Assignment 5, Question 5, Amogha Sekhar, A53301791

Using gradient descent

In [1]:

```
import numpy as np
import math

#label 0 for 3 and label 1 for 5
def get_data(f1, f2):
    raw_data = []
    X =[]
    y = []
    with open(f1, 'r') as f:
        raw_data  = [l.strip() for l in f.readlines()]
    with open(f2, 'r') as f:
        raw_data.extend([l.strip() for l in f.readlines()])

    n = int(len(raw_data)/2)

    y = [0] * n
    y.extend([1]*n)

    for i in range(len(raw_data)):
        line = raw_data[i]
        temp = []
        for x in line.split(" "):
            temp.append(int(x))
        X.append(temp)
    y = np.array(y)
    X = np.array(X)

    return X, y

X_train, y_train = get_data('train3_oddYr.txt', 'train5_oddYr.txt')
X_test, y_test =  get_data('test3_oddYr.txt', 'test5_oddYr.txt')
```

In [2]:

```
#sigmoid function

def sigmoid(x):
        return 1.0 / (1.0 + np.exp(-x))
```

In [3]:

```
#weight vector
w= np.zeros([64])
```

In [4]:

```
lr= 0.2/len(X_train)
print(lr)
```

```
0.00014285714285714287
```

In [5]:

```
#number of iterations
iter= 50000
```

In [6]:

```
#function to compute the log-likelihood
def log_likelihood(w, x, y):
    sum=0
    for i in range(len(x)):
        pred= sigmoid(np.dot(x[i],w.T))
        if y[i]==0:
            sum+= math.log(1-pred)
        else:
            sum+= math.log(pred)

    return sum
```

In [7]:

```
#shuffling the dataset
import random
Xy= list(zip(X_train, y_train))
random.shuffle(Xy)
X_train= [d[0] for d in Xy]
y_train = [d[1] for d in Xy]
```

In [8]:

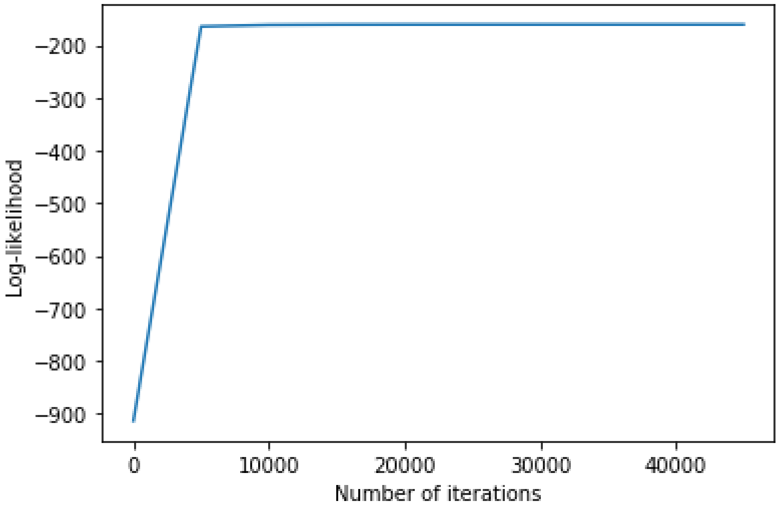
```
iterations= []
ll= []
error_rate= []
error_rate_3= []
error_rate_5= []

for i in range(iter):
    for l in range(len(X_train)):
        z= np.dot(X_train[l],w.T)
        pred = sigmoid(z)
        diff = y_train[l] - pred
        grad= diff* X_train[l]
        w += np.dot(lr, grad)
    if i % 5000 == 0:
        print("Iteration: ", i)
        iterations.append(i)
        x= (log_likelihood(w, X_train, y_train))
        print(x)
        ll.append(x)
        error= 0
        error_3= 0
        error_5= 0
        for m in range(len(X_train)):
            z= np.dot(X_train[m], w.T)
            pred= sigmoid(z)
            if round(pred) != y_train[m]:
                if y_train[m]== 0:
                    error_3+= 1
                    error+= 1
                else:
                    error_5+= 1
                    error+= 1

        error_rate.append(error/len(X_train))
        error_rate_3.append(error_3/len(X_train))
        error_rate_5.append(error_5/len(X_train))
```

```
Iteration:  0
-913.8448720707122
Iteration:  5000
-164.4773374079698
Iteration:  10000
-161.49109983965346
Iteration:  15000
-160.90555565402974
Iteration:  20000
-160.7563439548372
Iteration:  25000
-160.7138392321347
Iteration:  30000
-160.7010385746272
Iteration:  35000
-160.69707740043168
Iteration:  40000
-160.6958425889954
```

```
Iteration:  45000
-160.69546303231613
```

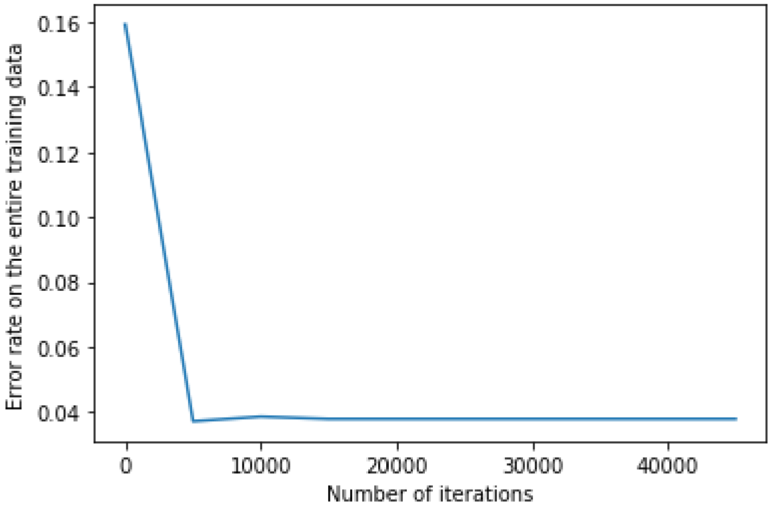
In [9]:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(iterations, ll)
plt.xlabel("Number of iterations")
plt.ylabel("Log-likelihood")
plt.show()
```



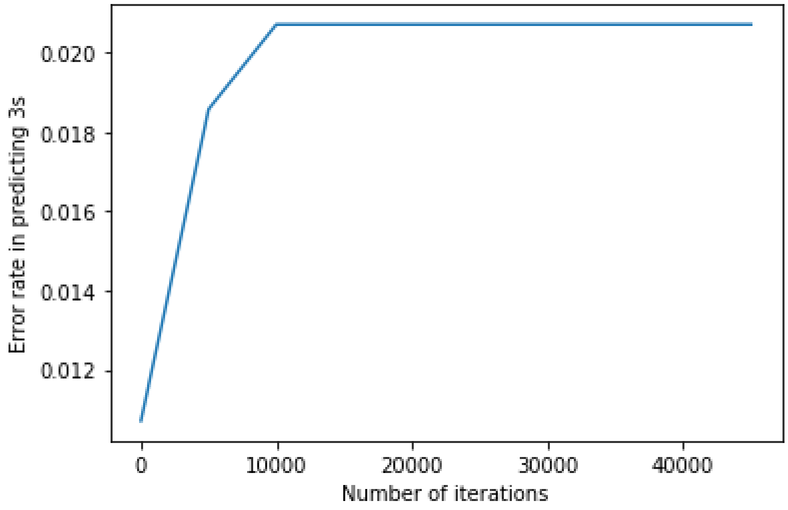
In [10]:

```
#on the training data
plt.plot(iterations, error_rate)
plt.xlabel("Number of iterations")
plt.ylabel("Error rate on the entire training data")
plt.show()
```



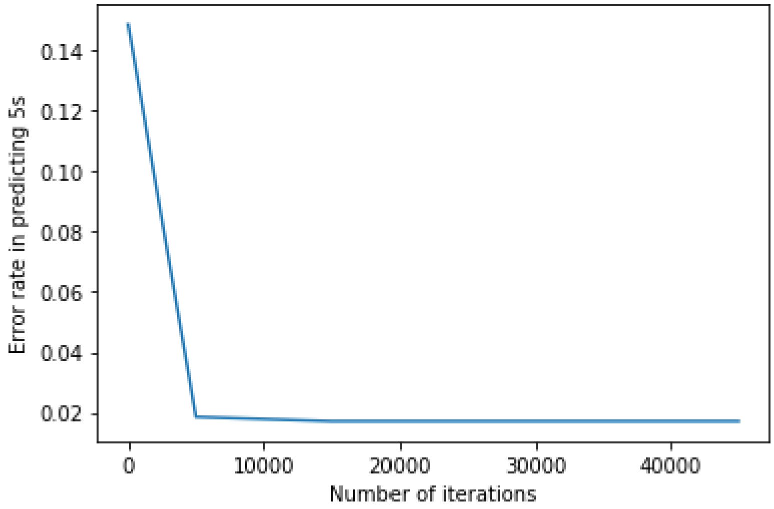
In [11]:

```
#on the training data
plt.plot(iterations, error_rate_3)
plt.xlabel("Number of iterations")
plt.ylabel("Error rate in predicting 3s")
plt.show()
```



In [12]:

```
#on the training data
plt.plot(iterations, error_rate_5)
plt.xlabel("Number of iterations")
plt.ylabel("Error rate in predicting 5s")
plt.show()
```



In [13]:

```
print(w)
```

```
[-0.70142002 -1.78974526 -1.09517625 -1.56056707 -0.61186491 -1.19527036
  0.80513707  1.97996181 -0.30630571 -0.27336541  0.33760441 -0.03573582
 -0.7001968   1.00595075 -1.49952549 -1.51410911  4.53457038  1.39801323
  1.62944226  0.09589359  1.03627812 -2.47825194 -2.46698309 -2.94432925
  0.7533847   0.36389173  0.79236113 -0.36526679 -0.53075228 -2.81149831
  0.53303721 -0.06521238  0.66903417  1.33370927  0.11273271 -0.48363481
 -0.63321262 -0.02991974 -0.67736535 -0.06153866  1.34318369 -0.30175553
 -0.45830022 -0.22661753 -0.05369975 -1.16850445  1.03719343 -1.89507721
  1.75919754 -0.77950427  1.42494901  0.7411996   0.54165717 -0.47592851
  0.12202656 -1.76627149  0.74513701  0.35908298  0.7887806   2.71257091
  0.42891639  0.75510338  0.99147507 -0.63298529]
```

In [14]:

```
#Error rate on the combined training set

errors  = 0
for i in range (len(X_train)):
    z = np.dot(X_train[i], w.T)
    pred = sigmoid(z)
    if round(pred) != y_train[i]:
        errors += 1
print ("Error rate on training set: ", errors/len(X_train))
```

```
Error rate on training set:  0.03785714285714286
```

In [15]:

```
#Error rate on the training set for predicting 3s and 5s separately

error_3= 0
error_5=0

for i in range(len(X_train)):
    z= np.dot(X_train[i], w.T)
    pred= sigmoid(z)
    if round(pred) != y_train[i]:
        if y_train[i]== 0:
            error_3+= 1
        else:
            error_5+= 1
print("Error rate for predicting 3 on training set: ", error_3/len(X_train)
print("Error rate for predicting 5 on training set: ", error_5/len(X_train)
```

```
Error rate for predicting 3 on training set:  0.020714285714285713
Error rate for predicting 5 on training set:  0.017142857142857144
```

In [16]:

```
#Error rate on the combined test set

errors  = 0
for i in range (len(X_test)):
    z = np.dot(X_test[i], w.T)
    pred = sigmoid(z)
    if round(pred) != y_test[i]:
        errors += 1
print ("Error rate on test set: ", errors/len(X_test))
```

```
Error rate on test set:  0.06625
```

In [17]:

```
#Error rate on the test set for predicting 3s and 5s separately

error_3= 0
error_5=0

for i in range(len(X_test)):
    z= np.dot(X_test[i], w.T)
    pred= sigmoid(z)
    if round(pred) != y_test[i]:
        if y_test[i]== 0:
            error_3+= 1
        else:
            error_5+= 1
print("Error rate for predicting 3 on test set: ", error_3/len(X_test))
print("Error rate for predicting 5 on test set: ", error_5/len(X_test))
```

```
Error rate for predicting 3 on test set:  0.0375
Error rate for predicting 5 on test set:  0.02875
```

In [ ]: