1. Logistic Regression on Toy data

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt

% matplotlib inline
```

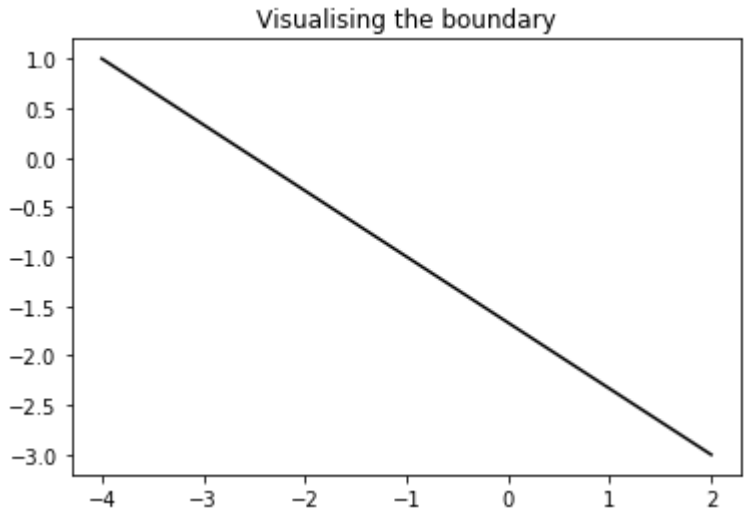
In [2]:

```
def sigmoid(z):
    return np.array([1/(1+np.exp(-i)) for i in z])

def quantise(g):
    return np.array([int(i>=0.5) for i in g])

def plot_error(error):
    plt.plot(range(len(error)),error, color="red")
    plt.title("error vs iteration")
    plt.show()
```

Simplest Case

In [3]:

```
# boundary : 2x1 + 3x2 + 5 = 0

a, b, c = 2, 3, 5

x_axis = np.linspace(-4,2,10)
y_axis = (2*x_axis + 5)*(-1.0/3)

plt.plot(x_axis,y_axis,color='black')
plt.title("Visualising the boundary")
plt.show()
```

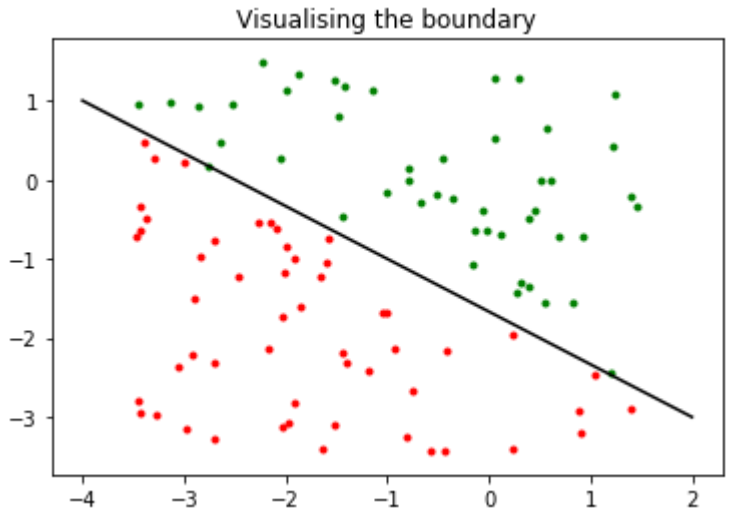


In [4]:

```
# generating data

m = 100

x = np.ones(m), np.random.rand(m,)*5 - 3.5, np.random.rand(m,)*5 - 3.5
x_test = np.ones(m/4), np.random.rand(m/4,)*5 - 3.5, np.random.rand(m/4,)*5 - 3.5


label = np.array(map(int,((2*x[1] + 3*x[2] + 5) >= 0)))
y_t = np.array(map(int,((2*x_test[1] + 3*x_test[2] + 5) >= 0)))
```

In [5]:

```
color = ["red","green"]

for x1,x2,c in zip(x[1],x[2],label):
    plt.scatter(x1,x2,marker='.',color=color[c])

plt.plot(x_axis,y_axis,color='black')
plt.title("Visualising the boundary")
plt.show()
```



In [6]:

```
W = np.ones(3)
y = label
error = []
learning_rate = 1

for _ in range(3000):

    z = np.matmul(W,x)
    y_ = sigmoid(z)

    cross_entropy = np.sum(-y*np.log(y_) - (1-y)*np.log(1-y_))
    error.append(cross_entropy)

    dL_dw = np.array([np.mean((y - y_)*xj) for xj in x])

    W += learning_rate*dL_dw
```

In [7]:

```
plot_error(error)
```

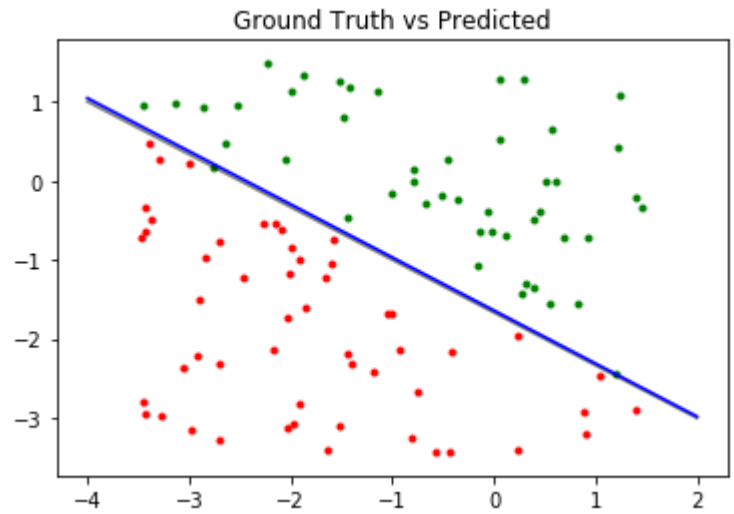
In [8]:

```
print("weights : " + str(W))
print("train error : " + str(error[-1]))
```

```
weights : [ 14.09466283   5.76791832   8.60135652]
train error : 3.39003193226
```

In [9]:

```
for x1,x2,c in zip(x[1],x[2],label):
    plt.scatter(x1,x2,marker='.',color=color[c])


y_axis_2 = (W[1]*x_axis + W[0])*(-1.0/W[2])

plt.plot(x_axis,y_axis,color='black',alpha=0.5)
plt.plot(x_axis,y_axis_2,color='blue')
plt.title("Ground Truth vs Predicted")
plt.show()
```



In [10]:

```
z_test = np.matmul(W,x_test)
y_test = sigmoid(z_test)

test_error = np.sum(-y_t*np.log(y_test) - (1-y_t)*np.log(1-y_test))

print("test_error : "+str(test_error))
```

```
test_error : 0.124419084931
```

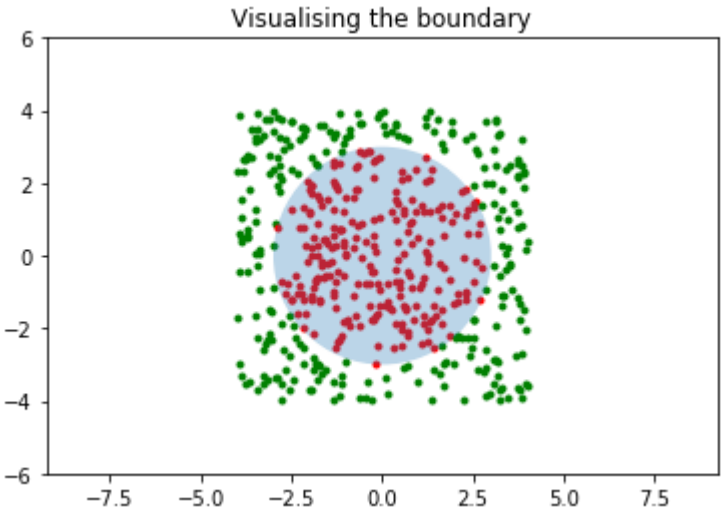
Linearly Inseparable Data

In [11]:

```
# boundary x1^2 + x2^2 = 9

m = 500

x = [np.ones(m), (np.random.rand(m)-0.5)*8, (np.random.rand(m)-0.5)*8]
x = np.array([x[0], x[1], x[2], x[1]**2, x[2]**2])

x_test = [np.ones(m/4), (np.random.rand(m/4)-0.5)*8, (np.random.rand(m/4)-0.5)*8]
x_test = np.array([x_test[0], x_test[1], x_test[2], x_test[1]**2, x_test[2]**2])

label = np.array(map(int,x[1]**2 + x[2]**2 - 9 >= 0))
y_t = np.array(map(int,x_test[1]**2 + x_test[2]**2 - 9 >= 0))
```

In [12]:

```
ax = plt.gca()

ax.cla()
ax.set_xlim((-4, 4))
ax.set_ylim((-6, 6))
ax.set_aspect('equal', adjustable='datalim')

boundary = plt.Circle((0,0),3,alpha=0.3)
ax.add_artist(boundary)

color = ["red","green"]

for x1,x2,c in zip(x[1],x[2],label):
    plt.scatter(x1,x2,marker='.',color=color[c])

plt.title("Visualising the boundary")
plt.show()
```



In [13]:

```
print("Samples from class 0 : "+str(len(label[label==0])))
print("Samples from class 1 : "+str(len(label[label==1])))
```

```
Samples from class 0 : 240
Samples from class 1 : 260
```

In [14]:

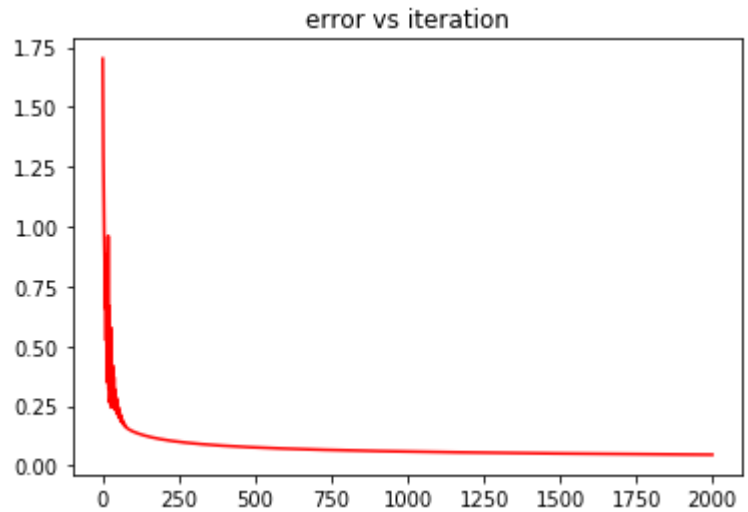
```
W_m = np.random.randn(5)
```

In [15]:

```
y = label
error = []

learning_rate = 0.5

for _ in range(2000):

    z = np.matmul(W_m,x)
    y_ = sigmoid(z)

    cross_entropy = np.mean(-y*np.log(y_) - (1-y)*np.log(1-y_))
    error.append(cross_entropy)

    dL_dw = np.array([np.mean((y - y_)*xj) for xj in x])

    W_m += learning_rate*dL_dw
```

In [16]:

```
plot_error(error)
```



In [17]:

```
print("weights : " + str(W_m))
print("train_error : "+str(error[-1]))
```

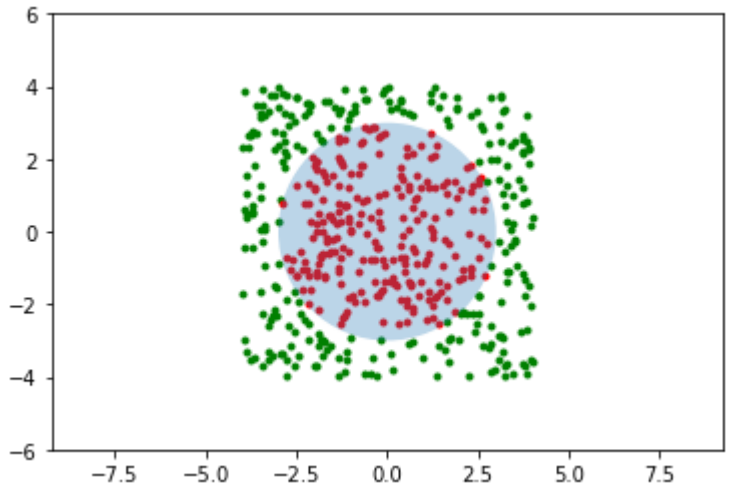
```
weights : [-12.2171674    0.05066335  -0.15477468   1.37882289   1.342
39664]
train_error : 0.0460426923647
```

In [18]:

```
predicted = quantise(sigmoid(np.matmul(W_m,x)))
```

In [19]:

```
plt.title("Ground Truth vs Predicted")

ax = plt.gca()

ax.cla()
ax.set_xlim((-4, 4))
ax.set_ylim((-6, 6))
ax.set_aspect('equal', adjustable='datalim')

boundary = plt.Circle((0,0),3,alpha=0.3)

ax.add_artist(boundary)
for x1,x2,c in zip(x[1],x[2],predicted):
    plt.scatter(x1,x2,marker='.',color=color[c])

plt.show()
```



In [20]:

```
z_test = np.matmul(W_m,x_test)
y_test = sigmoid(z_test)

test_error = np.mean(-y_t*np.log(y_test) - (1-y_t)*np.log(1-y_test))

print("test_error : "+str(test_error))
```

```
test_error : 0.0460510491328
```

What is the major modification that is required for the nonlinear case over the linear one

Had to include higher degree features in the input to account for the non-linearity in the boundary

Which one of batch or stochastic gradient descent do you think will work better here? Why?

In general SGD performs better than batch gradient descent. This is because SGD steps are cheaper and faster to perform, also since they look at each example at a time, error is not sure to decrease at each iteration which is good incase we are stuck at a local minima or saddle point, we can escape to a point of higher error and hope to choose a different path.

The cross entropy loss function is a convex funtion which implies convergence is sure with both the methods given enough iterations.

The general code can be found above.

Test cases are general because they were generated the same way training data was generated Test error is in the range of train error and also train error has saturated, thus code works.