5.5

*Not using the last four days of 2000 for calculating first four days of 2001.

a)

[a4 a3 a2 a1] =

[0.04678134 -0.01364498 0.01974237 0.94520006]

Since we observe that mean squared error is high, both for training and test data, we can conclude that index on a day can't be predicted very accurately with indices of preceding four days. Moreover, in real scenario too index on one day is dependent on many other factors. So just last four days' indices might not be a good basis.

b)

Mean Squared error on data from year 2000: 13918.6329208

Mean Squared error on data from year 2001: 3018.26784093

No, I would not recommend this model for Stock market prediction because. The model has high mean squared error. Surprisingly, this works better on test data than on training data.

c)

```
#Source
"""
Created on Thu Oct 27 21:33:02 2016

@author: gopal
"""
import numpy as np
nasdaq2k = np.loadtxt('nasdaq00.txt') #249
nasdaq2k1 = np.loadtxt('nasdaq01.txt') #248

d = 4
# xn = linear combination of xn-1, xn-2, xn-3, xn-4
A = np.zeros(shape=(d,d), dtype='float')
B = np.zeros(d, dtype='float')
for i in range(d,len(nasdaq2k)):
    X = np.array(nasdaq2k[i-d:i])
    A= np.add(A, np.outer(X,X))
    B= np.add(B, nasdaq2k[i] * X)

a = np.linalg.solve(A,B)
print('[a4 a3 a2 a1] = ', a)

def Prob(y, X, W):
    Py_x = 1/(np.sqrt(2 * np.pi)) * np.exp( - (y-W*X)**2 / 2)
    return Py_x

mse2k =0.0
for i in range (d, len(nasdaq2k)):
    X = np.array(nasdaq2k[i-d:i])
    Y = nasdaq2k[i]
    mse2k = mse2k + ((Y-np.dot(a,X))**2)

mse2k = mse2k / (len(nasdaq2k)-d)
print('Mean Squared error on data from year 2000:', mse2k)
```

```
mse2k1 =0.0
for i in range (d, len(nasdaq2k1)):
    X = np.array(nasdaq2k1[i-d:i])
    Y = nasdaq2k1[i]
    mse2k1 = mse2k1 + ((Y-np.dot(a,X))**2)

mse2k1 = mse2k1 / (len(nasdaq2k1)-d)
print('Mean Squared error on data from year 2001:', mse2k1)
```
--xx--

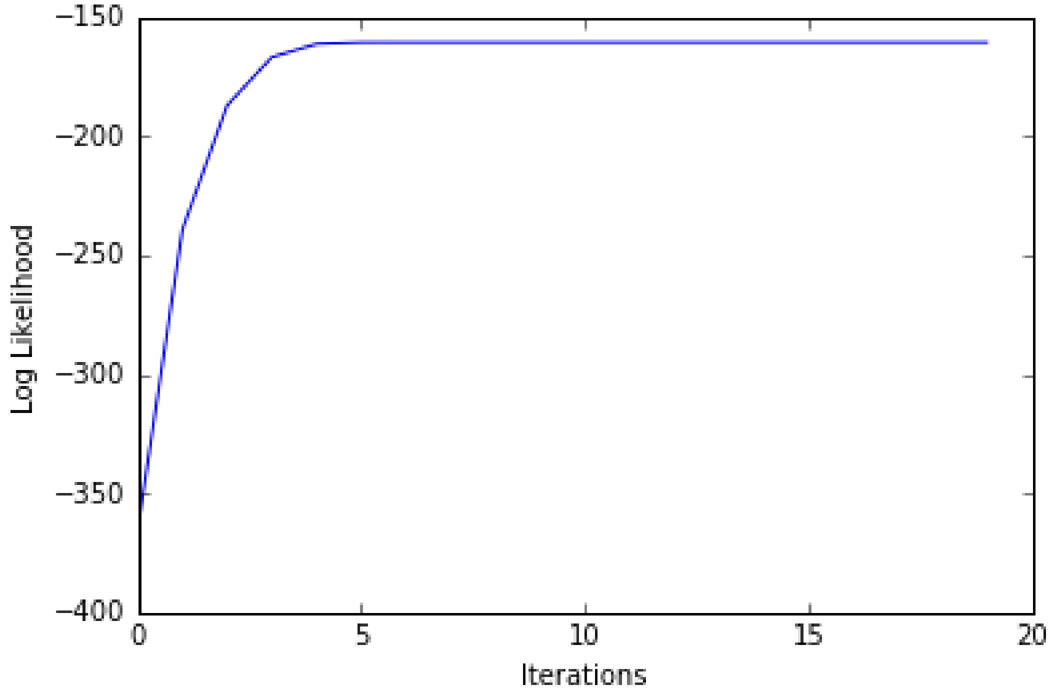
5.6

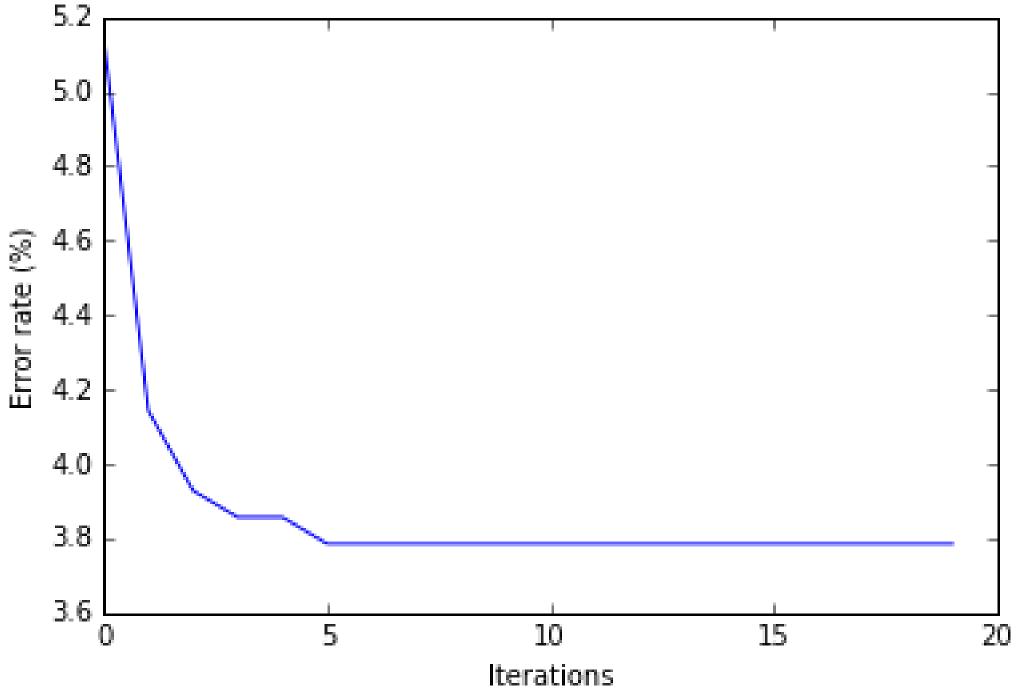
Method Used: Newton's Method for Logistic Regression.

Iteration	Loglikelihood	Error Rate (%)
1	-364.9429571	5.142857143
2	-239.181425	4.142857143
3	-187.1812638	3.928571429
4	-166.8755431	3.857142857
5	-161.3423654	3.857142857
6	-160.7069619	3.785714286
7	-160.6947506	3.785714286
8	-160.6947448	3.785714286
9	-160.6947448	3.785714286
10	-160.6947448	3.785714286
11	-160.6947448	3.785714286
12	-160.6947448	3.785714286
13	-160.6947448	3.785714286
14	-160.6947448	3.785714286
15	-160.6947448	3.785714286
16	-160.6947448	3.785714286
17	-160.6947448	3.785714286
18	-160.6947448	3.785714286
19	-160.6947448	3.785714286
20	-160.6947448	3.785714286

a) Weight Matrix (W [8x8])

-0.6987	-1.7909	-1.0958	-1.5593	-0.6128	-1.1960	0.8050	1.9817
-0.3070	-0.2752	0.3373	-0.0348	-0.7024	1.0082	-1.5007	-1.5141
4.5384	1.3988	1.6299	0.0954	1.0376	-2.4795	-2.4670	-2.9457
0.7536	0.3637	0.7941	-0.3656	-0.5324	-2.8131	0.5335	-0.0648
0.6672	1.3348	0.1124	-0.4831	-0.6311	-0.0300	-0.6769	-0.0605
1.3431	-0.3001	-0.4579	-0.2279	-0.0546	-1.1705	1.0381	-1.8979
1.7598	-0.7812	1.4258	0.7418	0.5411	-0.4761	0.1211	-1.7666
0.7468	0.3606	0.7859	2.7191	0.4306	0.7549	0.9919	-0.6338





b) Error rate on test images = 6.625%

```python
# Source
"""
Created on Thu Oct 27 23:56:30 2016

@author: gopal
"""
print ("Logistic Regression using Newton's Method")

import numpy as np
import matplotlib.pyplot as plt
def prepData():
    train3 = np.loadtxt('train3.txt', dtype=int)
    train5 = np.loadtxt('train5.txt', dtype=int)
    test3 = np.loadtxt('test3.txt', dtype=int)
    test5 = np.loadtxt('test5.txt', dtype=int)
    trainlabel3 = np.zeros(len(train3), dtype=int)
    trainlabel5 = np.ones(len(train5), dtype=int)
    testlabel3 = np.zeros(len(test3), dtype=int)
    testlabel5 = np.ones(len(test5), dtype=int)
    trainSet = np.append(train3, train5, axis=0)
    trainLabel = np.append(trainlabel3, trainlabel5)
    testSet = np.append(test3, test5, axis=0 )
    testLabel = np.append(testlabel3, testlabel5)
    return trainSet, trainLabel, testSet, testLabel

trainData, trainLabel, testData, testLabel = prepData()

def sigmoid(z):
    return 1/(1+np.exp(-z))

def derivatives(y,W,X):
    sigma = sigmoid(np.dot(W,X))
    gradient = (y-sigma)*X
    hesian = -(sigma*(1-sigma))*np.outer(X,X)
    return gradient, hesian

def logLikelihood(data, label, W):
    llhood = 0.0
    for i in range(len(data)):
        llhood += ((label[i]*np.log(sigmoid(np.dot(W, data[i])))) + ((1-
label[i])*np.log(sigmoid(-np.dot(W, data[i])))))
    return llhood

def Classify(data, W):
    res = np.dot(data, W)
    result =np.zeros(len(res))
    for i in range(len(res)):
        if res[i]>=0 :
            result[i] = 1
    return result

W = np.zeros(trainData.shape[1])
iterations = 20
loops = np.array([])
llhoods = np.array([])
errorRates = np.array([])
```

```python
for loop in range(iterations):
    G = np.zeros(trainData.shape[1])
    H = np.zeros(shape=(trainData.shape[1], trainData.shape[1]))
    for i in range(len(trainData)):
        gradient, hesian = derivatives(trainLabel[i], W, trainData[i])
        G = G + gradient
        H = H + hesian
    W = W - np.linalg.solve(H, G)
    llhood = logLikelihood(trainData, trainLabel, W)
    classify = Classify(trainData, W)
    errorRate = 100*np.count_nonzero(classify-trainLabel)/len(trainLabel)
    errorRates = np.append(errorRates, [errorRate], axis=0)
    loops = np.append(loops, [loop], axis=0)
    llhoods = np.append(llhoods, [llhood], axis=0)
    print (llhood)

plt.figure()
f, axes = plt.subplots(1,1)
#axes.plot(loops, llhoods)
#axes.set_xlabel("Iterations")
#axes.set_ylabel("Log Likelihood")

axes.plot(loops, errorRates)
axes.set_xlabel("Iterations")
axes.set_ylabel("Error rate (%)")

W_mesh = np.reshape(W, (8,8))

testClassify = Classify(testData, W)
testErrorRate = 100*np.count_nonzero(testClassify-testLabel)/len(testLabel)
print("Test Error Rate:", testErrorRate)
```