Maching Learning Project02

October 1, 2019

1 Build a binary classifier for human versus horse

20166450

- This is given by Professor

[229]:
```
import torch
from torch.utils.data import Dataset, DataLoader
import torchvision.transforms as transforms
from torch.autograd import Variable
import torchvision
import os
import sys

from scipy.special import xlogy

import matplotlib.pyplot as plt
import numpy as np


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 = 'C:\\ ␣
↪ \\MachineLearningProject\\horse-or-human\\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=1,␣
↪shuffle=False, num_workers=1)
```

```
validation_data_path = 'C:\\ 
↪ \\MachineLearningProject\\horse-or-human\\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=1, shuffle=False,
↪num_workers=1)
```

- This is given by Professor

[230]:
```
train_labels=np.zeros(1027)
test_labels=np.zeros(256)

train_datas=np.zeros((1027,10001))
test_datas=np.zeros((256,10001))



for epoch in range(1):
    sum=0
    # load training images of the batch size for every iteration
    for i, data in enumerate(trainloader):

        inputs, labels = data
        train_labels[i]=int(labels)
        reinputs=inputs.reshape(10000)
        reinputs=np.array(reinputs)
        reinputs=np.hstack((reinputs,1))
        train_datas[i]=reinputs

    train_datas=train_datas.T

    for i, data in enumerate(valloader):
        sum+=1
        inputs, labels = data
        test_labels[i]=int(labels)
        reinputs=inputs.reshape(10000)
        reinputs=np.array(reinputs)
        reinputs=np.hstack((reinputs,1))
        test_datas[i]=reinputs

    test_datas=test_datas.T
```

- for calculating Accuracy of labels

[240]:
```
def accuracy_func(h_,label):
    label_result=np.zeros(len(h_))
    correct=0

    for i in range(len(label)):
        if(h_[i]<0.5):
            label_result[i]=0
        elif(h_[i]>=0.5):
            label_result[i]=1

        if(label_result[i]==label[i]):
            correct+=1
    total= correct/len(label)

    return total

np.set_printoptions(threshold=sys.maxsize)
```

- set Array and epoch number.

[315]:
```
NUM_EPOCH=4000

total_loss=np.zeros(NUM_EPOCH)
total_loss_test=np.zeros(NUM_EPOCH)

accuracy=np.zeros(NUM_EPOCH)
accuracy_test=np.zeros(NUM_EPOCH)

known_train=np.zeros((10001))
known_test=np.zeros((10001))
```

- Optimization
- Vertorizing Logistic Regression'c gradient Computation

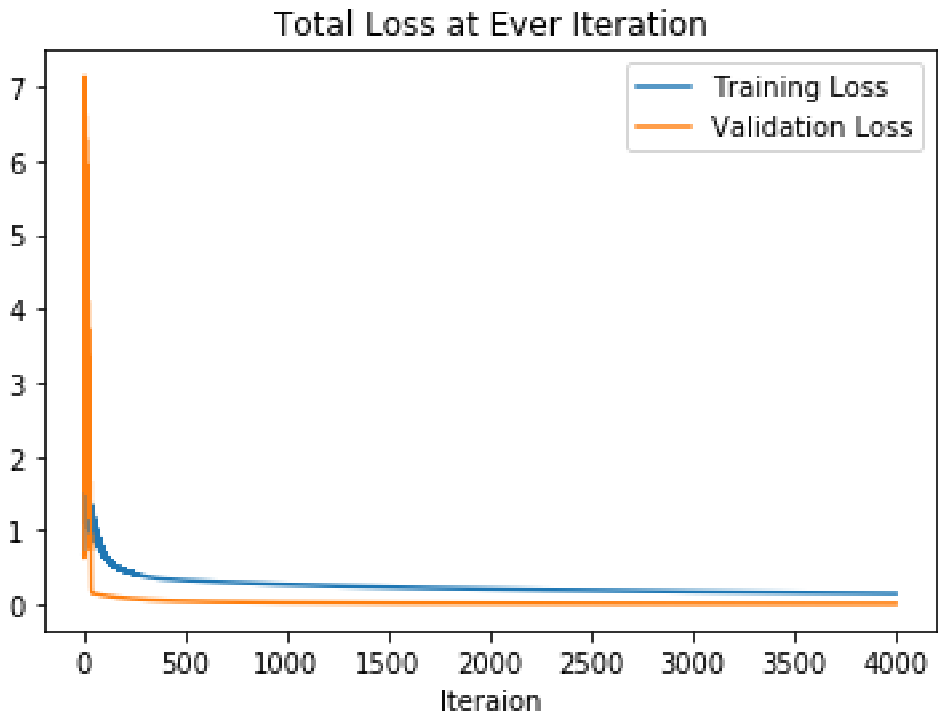
[316]:
```
l_rate=0.005

delta = 1e-70
for i in range(NUM_EPOCH):
    #Vectorizing Logistic Regression for train_set
    L=0
    h=0
    j=0
    z=known_train@train_datas
    h=1.0/(1+np.exp(-z))
    j=-(xlogy(train_labels,h+delta)+xlogy(1-train_labels,1-h+delta)).sum()/1027
    L=h-train_labels
```

```
    dw=train_datas@L
    dw=dw/1027
    known_train-=l_rate*dw
    total_loss[i]=j
#     print("train=",j)

    accuracy[i]=accuracy_func(h,train_labels)
  #  print(j)


    #Vectorizing Logistic Regression for test_set
    L_v=0
    h_v=0
    j_v=0
    z_v=known_test@test_datas
    h_v=1.0/(1.0+np.exp(-z_v))
    j_v=-(xlogy(test_labels,h_v+delta)+xlogy(1-test_labels,1-h_v+delta)).sum()/
 ↪256
    L_v=h_v-test_labels
    dw_v=test_datas@L_v
    dw_v=dw_v/256
    known_test-=l_rate*dw_v
    total_loss_test[i]=j_v
 #   print("test=",j_v)

    accuracy_test[i]=accuracy_func(h_v,test_labels)
```

- Plot the loss of Train and Validation at every iteration
- I divdied Loss to number of datasets in order to compare

[317]:
```
plt.plot(total_loss,label='Training Loss')
plt.plot(total_loss_test,label='Validation Loss')
plt.legend(loc='upper right')
plt.title("Total Loss at Every Iteration")
plt.xlabel("Iteraion")
```

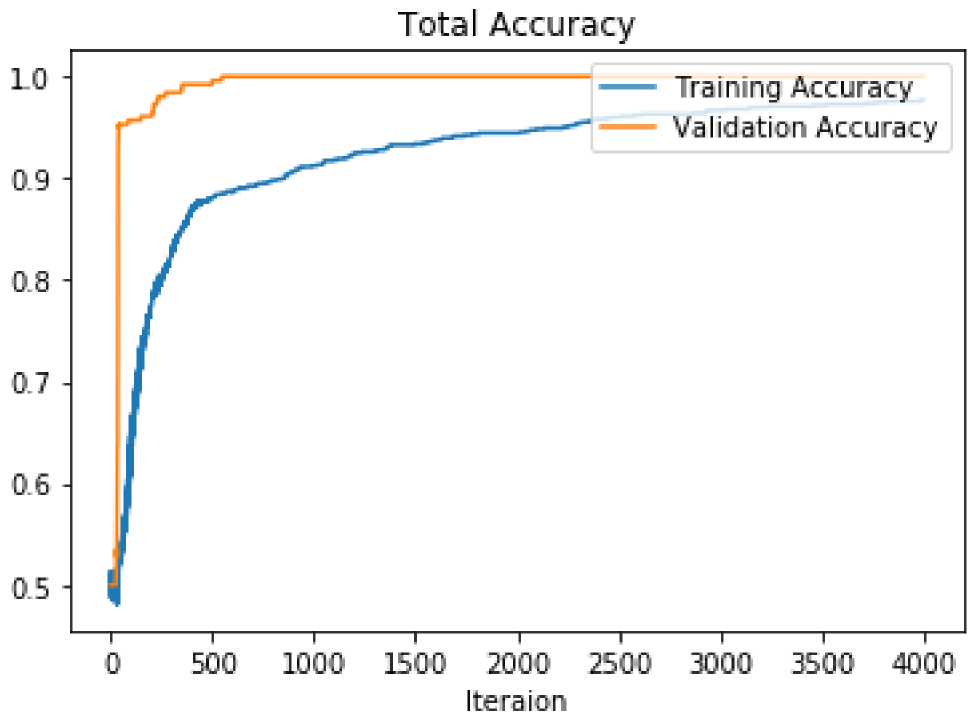
[317]:
```
Text(0.5, 0, 'Iteraion')
```



- Plot the Accuracy of Train and Validation

[318]:
```
plt.plot(accuracy,label='Training Accuracy')
plt.plot(accuracy_test,label='Validation Accuracy')
plt.legend(loc='upper right')
plt.title("Total Accuracy")
plt.xlabel("Iteraion")
```

[318]:
```
Text(0.5, 0, 'Iteraion')
```



- Present the table for the final accuracy and loss with training and validation datasets
- Accuracy multiply 100 to present as a persent(%)

[319]:
```
print("""
+-----------------------+-----------------------+---------------------+
|       Data set        |         Loss          |      Accuracy       |
+-----------------------+-----------------------+---------------------+
|       Training        |        %6.3f          |      %6.2f %%      |
+-----------------------+-----------------------+---------------------+
|      Validation       |        %6.3f          |      %6.2f %%      |
+-----------------------+-----------------------+---------------------+"""
%(total_loss[3998],accuracy[3998]*100,total_loss_test[3998],accuracy_test[3998]*100))
```

```
+-----------------------+-----------------------+---------------------+
|       Data set        |         Loss          |      Accuracy       |
+-----------------------+-----------------------+---------------------+
|       Training        |         0.147          |       97.76 %      |
+-----------------------+-----------------------+---------------------+
|      Validation       |         0.007          |      100.00 %      |
+-----------------------+-----------------------+---------------------+
```