handwritten_digit

November 3, 2021

1 Problem 5 - Handwritten digit classification

a) Perform a logistic regression (using gradient ascent or Newton's method) on the images in files ***train3.txt*** and ***train5.txt***. Indicate clearly the algorithm used, and provide evidence that it has converged (or nearly converged) by plotting or printing out the log-likelihood on several iterations of the algorithm, as well as the percent error rate on the images in these files. Also, print out the 64 elements of your solution for the weight vector as an 8x8 matrix.

1.0.1 Part a

1.0.2 In this task, I will be using the Gradient Ascent method of logistic regression to classify handwritten digits from the MNIST database.

[39]:
```
import numpy as np
import matplotlib.pyplot as plt
from pandas import *
```

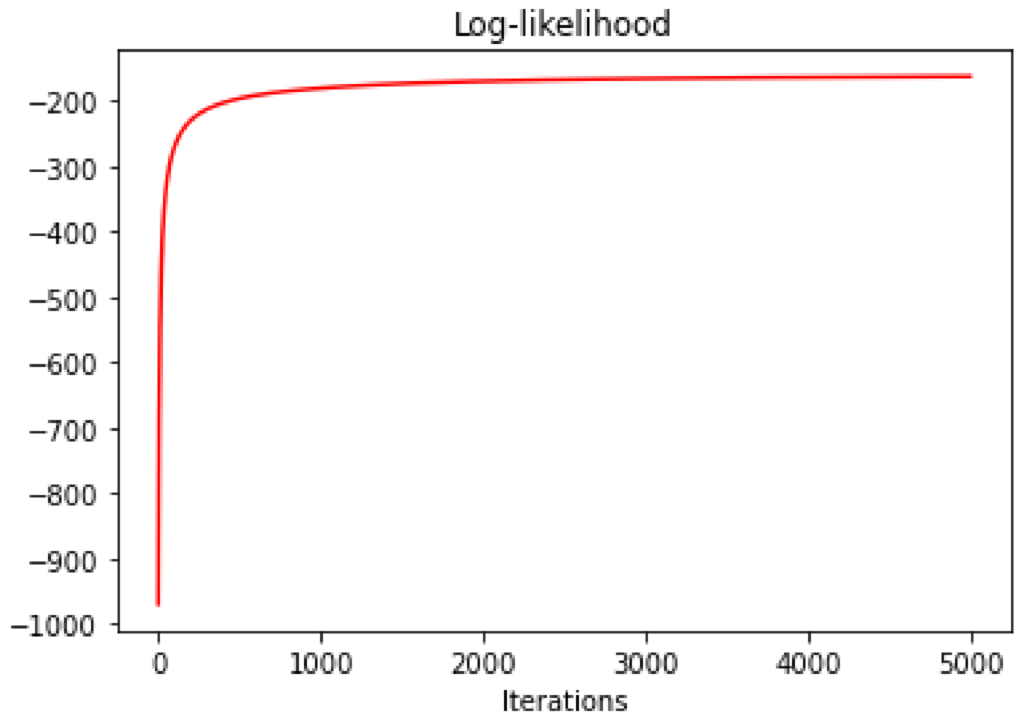
[40]:
```
# Load files and concatenate train and test files together
train3 = np.loadtxt('train3.txt', dtype=int)
train5 = np.loadtxt('train5.txt', dtype=int)
test3 = np.loadtxt('test3.txt', dtype=int)
test5 = np.loadtxt('test5.txt', dtype=int)

train = np.concatenate((train3, train5), axis=0)
test = np.concatenate((test3, test5), axis=0)

train_labels = [0] * train3.shape[0] + [1] * train5.shape[0]
test_labels = [0] * test3.shape[0] + [1] * test5.shape[0]
```

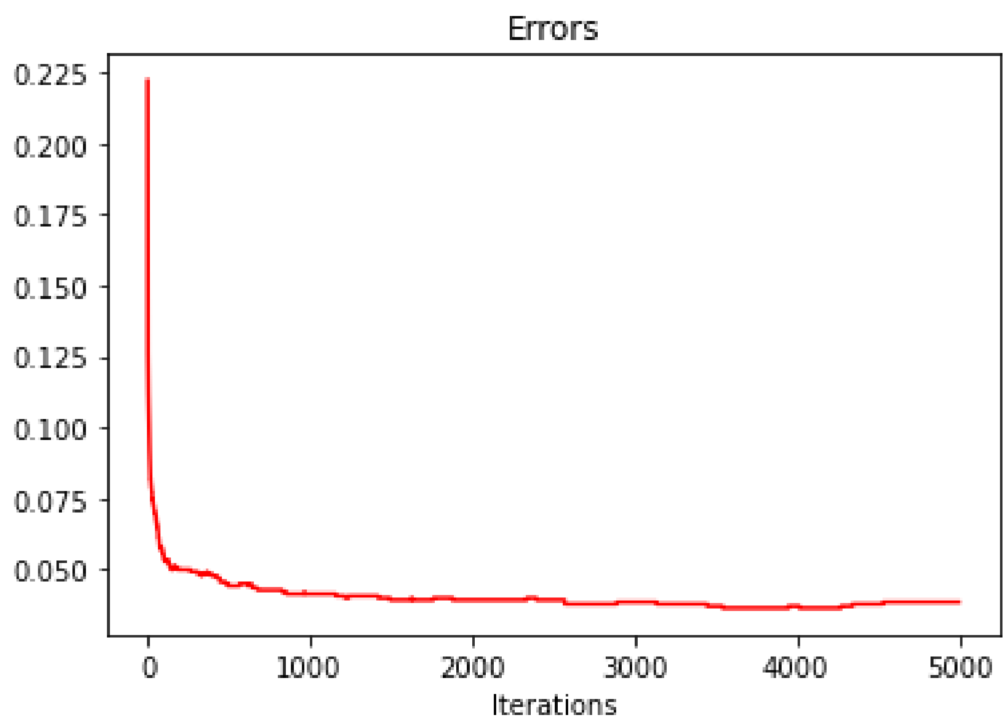
[41]:
```
def sigmoid(x,w):
    return (1.0/(1+np.exp(-np.dot(w,x))))

def gradient(x,y,w):
    return(np.multiply((y - sigmoid(w, x)), x))

def loglikelihood(x,y,w):
    return (y * np.log(sigmoid(w,x)) + (1-y) * np.log(1-sigmoid(w,x)))
```

[42]:
```python
def error_prediction(data, labels, w):
    correct = 0
    for t in range(data.shape[0]):
        sig = sigmoid(w, data[t])
        if (labels[t]==1 and sig>=0.5) or (labels[t]==0 and sig<0.5):
            correct += 1
    error = (data.shape[0] - correct)*1.0 / data.shape[0]
    return error

def gradient_ascent(data, labels, steps):
    alpha = 0.0002
    weights = np.zeros(data.shape[1])
    likelihood_weights = []
    errors = []

    for i in range(steps):
        lw_sum = 0
        grad = 0
        for j in range(data.shape[0]):
            lw_sum += loglikelihood(data[j],labels[j],weights)
            grad += gradient(data[j],labels[j],weights)
        weights = weights + alpha * grad
        likelihood_weights.append(lw_sum)
        errors.append(error_prediction(data,labels,weights))
    return weights, likelihood_weights, errors
```

Log-likelihood and error output

[43]:
```python
steps = 5000
weights, likelihood_weights, errors = gradient_ascent(train, train_labels, steps)
plt.plot(likelihood_weights, 'r-')
plt.xlabel("Iterations")
plt.title("Log-likelihood")
plt.show()
plt.plot(errors, 'r-')
plt.xlabel("Iterations")
plt.title("Errors")
plt.show()
```


Log-likelihood
−200
−300
−400
−500
−600
−700
−800
−900
−1000
0
1000
2000
3000
4000
5000
Iterations


Errors
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0
1000
2000
3000
4000
5000
Iterations

Weight matrix

```
[44]: print('\n'.join(['\t'.join([str(round(cell,7)) for cell in row]) for row in␣
      ↪weights.reshape([8,8])]))
```

```
-0.8797974      -1.4877361      -1.1609024      -1.2031431      -0.711716
-0.8885907      0.8056814       1.7719741
0.0007422       -0.1388434      0.2383485       -0.0676125      -0.4321559
0.7737291       -1.3078541      -1.3306284
3.5719539       1.3666871       1.4310638       0.1802701       0.733658
-2.0656259      -2.4048238      -2.5712567
0.7751998       0.3807532       0.6289521       -0.3051188      -0.4752028
-2.3491697      0.4132579       -0.0314203
0.521081        1.1247831       0.0576895       -0.3606556      -0.639655
-0.1502495      -0.4792577      -0.2629605
1.1833087       -0.2161415      -0.3391773      -0.1164656      0.0660542
-0.9162373      0.8473631       -1.5487284
1.4700634       -0.6591165      1.299139        0.5986168       0.4468133
-0.348847       0.1911971       -1.2810957
0.5834873       0.2840508       0.8985841       1.9238005       0.4840218
0.6502586       0.6400514       -0.4983526
```

1.0.3 Part b

```
[45]: print("Error rate on test3 images:")
      print(error_prediction(test3, [0] * test3.shape[0], weights))

      print("Error rate on test5 images:")
      print(error_prediction(test5, [1] * test5.shape[0], weights))
```

```
Error rate on test3 images:
0.065
Error rate on test5 images:
0.055
```