

Overview: HW3 - Question 4

In this coding question, you'll implement a classifier with logistic regression

$$F(w) = \frac{1}{N} \sum_{i=1}^N \log(1 + e^{-\langle w, x_i \rangle y_i}).$$

For this problem, I would suggest using functions to prepare the dataset, run gradient descent, and return classification error. By doing this, you only have to write the code one time and just use the functions to return results for part (4c).

Loading MNIST Data

In this section, you will learn to load MNIST data. If you do not have tensorflow available on your jupyter notebook, uncomment the next cell, run it, restart the kernel, and comment the next cell once more.

```
In [82]: # !pip3 install scikit-learn
```

```
In [83]: # import statements
import pandas as pd
import numpy as np
import random
from matplotlib import pyplot as plt
from sklearn.datasets import fetch_openml
```

```
In [84]: # this cell will take a minute to run depending on your internet connection
X, y = fetch_openml('mnist_784', version=1, return_X_y=True) # getting data
print('X shape:', X.shape, 'y shape:', y.shape)
```

X shape: (70000, 784) y shape: (70000,)

```
In [85]: # this cell processes some of the data

# if this returns an error of the form "KeyError: 0", then try running the 1
# X = X.values # this converts X from a pandas dataframe to a numpy array

X = X.values
digits = {}
for j in range(10):
    for i in range(len(y)): # takes data assigns it into a dictionary
        digits[int(y[i])].append(X[i].reshape(28,28))
```

```

digits = {j:np.stack(digits[j]) for j in range(10)} # stack everything to be
for j in range(10):
    print('Shape of data with label', j, ': ', digits[j].shape )

```

```

Shape of data with label 0 : (6903, 28, 28)
Shape of data with label 1 : (7877, 28, 28)
Shape of data with label 2 : (6990, 28, 28)
Shape of data with label 3 : (7141, 28, 28)
Shape of data with label 4 : (6824, 28, 28)
Shape of data with label 5 : (6313, 28, 28)
Shape of data with label 6 : (6876, 28, 28)
Shape of data with label 7 : (7293, 28, 28)
Shape of data with label 8 : (6825, 28, 28)
Shape of data with label 9 : (6958, 28, 28)

```

```

In [ ]: # this cell would stack 100 examples from each class together
# this cell also ensures that each pixel is a float between 0 and 1 instead of 0-255
data = []

for i in range(10):
    flattened_images = digits[i][:100].reshape(100,-1)
    data.append(flattened_images)

data = np.vstack(data)
data = data.astype('float32') / 255.0
data.shape

```

```
Out[ ]: (1000, 784)
```

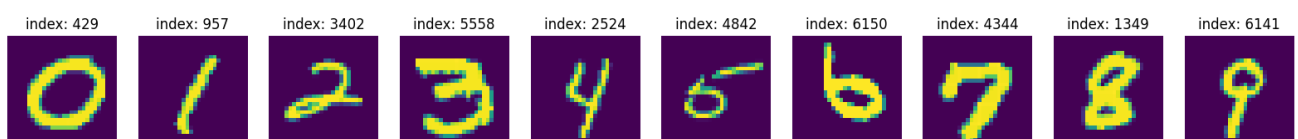
(4a) Plotting

Display one randomly selected image from your training data for each digit class.
Provide the index number for each image.

```

In [ ]: plt.figure(figsize=(20, 10))
for i in range(10):
    index = random.randint(0, digits[i].shape[0] - 1)
    plt.subplot(1, 10, i+1)
    plt.imshow(digits[i][index])
    plt.axis('off')
    plt.title(f"index: {index}")
plt.show()

```



(4b) Label data

Select the first 500 examples of 0's and 1's for this example, those will form the training data $(x_i, y_i) \in \mathbb{R}^{784} \times \{-1, 1\}, i = 1, \dots, 1000$. Assign label $y_i = 1$ for 1s and $y_i = -1$ for 0s. Also, renormalize your x_i so that the pixel values are floats between 0 and 1, instead of ints from 0 to 255. You can do this by augmenting the code given above for stacking data from different classes.

```
In [ ]: x_0 = digits[0][:500].reshape(500,-1)
x_1 = digits[1][:500].reshape(500,-1)

x_train = np.vstack((x_0, x_1))
y_train = np.hstack((-1 * np.ones(500), np.ones(500)))

Out [ ]: (1000, 784)
```

(4c) Running Gradient Descent

Implement and run a Gradient Descent algorithm, with step-size $\mu = 10^{-4}$, to optimize the function above associated with this setup. You should run your algorithm for at least $T = 10,000$ iterations, but if your computer can handle it try $T = 100,000$ or until a reasonable stopping criterion is satisfied. Provide a plot showing the value of $F(w)$ at each iteration. Also, feel free to adjust μ to be larger / smaller if the plot does not match your expectations.

```
In [118...] mu = 1e-4
T = 100000

w_values = []
iterations = range(T)

def F(w):
    sum = 0
    N = len(x_train)
    for i in range(N):
        sum += np.log(1 + np.exp(-y_train[i] * np.dot(w, x_train[i])))
    return sum / N

def dF(w):
    sum = 0
    N = len(x_train)
    for i in range(N):
        sum += -y_train[i] * np.exp(-y_train[i] * np.dot(w, x_train[i])) * x
```

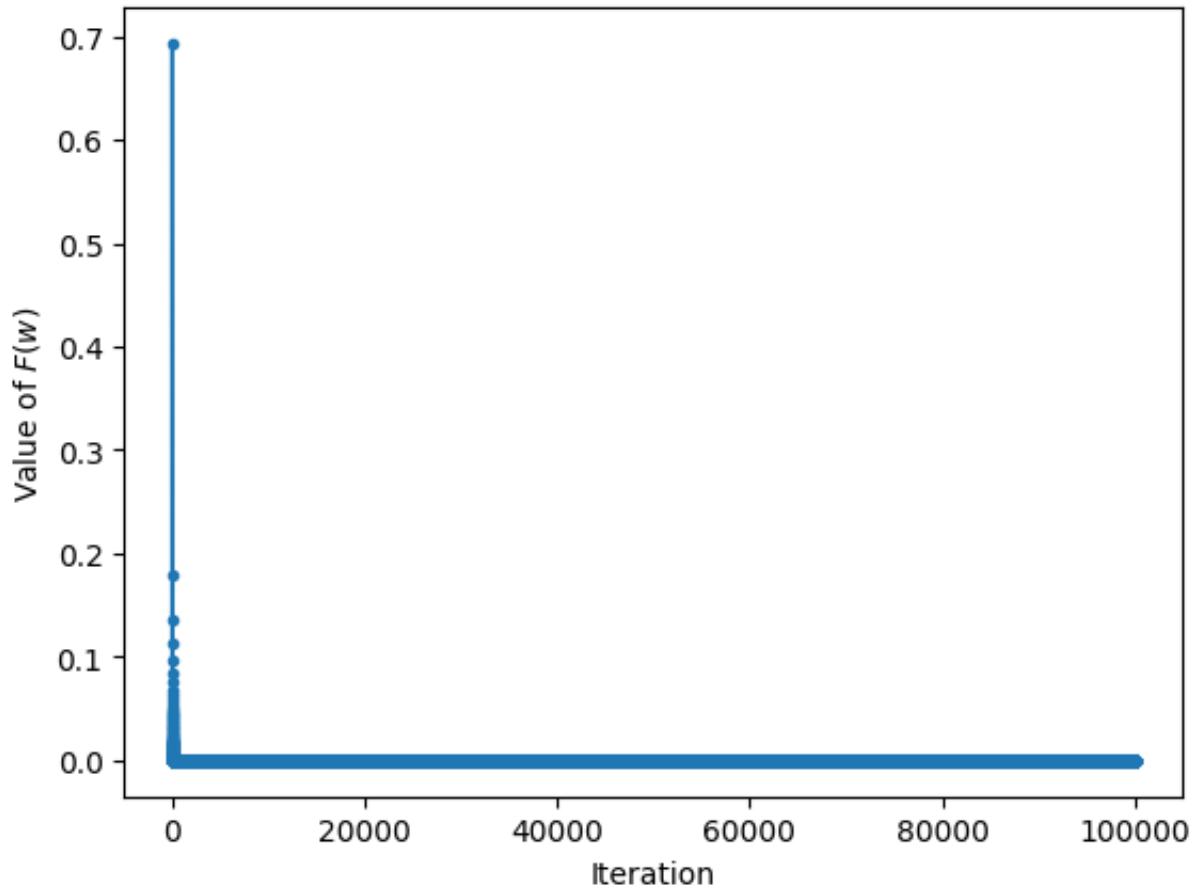
```

return sum / N

w = np.zeros(x_train.shape[1])
for i in iterations:
    w_values.append(F(w))
    w = w - mu * dF(w)

plt.plot(iterations, w_values, marker='.')
plt.xlabel("Iteration")
plt.ylabel(r"Value of $F(w)$")
plt.show()

```



(4d) Discussion about gradient descent

Comment on the resulting plot. In particular, does the shape of $F(w)$ suggest you've successfully converged to a local or global minimum? Does it appear you chose a good stopping criteria? Explain whether your answers to these questions are consistent with the theory we discussed in class (and in the notes). Be specific i.e., point to a specific theorem (or theorems) and indicate why it does or does not explain the behavior of the algorithm. Would the theory dictate a different choice of μ than the one we used?

Put discussion here:

Yes, the plot suggests that we have converged to a minimum. Since we have proved $F(w)$ is convex, the minimum we reached is a global minimum. Notice that $\nabla F(w)$ we computed in homework 2 is bounded, so $F(w)$ is L -smooth. Assuming that our choice of μ is sufficiently small. By a Theorem in Lecture 8,

$$F(w^{(t)}) \leq \frac{\|w^{(0)} - w^*\|^2}{2\mu t} + F(w^*).$$

This suggests that $F(w^{(t)})$ is of order $O(1/t)$, which matches with our result. The theorem suggests us to pick $\mu = 1/L$, but the value of L is unknown so we are unsure whether it is a different choice than ours.

Now, use the w you found from part (a) to classify the first 500 `\emph{test}` data points associated to each of the 0 and 1 handwritten digits. Recall that you need to use the function $y = \text{sign}(w^T x)$ to classify. What was the classification error rate associated with the two digits on the test data (this should be a number between 0 and 1)? What was it on the training data? Does this relationship make sense?

```
In [119... x_0_test = digits[0][500:1000].reshape(500,-1)
x_1_test = digits[1][500:1000].reshape(500,-1)

x_test = np.vstack((x_0_test, x_1_test))
y_test = np.hstack((-1 * np.ones(500), np.ones(500)))

error = 0

for i in range(1000):
    if np.dot(w, x_test[i]) > 0:
        y_test[i] = 1
    else:
        y_test[i] = -1
    error += (y_test[i] != y_train[i])

print("Error rate:", error / 1000 * 100, "%")
```

Error rate: 0.1 %