('x')
plt.ylabel('Kernel Amplitude')
plt.show()
```

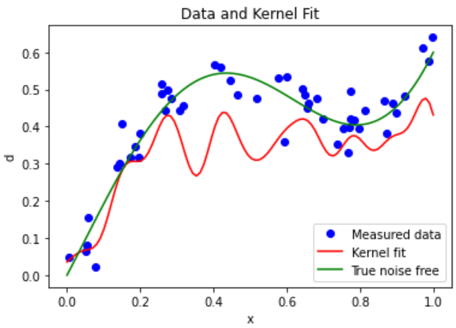
```
Sigma =  0.04
```



In [15]:
```
# Kernel fitting to data

lam = 1 #ridge regression parameter

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        distsq[i,j]=(x[i]-x[j])**2

K = np.exp(-distsq/(2*sigma**2))

alpha = np.linalg.inv(K+lam*np.identity(n))@d
```

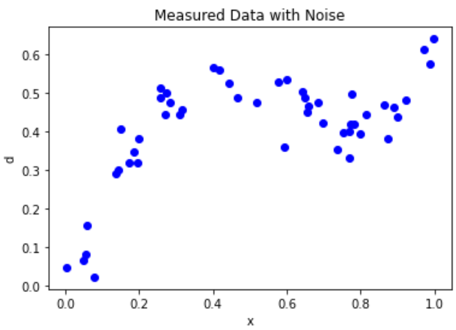
In [16]:
```python
# Generate smooth curve corresponding to data fit

distsq_xtest = np.zeros((p,n),dtype=float)
for i in range(0,p):
    for j in range(0,n):
        distsq_xtest[i,j] = (x_test[i]-x[j])**2

dtest = np.exp(-distsq_xtest/(2*sigma**2))@alpha

dtrue = x_test*x_test + 0.4*np.sin(1.5*np.pi*x_test)  # noise free data for comparison

print('Sigma = ',sigma)
print('Lambda = ',lam)
plt.plot(x,d,'bo',label='Measured data')
plt.plot(x_test,dtest,'r',label='Kernel fit')
plt.plot(x_test,dtrue,'g',label='True noise free')
plt.title('Data and Kernel Fit')
plt.legend(loc='lower right')
plt.xlabel('x')
plt.ylabel('d')
plt.show()
```

```
Sigma =  0.04
Lambda =  1
```



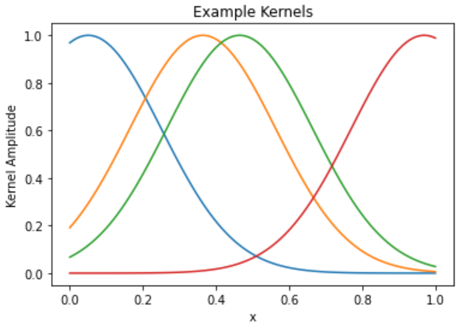
lamda = 1, sigma = 0.2

In [17]:
```python
np.random.seed(1024)  # ensure same noise for each run

# number of training points
n = 50

# sample n random points between 0 and 1
x = np.random.rand(n,1)

# set d = x^2 + .4 sin(1.5 pi x) + noise
d = x*x + 0.4*np.sin(1.5*np.pi*x) +0.04*np.random.randn(n,1)

# plot result
plt.plot(x,d,'bo')
plt.xlabel('x')
plt.ylabel('d')
plt.title('Measured Data with Noise')
plt.show()
```



In [18]:
```
sigma = 0.2 #defines Gaussian kernel width
p = 100 #number of points on x-axis

# Display examples of the kernels
x_test = np.linspace(0,1.00,p)  # uniformly sample interval [0,1]
j_list = [5, 36, 46, 96]  #list of indices for example kernels

Kdisplay = np.zeros((p,len(j_list)),dtype=float)

for i in range(p):
    for j in range(len(j_list)):
        Kdisplay[i,j]= np.exp(-(x_test[i]-x_test[j_list[j]])**2/(2*sigma**2))

print('Sigma = ',sigma)
plt.plot(x_test, Kdisplay)
plt.title('Example Kernels')
plt.xlabel('x')
plt.ylabel('Kernel Amplitude')
plt.show()
```

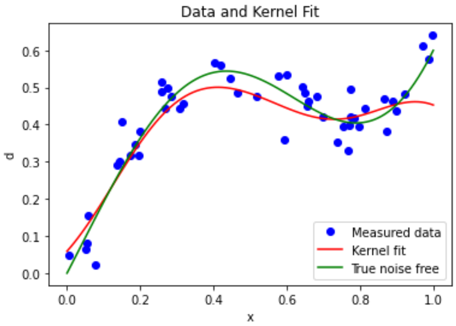
```
Sigma =  0.2
```



In [19]:
```
# Kernel fitting to data

lam = 1 #ridge regression parameter

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        distsq[i,j]=(x[i]-x[j])**2

K = np.exp(-distsq/(2*sigma**2))

alpha = np.linalg.inv(K+lam*np.identity(n))@d
```

In [20]:
```
# Generate smooth curve corresponding to data fit

distsq_xtest = np.zeros((p,n),dtype=float)
for i in range(0,p):
    for j in range(0,n):
        distsq_xtest[i,j] = (x_test[i]-x[j])**2

dtest = np.exp(-distsq_xtest/(2*sigma**2))@alpha

dtrue = x_test*x_test + 0.4*np.sin(1.5*np.pi*x_test)  # noise free data for comparison

print('Sigma = ',sigma)
print('Lambda = ',lam)
plt.plot(x,d,'bo',label='Measured data')
plt.plot(x_test,dtest,'r',label='Kernel fit')
plt.plot(x_test,dtrue,'g',label='True noise free')
plt.title('Data and Kernel Fit')
plt.legend(loc='lower right')
plt.xlabel('x')
plt.ylabel('d')
plt.show()
```

```
Sigma =  0.2
Lambda =  1
```



A low sigma value will lead to function having more bumps and being very erractic. Higher values of sigma will lead to the curve being very smooth and utlimately end up approximating the dataset more generally than if the sigma was lower. Lower sigma vals will try to catch every little perturbation in the dataset.

High values of lamda often overshoot the dataset and make very error prone curve fits. You essentially underfit the data. Low values of lamda do the exact opposite.

1c)

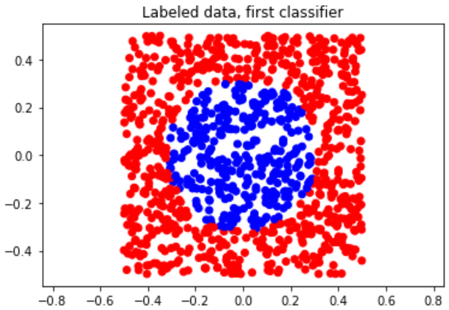
We could apply the principle of cross validation to select appropriate paramters for lamda and sigma. We could split the data into test and train subsets and train the classifier on the training dataset and after training apply the test dataset to the model to see how it performs. This deals with the issue of overfitting and underfitting.

In [ ]:

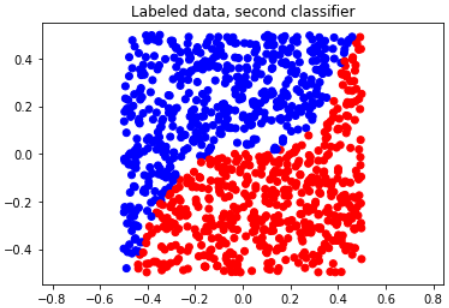
2a)

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

p = int(2) #features
n = int(1000) #examples

## generate training data
X = np.random.rand(n,p)-0.5
Y1 = np.sign(np.sum(X**2,1)-.1).reshape((-1, 1))

Y2 = np.sign(5*X[:,[0]]**3-X[:,[1]])
Y = np.hstack((Y1, Y2))
```

In [2]:
```
# Plot training data for first classification problem
plt.scatter(X[:,0], X[:,1], color=['b' if i==-1 else 'r' for i in Y1[:,0]])
plt.axis('equal')
plt.title('Labeled data, first classifier')
plt.show()
```



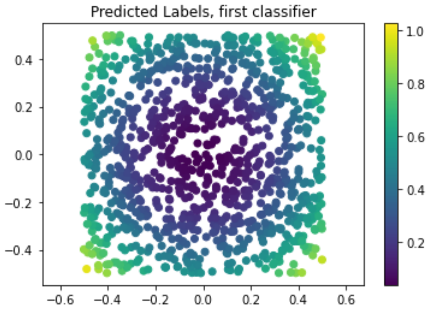
In [3]:
```
# Plot training data for second classification problem
plt.scatter(X[:,0], X[:,1], color=['b' if i==-1 else 'r' for i in Y2[:,0]])
plt.title('Labeled data, second classifier')
plt.axis('equal')
plt.show()
```

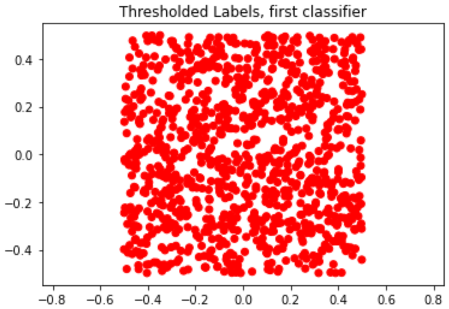


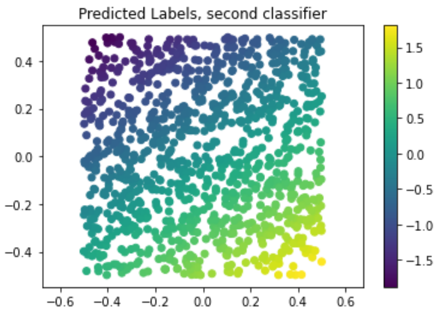
In [4]:
```
# Train Classifiers

sigma = 5
lam = 0.01

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        d = np.linalg.norm(X[i,:]-X[j,:])
        distsq[i,j]=d**2

K = np.exp(-distsq/(2*sigma**2))

alpha1 = np.linalg.inv(K+lam*np.identity(n))@Y1
alpha2 = np.linalg.inv(K+lam*np.identity(n))@Y2
```

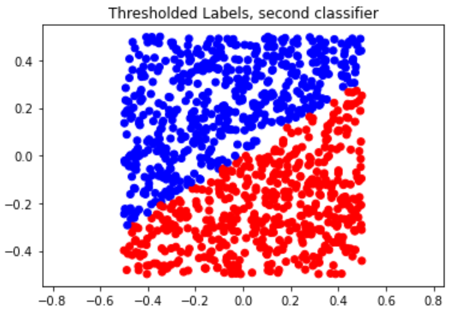
In [5]:
```
# Predict labels

Yhat = K@np.hstack((alpha1, alpha2))
Yhat_thresh=np.sign(Yhat)
```

In [6]:
```python
# Display results

plt.scatter(X[:,0], X[:,1], c=Yhat[:,0])
plt.colorbar()
plt.title('Predicted Labels, first classifier')
plt.axis('equal')
plt.show()

plt.scatter(X[:,0], X[:,1], color=['b' if i==-1 else 'r' for i in Yhat_thresh[:,0]])
plt.axis('equal')
plt.title('Thresholded Labels, first classifier')
plt.show()

plt.scatter(X[:,0], X[:,1], c=Yhat[:,1])
plt.title('Predicted Labels, second classifier')
plt.colorbar()
plt.axis('equal')
plt.show()

plt.scatter(X[:,0], X[:,1], color=['b' if i==-1 else 'r' for i in Yhat_thresh[:,1]])
plt.axis('equal')
plt.title('Thresholded Labels, second classifier')
plt.show()
```









In [7]:
```
err_c1 = np.sum(np.abs(Yhat_thresh[:,0]-Y[:,0]))
print('Errors, first classifier:', err_c1)

err_c2 = np.sum(np.abs(Yhat_thresh[:,1]-Y[:,1]))
print('Errors, second classifier:', err_c2)
```

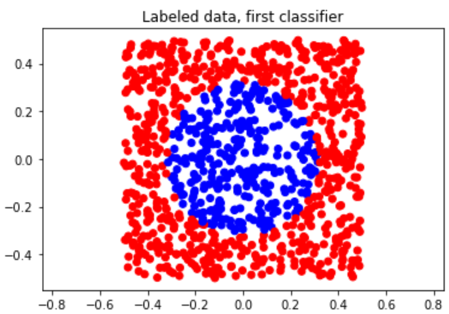
```
Errors, first classifier: 632.0
Errors, second classifier: 142.0
```

In [ ]:

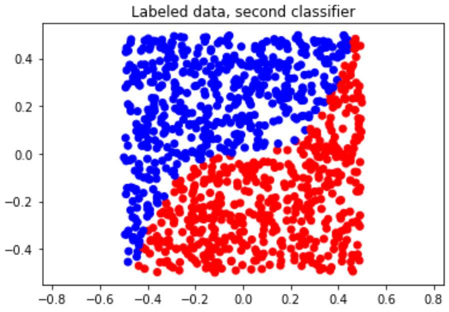
2b)

In [8]:
```
p = int(2) #features
n = int(1000) #examples

## generate training data
X = np.random.rand(n,p)-0.5
Y1 = np.sign(np.sum(X**2,1)-.1).reshape((-1, 1))

Y2 = np.sign(5*X[:,[0]]**3-X[:,[1]])
Y = np.hstack((Y1, Y2))
```

In [9]:
```
# Plot training data for first classification problem
plt.scatter(X[:,0], X[:,1], color=['b' if i==-1 else 'r' for i in Y1[:,0]])
plt.axis('equal')
plt.title('Labeled data, first classifier')
plt.show()
```



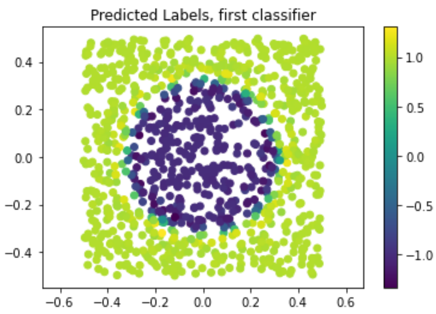
In [10]:
```python
# Plot training data for second classification problem
plt.scatter(X[:,0], X[:,1], color=['b' if i==-1 else 'r' for i in Y2[:,0]])
plt.title('Labeled data, second classifier')
plt.axis('equal')
plt.show()
```

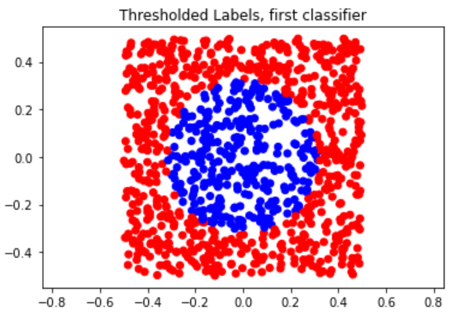


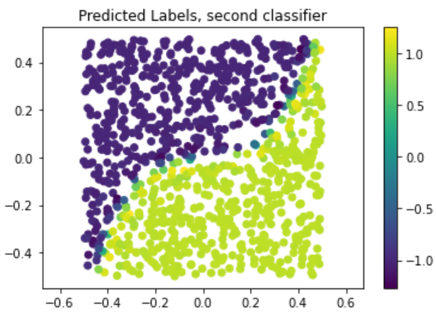
In [11]:
```python
# Train Classifiers

sigma = 0.05
lam = 0.01

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        d = np.linalg.norm(X[i,:]-X[j,:])
        distsq[i,j]=d**2

K = np.exp(-distsq/(2*sigma**2))

alpha1 = np.linalg.inv(K+lam*np.identity(n))@Y1
alpha2 = np.linalg.inv(K+lam*np.identity(n))@Y2
```

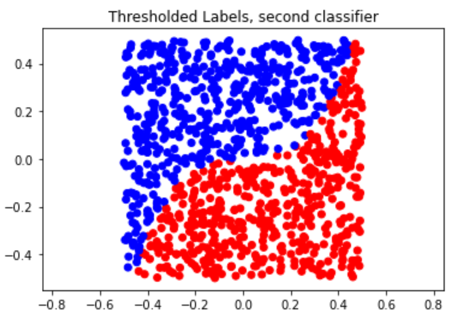
In [12]:
```python
# Predict labels

Yhat = K@np.hstack((alpha1, alpha2))
Yhat_thresh=np.sign(Yhat)
```

In [13]:
```python
# Display results

plt.scatter(X[:,0], X[:,1], c=Yhat[:,0])
plt.colorbar()
plt.title('Predicted Labels, first classifier')
plt.axis('equal')
plt.show()

plt.scatter(X[:,0], X[:,1], color=['b' if i==-1 else 'r' for i in Yhat_thresh[:,0]])
plt.axis('equal')
plt.title('Thresholded Labels, first classifier')
plt.show()

plt.scatter(X[:,0], X[:,1], c=Yhat[:,1])
plt.title('Predicted Labels, second classifier')
plt.colorbar()
plt.axis('equal')
plt.show()

plt.scatter(X[:,0], X[:,1], color=['b' if i==-1 else 'r' for i in Yhat_thresh[:,1]])
plt.axis('equal')
plt.title('Thresholded Labels, second classifier')
plt.show()
```









In [14]:
```python
err_c1 = np.sum(np.abs(Yhat_thresh[:,0]-Y[:,0]))
print('Errors, first classifier:', err_c1)

err_c2 = np.sum(np.abs(Yhat_thresh[:,1]-Y[:,1]))
print('Errors, second classifier:', err_c2)
```

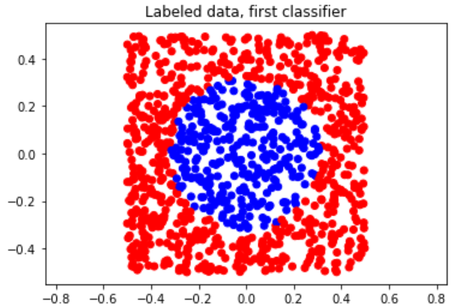
```
Errors, first classifier: 0.0
Errors, second classifier: 0.0
```

In [ ]:

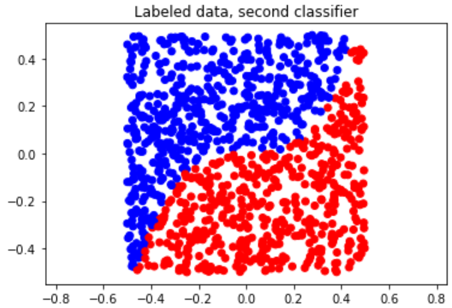
2c)

In [15]:
```
import numpy as np
import matplotlib.pyplot as plt

p = int(2) #features
n = int(1000) #examples

## generate training data
X = np.random.rand(n,p)-0.5
Y1 = np.sign(np.sum(X**2,1)-.1).reshape((-1, 1))

Y2 = np.sign(5*X[:,[0]]**3-X[:,[1]])
Y = np.hstack((Y1, Y2))
```

In [16]:
```
# Plot training data for first classification problem
plt.scatter(X[:,0], X[:,1], color=['b' if i==-1 else 'r' for i in Y1[:,0]])
plt.axis('equal')
plt.title('Labeled data, first classifier')
plt.show()
```



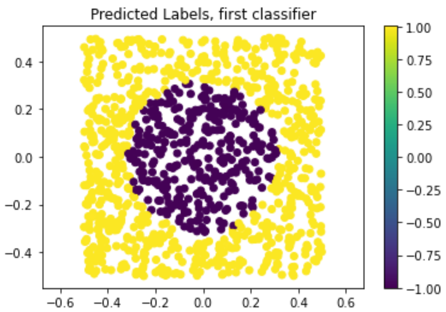
In [17]:
```
# Plot training data for second classification problem
plt.scatter(X[:,0], X[:,1], color=['b' if i==-1 else 'r' for i in Y2[:,0]])
plt.title('Labeled data, second classifier')
plt.axis('equal')
plt.show()
```

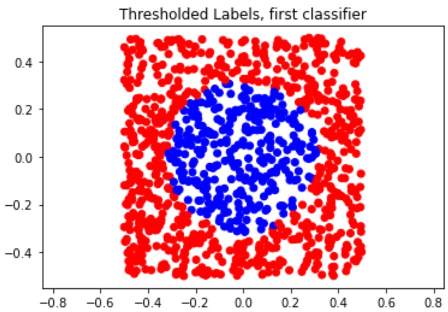


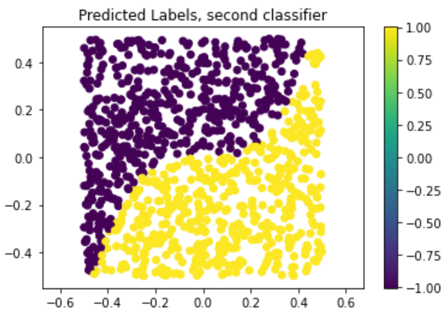
In [18]:
```
# Train Classifiers

sigma = 0.005
lam = 0.01

distsq=np.zeros((n,n),dtype=float)

for i in range(0,n):
    for j in range(0,n):
        d = np.linalg.norm(X[i,:]-X[j,:])
        distsq[i,j]=d**2

K = np.exp(-distsq/(2*sigma**2))

alpha1 = np.linalg.inv(K+lam*np.identity(n))@Y1
alpha2 = np.linalg.inv(K+lam*np.identity(n))@Y2
```

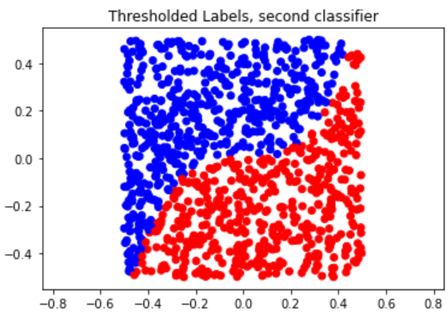
In [19]:
```
# Predict labels

Yhat = K@np.hstack((alpha1, alpha2))
Yhat_thresh=np.sign(Yhat)
```

In [20]:
```
# Display results

plt.scatter(X[:,0], X[:,1], c=Yhat[:,0])
plt.colorbar()
plt.title('Predicted Labels, first classifier')
plt.axis('equal')
plt.show()

plt.scatter(X[:,0], X[:,1], color=['b' if i==-1 else 'r' for i in Yhat_thresh[:,0]])
plt.axis('equal')
plt.title('Thresholded Labels, first classifier')
plt.show()

plt.scatter(X[:,0], X[:,1], c=Yhat[:,1])
plt.title('Predicted Labels, second classifier')
plt.colorbar()
plt.axis('equal')
plt.show()

plt.scatter(X[:,0], X[:,1], color=['b' if i==-1 else 'r' for i in Yhat_thresh[:,1]])
plt.axis('equal')
plt.title('Thresholded Labels, second classifier')
plt.show()
```









In [21]:
```
err_c1 = np.sum(np.abs(Yhat_thresh[:,0]-Y[:,0]))
print('Errors, first classifier:', err_c1)

err_c2 = np.sum(np.abs(Yhat_thresh[:,1]-Y[:,1]))
print('Errors, second classifier:', err_c2)
```

```
Errors, first classifier: 0.0
Errors, second classifier: 0.0
```

We see that the sigma parameter when decreased gives much better decision boundaries. Boundaries that classify lower and lower amounts of points incorrectly.

There is a downside to this. You can overfit the data if the sigma is too small as your classifier will try to classofy every little detail as part of the function, even noise. This is undesirable.

In [ ]:

```

```

3. a) $x = \text{sign}\left(\sum_{i}^{N} \alpha_i \cdot K(x, x^i)\right)$

b) As $\alpha_i = 0$ for $i = 1, 2, \ldots 99, 102, 103 \ldots 1000$

Thus

x becomes

$$x = \text{sign}\left(\alpha_{100} K(x, x^{100}) + \alpha_{101} K(x, x^{101})\right)$$