ESE 4020/5420 - Problem Set #5

Author: `Mohammed Raza Syed`

```
# These are the only imports you can use for Question 1
import numpy as np
import matplotlib.pyplot as plt
```

Question 1: Logistic Regression from Scratch

Load the data

```
# TO-DO: Use np.load(...) to read the data then assign the data to X, and the labels to y.
X = np.load('./data-Q1.npy')
y = np.load('./label-Q1.npy')
```

Visualize one input data point as an image for each of label 0 and label 1.

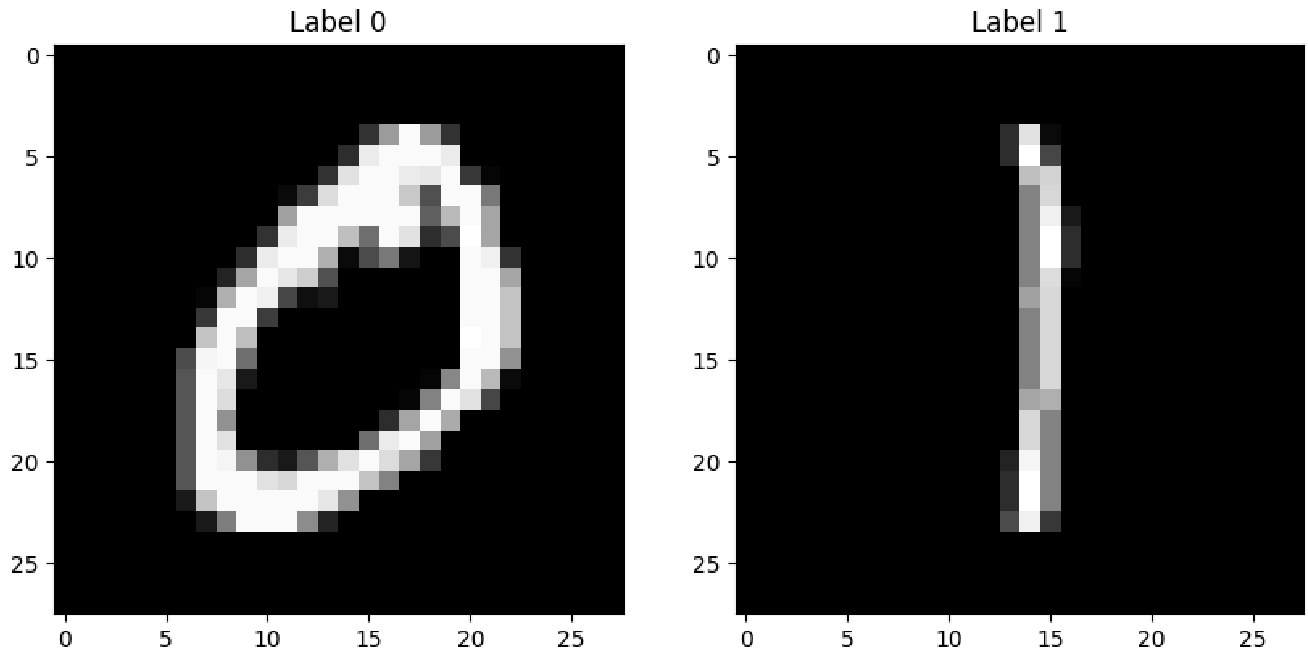
```
# TO-DO: The data should be reshaped back to [28 x 28] to be able to visualize it using plt.imshow(
label_0 = np.where(y==0)[0][0]
label_1 = np.where(y==1)[0][0]

plt.figure(figsize=(10,8))

plt.subplot(1,2,1)
plt.imshow(X[label_0].reshape(28,28),cmap='gray')
plt.title('Label 0')


plt.subplot(1,2,2)
plt.imshow(X[label_1].reshape(28,28),cmap='gray')
plt.title('Label 1')

plt.show()
```



Since the data is in between 0 to 255, normalize the data to [0, 1]

```
# TO-DO: Normalize the data
X = X/255.0
```

Set $y_i = +1$ for images originally labeled 0, and $y_i = -1$ for images originally labeled 1.

```
# TO-DO: Convert labels
y = np.where(y==0, 1, -1)
```

Split the data randomly into train and test with a ratio of 80:20.

```
# TO-DO: Create function that takes in (X,y) and splits the data by returning (X_train, y_train, )
def split(X,y):
  np.random.seed(69)
  index = np.arange(X.shape[0])
  np.random.shuffle(index)

  s = int(0.8 * X.shape[0])
  train_ind = index[:s]
  test_ind = index[s:]

  X_train, y_train = X[train_ind], y[train_ind]
  X_test, y_test = X[test_ind], y[test_ind]

  return X_train, y_train, X_test, y_test
```

```
# TO-DO: Call your function to perform the Train-Test Split on our data
X_train, y_train, X_test, y_test = split(X,y)
```

Initialize the coefficients $\beta_0^{(1)}, \vec{\beta}$ using a Normal Distribution of mean 0 and variance 1.

For β_1, initialize all d entries to be $N(0, 1)$)

```
# TO-DO: Initialize all d entries to be sampled from a standard normal distribution
d = X.shape[1]
B = np.random.randn(d)
B_0 = np.random.randn()
```

Compute the loss function

$$L(\beta_0^{(1)}, \vec{\beta}) = \frac{1}{m} \sum_{i=1}^{m} \ln(1 + e^{-y_i(\beta_0 + \sum_{j=1}^{d} \beta_j x_{i,j})})$$

where $x_{i,j}$ is the j-th of entry data point $\vec{\mathbf{x}_i}$

```
# TO DO: Helper function to compute loss
def compute_loss(data, labels, B, B_0):
  lin = np.dot(data, B) + B_0
  inner = np.exp(-labels * lin)
  loss = np.mean((np.log(1+inner)))
  return loss
```

Compute the gradients of the loss function

$$\frac{\partial L}{\partial \beta_0} = d\beta_0 = -\frac{1}{m} \sum_{i=1}^{m} \frac{e^{-y_i \cdot (\beta_0 + \vec{\beta}^T \vec{x_i})}}{1 + e^{-y_i(\beta_0 + \vec{\beta}^T \vec{x_i})}} y_i$$

$$\nabla_\beta L = d\vec{\beta} = -\frac{1}{m} \sum_{i=1}^{m} \frac{e^{-y_i \cdot (\beta_0 + \vec{\beta}^T \vec{x_i})}}{1 + e^{-y_i(\beta_0 + \vec{\beta}^T \vec{x_i})}} y_i x_i$$

```
# TO-DO: Helper function to compute loss
def compute_gradients(data, labels, B, B_0):
    lin = np.dot(data, B) + B_0

    inner = np.exp(-labels * lin)
    probs = inner / (1 + inner)

    grad_B_0 = -np.mean(probs * labels)

    grad_B = -np.dot((probs * labels), data) / len(labels)
    return grad_B_0, grad_B
```

Update the parameters using gradient updates from the train set as:

$$\beta_j \leftarrow \beta_j - 0.05 \cdot d\beta_j$$
$$\beta_0 \leftarrow \beta_0 - 0.05 \cdot d\beta_0$$

Repeat the process for 50 iterations. You should save your results for each of the 50 epochs in `accuracy_hist`, `train_loss_hist`, and `test_loss_hist`.

```
lr = 0.05

accuracy_hist = []
train_loss_hist = []
test_loss_hist = []

for epoch in range(50):
    grad_B_0, grad_B = compute_gradients(X_train, y_train, B, B_0)
    B = B - (lr * grad_B)
    B_0 = B_0 -  (lr * grad_B_0)
```

```
    train_loss = compute_loss(X_train, y_train, B, B_0)
    train_predictions = np.sign(np.dot(X_train, B) + B_0)
    train_accuracy = np.mean(train_predictions == y_train)

    test_loss = compute_loss(X_test, y_test, B, B_0)
    test_predictions = np.sign(np.dot(X_test, B) + B_0)
    test_accuracy = np.mean(test_predictions == y_test)

    train_loss_hist.append(train_loss)
    test_loss_hist.append(test_loss)
    accuracy_hist.append(test_accuracy)

print("Training complete!")
print(f"Final training loss: {train_loss_hist[-1]}")
print(f"Final test loss: {test_loss_hist[-1]}")
print(f"Final testing accuracy: {accuracy_hist[-1]}")
```

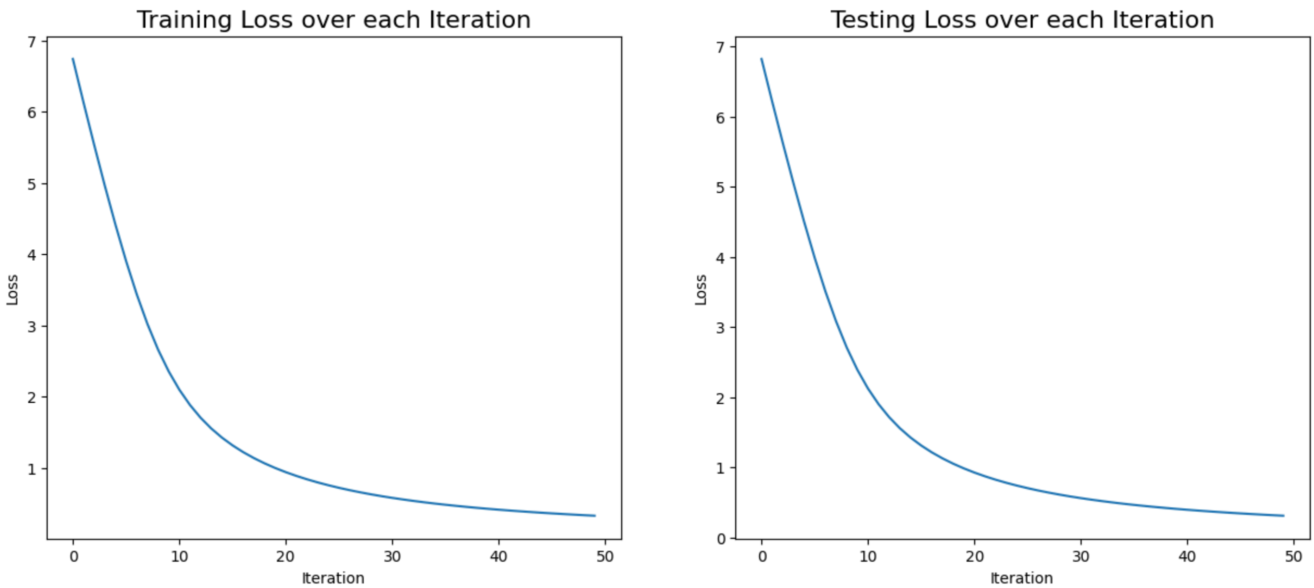
```
Training complete!
Final training loss: 0.32915255073808225
Final test loss: 0.3090091609857607
Final testing accuracy: 0.881935047361299
```

Plot your training and testing loss curves side-by-side below by running the plotting code given below. You don't need to modify these two cells.

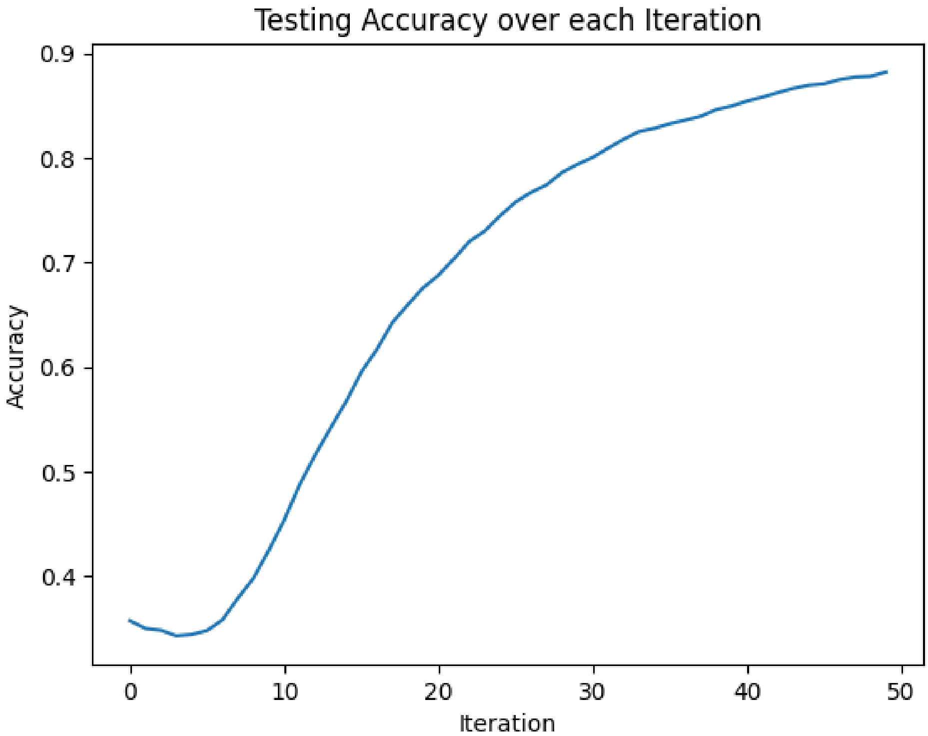
```
fig = plt.figure(figsize = (15,6))

# Plot Loss Function
ax1 = fig.add_subplot(121)
ax1.plot(train_loss_hist);
ax1.set_title("Training Loss over each Iteration", fontsize = 16);
ax1.set_xlabel("Iteration");
ax1.set_ylabel("Loss");

# Plot Accuracy Function
ax2 = fig.add_subplot(122)
ax2.plot(test_loss_hist);
ax2.set_title("Testing Loss over each Iteration", fontsize = 16);
ax2.set_xlabel("Iteration");
ax2.set_ylabel("Loss");
```



```
# Plot your accuracy curve below by running the template code below
plt.plot(accuracy_hist);
plt.title("Testing Accuracy over each Iteration");
plt.xlabel("Iteration");
plt.ylabel("Accuracy");
```



Congratulations!!! Now convert this to `.pdf`!

You're done with all the coding questions! Now convert this `.ipynb` into a `.pdf` neatly to prevent cells and figures from splitting across pages and getting weirdly truncated by following these steps:

1. Download your Colab notebook as an `.ipynb` file to save locally on your computer
2. Open it locally via Jupyter Notebook
3. Go to "print preview" mode
4. Print to `.pdf` from there.
5. Merge this PDF with your handwritten notes. You can use https://smallpdf.com/merge-pdf

Leave enough time to ask for help if you're stuck so you don't get hit with the Late Penalty!