Assignment 02

20142921 SengHyun Lee

2019.10.03

Binary Classification based on Logistic Regression

import library & plot functions

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
import time

import torch
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms
import torchvision
```

In [2]:

```
def output_plot(g1, g2, title, color, label, legend):
    plt.title(title)
    plt.plot(np.arange(1, len(g1) + 1), g1, color=color[0], alpha=0.5, label=label[0])
    plt.plot(np.arange(1, len(g2) + 1), g2, color=color[1], alpha=0.5, label=label[1])
    plt.legend(loc=legend)
    plt.show()


def output_frame_plot(tloss, vloss, tacc, vacc):
    print("           |   loss    |  accuracy  |")
    print("----------------------------------------")
    print("training   |   %.2f    |    %.2f    |" % (tloss, tacc))
    print("----------------------------------------")
    print("validation |   %.2f    |    %.2f    |" % (vloss, vacc))
    print("----------------------------------------")
```

Declare the constants

In [3]:

```
IMAGE_WIDTH = 100
IMAGE_HEIGHT = 100
IMAGE_CHANNEL = 1
DIMENSION = IMAGE_CHANNEL * IMAGE_HEIGHT * IMAGE_WIDTH
```

Load train & validation datasets

- batch size = 3
- number of workers = 1 (main process + worker1)
- number of epoch = 1

In [4]:

```
def pre_process(batch_size=3, num_workers=1):
    transform = transforms.Compose([  # transforms.Resize((256,256)),
        transforms.Grayscale(),
        # the code transforms.Graysclae() is for changing the size [3,100,100] to [1, 100,␣
 ↪100] (notice : [channel, height, width] )
        transforms.ToTensor(), ])

    # train_data_path = 'relative path of training data set'
    train_data_path = './horse-or-human/train'
    trainset = torchvision.datasets.ImageFolder(root=train_data_path, transform=transform)
    # change the valuse of batch_size, num_workers for your program
    # if shuffle=True, the data reshuffled at every epoch
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,␣
 ↪shuffle=False, num_workers=num_workers)

    validation_data_path = './horse-or-human/validation'
    valset = torchvision.datasets.ImageFolder(root=validation_data_path,␣
 ↪transform=transform)
    # change the valuse of batch_size, num_workers for your program
    valloader = torch.utils.data.DataLoader(valset, batch_size=batch_size, shuffle=False,␣
 ↪num_workers=num_workers)

    train_data = np.empty((DIMENSION, 0))
    validation_data = np.empty((DIMENSION, 0))

    train_label = np.array([])
    validation_label = np.array([])

    for i, data in enumerate(trainloader):
        # inputs is the image
        # labels is the class of the image
        inputs, labels = data

        # if you don't change the image size, it will be [batch_size, 1, 100, 100]

        # [batch_size, 1, height, width] => [ width * height * channel, batch_size ]
        x = np.array(inputs).transpose((2, 3, 0, 1)).reshape((DIMENSION, len(labels)))
        train_data = np.concatenate((train_data, x), axis=1)
        train_label = np.concatenate((train_label, np.array(labels)))

    # load validation images of the batch size for every iteration
    for i, data in enumerate(valloader):
        # inputs is the image
        # labels is the class of the image
```

```
        inputs, labels = data

        # [batch_size, 1, height, width] => [ width * height * channel, batch_size ]
        x = np.array(inputs).transpose((2, 3, 0, 1)).reshape((DIMENSION, len(labels)))
        validation_data = np.concatenate((validation_data, x), axis=1)
        validation_label = np.concatenate((validation_label, np.array(labels)))

    return train_data, validation_data, train_label, validation_label


t_data, v_data, t_label, v_label = pre_process(batch_size=3, num_workers=1)
```

Implements of binary classificiaton

- $y' = \sigma(z)$ where $z = w^T x + b$ and $\sigma(z) = \frac{1}{1+\exp(-z)}$
- (Learning Rate) $\alpha = 0.002$
- (Epsilon) $\rceil = 10^{-6}$
- (Cross-Entropy) $f(y', y) = -y \log y' - (1-y)\log(1-y')$
- (Loss function) $\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} f_i(y'_i, y_i)$
- Also, Partial differential derivative of Loss function is

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{1}{n}\sum_{x} x_i(\sigma(z) - y_i)$$

In [5]:

```
def binary_classify(train_data, validation_data, train_label, validation_label):

    learning_rate = 0.002
    epsilon = 10e-6

    w = np.zeros(IMAGE_WIDTH * IMAGE_HEIGHT + 1)  # model parameters with bias

    train_losses = []
    test_losses = []
    train_accuracies = []
    test_accuracies = []
    elapsed_times = []

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

    def distance(prob, ans):
        return -(np.nan_to_num(ans * np.log(prob)) + np.nan_to_num((1 - ans) * np.log(1 -
 ↪prob)))

    def loss(prob, ans):
        return (1 / len(ans)) * np.nan_to_num(np.sum(distance(prob, ans)))
```

```python
    def accuracy(prob, ans):
        arr = np.array(list(map(lambda x: 1 if x > 0.5 else 0, prob)))
        arr = list(filter(lambda x: x == 0, arr - ans))
        return len(arr) / len(ans)

    def dw(x, z):
        return (1 / x.shape[1]) * np.sum(x * (sigmoid(z) - train_label), axis=1)

    def iterate():
        p_train_loss = 0
        nonlocal w, train_losses, test_losses, train_accuracies, test_accuracies, elapsed_times

        train_data_with_bias = np.concatenate((train_data, np.ones((1, train_data.shape[1]))))
        validation_data_with_bias = np.concatenate((validation_data, np.ones((1, validation_data.shape[1]))))

        while True:
            start_time = time.time()

            train_z = np.dot(w.T, train_data_with_bias)
            test_z = np.dot(w.T, validation_data_with_bias)

            w = w - (learning_rate * dw(train_data_with_bias, train_z))

            end_time = time.time()

            n_train_loss = loss(sigmoid(train_z), train_label)
            n_test_loss = loss(sigmoid(test_z), validation_label)

            n_train_acc = accuracy(sigmoid(train_z), train_label)
            n_test_acc = accuracy(sigmoid(test_z), validation_label)

            train_losses.append(n_train_loss)
            test_losses.append(n_test_loss)
            train_accuracies.append(n_train_acc)
            test_accuracies.append(n_test_acc)
            elapsed_times.append(end_time-start_time)

            if abs(p_train_loss - n_train_loss) < epsilon:
                break
            else:
                p_train_loss = n_train_loss
                continue

    iterate()

    return train_losses, test_losses, train_accuracies, test_accuracies, elapsed_times
```

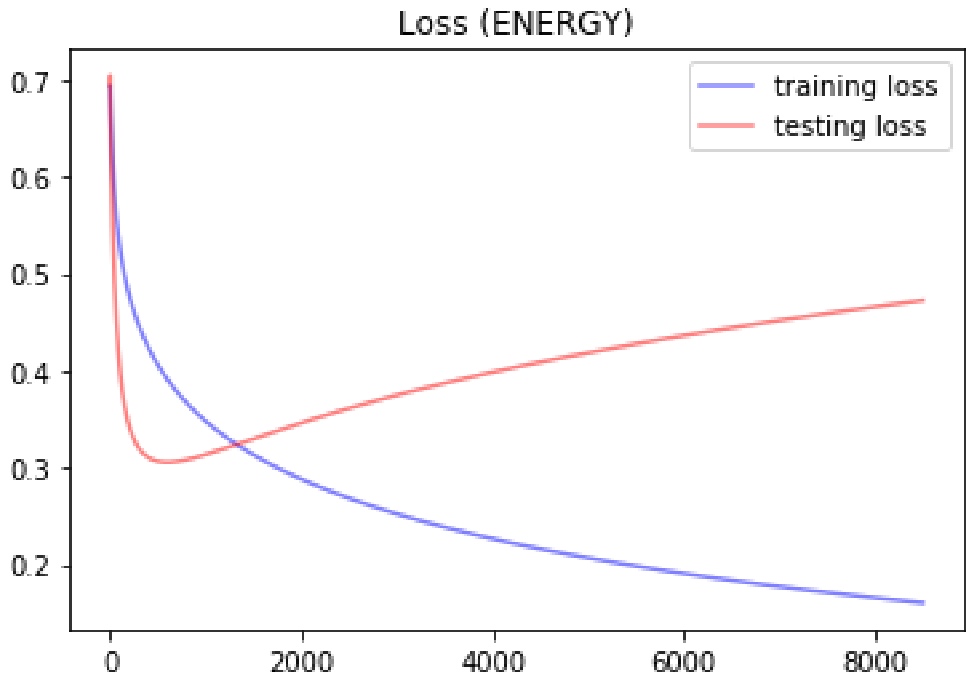
```
train_loss, test_loss, train_acc, test_acc, elapsed_time = binary_classify(t_data, v_data,␣
↪t_label, v_label)
```

Plot the learning curves

loss curve

In [6]:

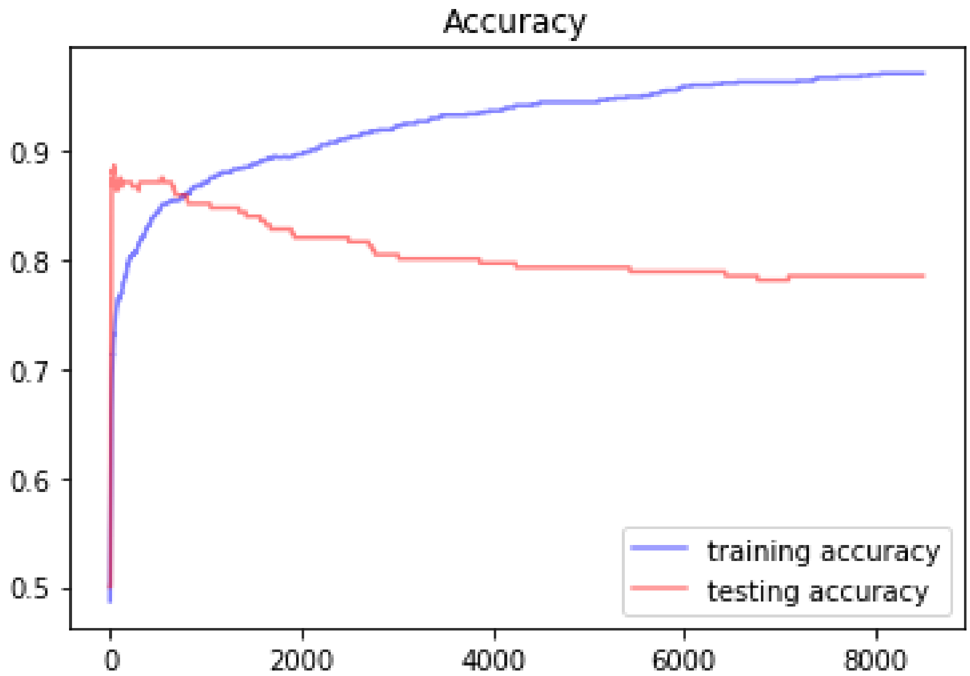
```
output_plot(train_loss, test_loss,
            title="Loss (ENERGY)", color=('blue', 'red'),
            label=('training loss', 'testing loss'), legend='upper right')
```



accuracy curve

In [7]:

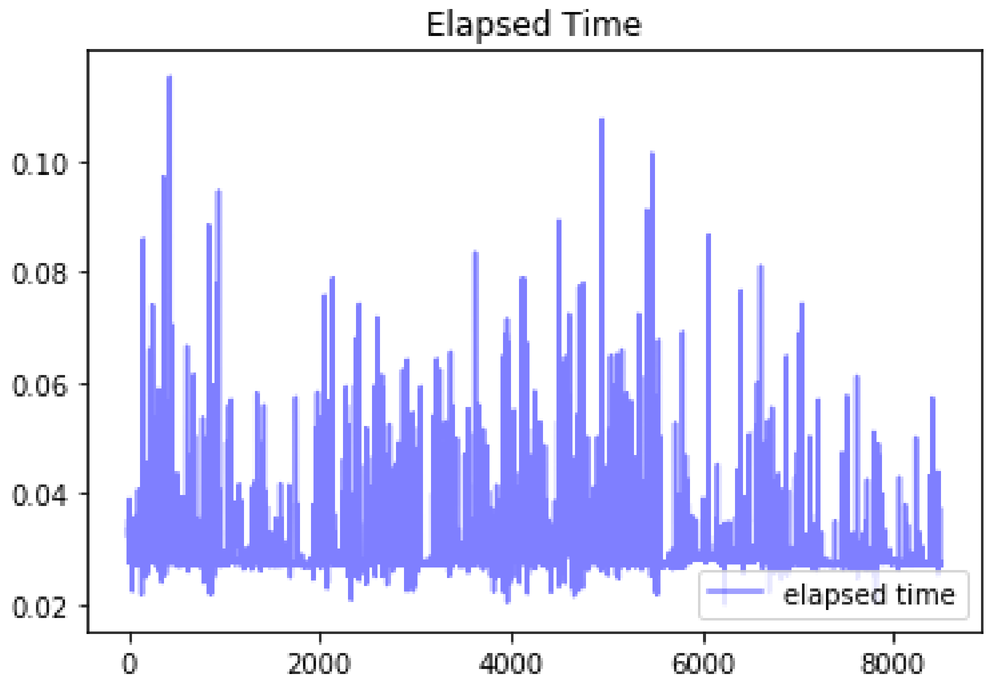
```
output_plot(train_acc, test_acc,
            title="Accuracy", color=('blue', 'red'),
            label=('training accuracy', 'testing accuracy'), legend='lower right')
```



curve of elapsed time

In [8]:

```
output_plot(elapsed_time, [],
            title="Elapsed Time", color=('blue', 'blue'),
            label=('elapsed time', ''), legend='lower right')
```



final accuracy and loss

In [9]:

```
output_frame_plot(train_loss[-1], test_loss[-1], train_acc[-1], test_acc[-1])
```

```
           |   loss    |  accuracy  |
---------------------------------------
training   |   0.16    |    0.97    |
---------------------------------------
validation |   0.47    |    0.79    |
---------------------------------------
```