CS547 HW2

Yue Cui (yuecui2)
Gaoyu Liu (gliu18)

Colab link:
https://colab.research.google.com/github/052D/CS547_SP2021/blob/main/HW2/CS547_HW2_Group32.ipynb?authuser=1

Problem 1

$$H(p', p) = p' \ln \frac{p'}{p} + (1 - p') \ln \frac{1 - p'}{1 - p}$$

Take the derivative of $H(p', p)$ w.r.t p':

$$\frac{\partial H}{\partial p'} = \ln(\frac{p'}{p}) + p' \cdot \frac{p'}{p} \cdot \frac{1}{p} - \ln \frac{1 - p'}{1 - p} + (1 - p') \cdot \frac{1 - p}{1 - p'} \cdot \frac{-1}{1 - p}$$
$$\frac{\partial H}{\partial p'} = \ln \frac{p'}{p} - \ln \frac{1 - p'}{1 - p}$$

Now, compute the second order derivitaive of $H(p', p)$ w.r.t p':

$$\frac{\partial^2 H}{\partial p'^2} = \frac{p}{p'} \cdot \frac{1}{p} - \frac{1 - p}{1 - p'} \cdot \frac{-1}{1 - p}$$
$$= \frac{1}{p'} + \frac{1}{1 - p'}$$

Since $p' \in (0, 1)$, $p' > 0$ and $1 - p' > 0$. Finally, $\frac{\partial^2 H}{\partial p'^2} > 0$.

Since the second order derivitaive of $H(p', p)$ w.r.t p' is strictly positive, $p' \mapsto H(p', p)$ is **convex** for $p \in (0, 1)$.

Problem 2

Take the first derivative of $f(\theta) = \theta p' - \ln\{pe^{\theta} + (1 - p)\}$ with respect to θ, we have:

$$\frac{df(\theta)}{d\theta} = p' - \frac{pe^{\theta}}{pe^{\theta} + (1 - p)}$$

To find θ that maximize $f(\theta)$, we set $\frac{df(\theta)}{d\theta} = 0$, hence:

$$p' - \frac{pe^{\theta}}{pe^{\theta} + (1 - p)} = 0$$
$$p'pe^{\theta} + p'(1 - p) = pe^{\theta}$$
$$(p'p - p)e^{\theta} = -p'(1 - p)$$
$$e^{\theta} = \frac{p' - pp'}{p - p'p}$$
$$\theta = \ln \frac{p' - pp'}{p - pp'}$$

Now substitute $\theta = \ln \frac{p' - pp'}{p - p'p}$ back to $f(\theta)$:

$$\max_{\theta \in R} \{\theta p' - \ln\{pe^{\theta} + (1 - p)\}\} = p' \ln \frac{p' - pp'}{p - pp'} - \ln\{pe^{\ln \frac{p' - pp'}{p - pp'}} + (1 - p)\}$$
$$= p' \ln \frac{p'(1 - p)}{p(1 - p')} - \ln\{p\frac{p' - pp'}{p - pp'} + (1 - p)\}$$
$$= p'(\ln \frac{p'}{p} + \ln \frac{1 - p}{1 - p'}) - \ln\{\frac{p(p' - pp') + (p - pp')(1 - p)}{p - pp'}\}$$
$$= p' \ln \frac{p'}{p} + p' \ln \frac{1 - p}{1 - p'} - \ln \frac{1 - p}{1 - p'}$$
$$= p' \ln \frac{p'}{p} + (1 - p') \ln \frac{1 - p'}{1 - p}$$

The result gives the **entropy function**. Therefore, it is verified that the Legendre-Fenchel transform of the logarithm of the moment generating function of a Bernoulli random variable $\ln\{pe^{\theta}+(1-p)\}$ indeed is the entropy function.

Problem 3

In [15…

```
# What version of Python do you have?
import sys

import torch
import pandas as pd
import sklearn as sk
import numpy as np
import matplotlib
#%matplotlib notebook
import matplotlib.pyplot as plt


print(f"PyTorch Version: {torch.__version__}")
print()
print(f"Python {sys.version}")
print(f"Pandas {pd.__version__}")
print(f"Scikit-Learn {sk.__version__}")
print("GPU is", "available" if torch.cuda.is_available() else "NOT AVAILABLE")
```

```
PyTorch Version: 1.7.1

Python 3.7.9 (default, Aug 31 2020, 17:10:11) [MSC v.1916 64 bit (AMD64)]
Pandas 1.2.1
Scikit-Learn 0.23.2
GPU is available
```

Generate the data.

In [32…

```
# generate 200 Gausian points
torch.manual_seed(1)
N = 200

data_input = torch.Tensor(N, 1).normal_(mean = 0., std = 1.)
#print(data_input)

labels_input = (data_input > 0).float()
#print(labels_input)

# check for GPU
if torch.cuda.is_available():
    data_input = data_input.cuda()
    labels_input = labels_input.cuda()
```

(1)

Define the class **LogisticRegression** using Pytorch. The Sigmoid function

$$S_{m,b}(X)=\frac{e^{mX+b}}{1+e^{mX+b}}$$

is used and a entropy function for binary classification problem is adopted, namely, **torch.nn.BCEWithLogitsLoss**.

For each training sample, the input X is a scalar with dimension 1, the output z is also a scalar with dimension of 1.

The learning rate is set to be 0.1.

In [13…

```
class LogisticRegression(torch.nn.Module):
    def __init__(self, input_size, num_classes):
        super(LogisticRegression, self).__init__()
        # https://pytorch.org/docs/stable/generated/torch.nn.Linear.html
        self.linear = torch.nn.Linear(input_size, num_classes)

    def forward(self, x):
```

```
        out = self.linear(x)
        return out



# flip n*2 points as needed (for (2) and (3))
def flip_n(data, labels, n):
    if n == 0:
        return labels.clone()
    # get the assending sorted indices
    sort_idx = data.argsort(dim=0)
    # get the total number of negative numbers
    num_neg = sum(data < 0)
    # get the indices to flip the labels
    flip_idx_neg = sort_idx[num_neg - n: num_neg]
    flip_idx_pos = sort_idx[num_neg: num_neg + n]
    # carry out the label flipping
    labels_res = labels.clone()
    labels_res[flip_idx_neg] = 1.
    labels_res[flip_idx_pos] = 0.
    # return the result
    return labels_res



def Carryout_logi_regression(data_input,
                             labels_input,
                             n=0,
                             lr=.1, max_iter=10000,
                             ):
    # process the labels
    labels_n = flip_n(data_input, labels_input, n)

    # instantiate the model object
    input_dim = 1  # data_raw.size()[0]
    num_classes = 1

    model = LogisticRegression(input_dim, num_classes)
    if torch.cuda.is_available():
        model.cuda()

    #Loss = torch.nn.CrossEntropyLoss()
    # https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html
    Loss = torch.nn.BCEWithLogitsLoss(reduction='mean')

    learningRate = lr
    optimizer = torch.optim.SGD(model.parameters(), lr=learningRate)

    for itr in range(max_iter):
        # Clear gradient buffers because we don't want any gradient from previous epoch to carry forward,
        # dont want to cummulate gradients
        optimizer.zero_grad()

        # get output from the model, given the inputs
        outputs = model(data_input)
        # print(outputs.size())
        # print(labels_input.size())

        # get loss for the predicted output
        lossvalue = Loss(outputs, labels_n)

        # get gradients w.r.t to parameters
        lossvalue.backward()
        # print(model.linear.weight.grad.item(),model.linear.bias.grad.item())

        # update parameters
        optimizer.step()
        if itr % 10000 == 0:
            print("iteration {}: loss={:.5f}, width={:.5f} ".format(itr, lossvalue.item(),
                                                                    1/model.linear.weight.item()))
    #
    print(f'The width of the transition layer for the model with {n} points\n',
```

```
        f'flipped on each side of 0 is {1/model.linear.weight.item():.5f}')
    print(f'================================================\n')

    return 1/model.linear.weight.item()
```

Now carry out the logistic regression.

In [14…
```
width_0 = Carryout_logi_regression(data_input,
                                   labels_input,
                                   n=0,
                                   lr=0.1, max_iter=100000,
                                   )
```

```
iteration 0: loss=0.45614, width=1.14401
iteration 10000: loss=0.05049, width=0.08269
iteration 20000: loss=0.03935, width=0.06494
iteration 30000: loss=0.03399, width=0.05647
iteration 40000: loss=0.03065, width=0.05120
iteration 50000: loss=0.02831, width=0.04748
iteration 60000: loss=0.02654, width=0.04466
iteration 70000: loss=0.02514, width=0.04242
iteration 80000: loss=0.02399, width=0.04058
iteration 90000: loss=0.02303, width=0.03903
The width of the transition layer for the model with 0 points
 flipped on each side of 0 is 0.03770
================================================
```

The width of the transition layer for the perfect training data (no wrong labels) after 100000 iterations with a learning rate of 0.1 is **0.03769566**.

In [16…
```
width_result = np.array([])
width_result = np.append(width_result, width_0)
#display(width_result)
```

(2)

In [14…
```
width_5 = Carryout_logi_regression(data_input,
                                   labels_input,
                                   n=5,
                                   lr=.1, max_iter=100000,
                                   )
```

```
iteration 0: loss=0.68865, width=20.24849
iteration 10000: loss=0.07547, width=0.09068
iteration 20000: loss=0.06993, width=0.07495
iteration 30000: loss=0.06804, width=0.06800
iteration 40000: loss=0.06718, width=0.06399
iteration 50000: loss=0.06673, width=0.06138
iteration 60000: loss=0.06649, width=0.05957
iteration 70000: loss=0.06634, width=0.05826
iteration 80000: loss=0.06626, width=0.05728
iteration 90000: loss=0.06620, width=0.05653
The width of the transition layer for the model with 5 points
 flipped on each side of 0 is 0.05594
================================================
```

The width of the transition layer for the training data containg 5 points on each side of the origin flipped to wrong labels after 100000 iterations with a learning rate of 0.1 is **0.05594289**.

In [16…
```
width_result = np.append(width_result, width_5)
#display(width_result)
```

(3)

In [14…
```
for item in [15, 20, 25, 30, 35]:
```

```
    width_ = Carryout_logi_regression(data_input,
                                      labels_input,
                                      n=item,
                                      lr=.001, max_iter=100000,
                                      )
    #
    width_result = np.append(width_result, width_)
```

```
iteration 0: loss=0.96724, width=-1.78957
iteration 10000: loss=0.31989, width=0.59803
iteration 20000: loss=0.26009, width=0.41097
iteration 30000: loss=0.23664, width=0.34307
iteration 40000: loss=0.22399, width=0.30587
iteration 50000: loss=0.21612, width=0.28173
iteration 60000: loss=0.21078, width=0.26453
iteration 70000: loss=0.20696, width=0.25154
iteration 80000: loss=0.20412, width=0.24132
iteration 90000: loss=0.20194, width=0.23302
The width of the transition layer for the model with 15 points
 flipped on each side of 0 is 0.22614
================================================

iteration 0: loss=0.60948, width=3.96420
iteration 10000: loss=0.32823, width=0.55989
iteration 20000: loss=0.28800, width=0.41470
iteration 30000: loss=0.27222, width=0.35635
iteration 40000: loss=0.26411, width=0.32363
iteration 50000: loss=0.25936, width=0.30235
iteration 60000: loss=0.25635, width=0.28732
iteration 70000: loss=0.25435, width=0.27610
iteration 80000: loss=0.25296, width=0.26743
iteration 90000: loss=0.25198, width=0.26052
The width of the transition layer for the model with 20 points
 flipped on each side of 0 is 0.25491
================================================

iteration 0: loss=0.94369, width=-1.96092
iteration 10000: loss=0.37282, width=0.64915
iteration 20000: loss=0.33087, width=0.46021
iteration 30000: loss=0.31761, width=0.39468
iteration 40000: loss=0.31177, width=0.36050
iteration 50000: loss=0.30879, width=0.33947
iteration 60000: loss=0.30714, width=0.32532
iteration 70000: loss=0.30618, width=0.31527
iteration 80000: loss=0.30559, width=0.30786
iteration 90000: loss=0.30523, width=0.30225
The width of the transition layer for the model with 25 points
 flipped on each side of 0 is 0.29793
================================================

iteration 0: loss=1.21433, width=-1.20772
iteration 10000: loss=0.41800, width=0.72365
iteration 20000: loss=0.38264, width=0.51319
iteration 30000: loss=0.37413, width=0.44730
iteration 40000: loss=0.37114, width=0.41544
iteration 50000: loss=0.36991, width=0.39730
iteration 60000: loss=0.36937, width=0.38606
iteration 70000: loss=0.36912, width=0.37875
iteration 80000: loss=0.36900, width=0.37385
iteration 90000: loss=0.36894, width=0.37049
The width of the transition layer for the model with 30 points
 flipped on each side of 0 is 0.36818
================================================

iteration 0: loss=0.74995, width=-5.87976
iteration 10000: loss=0.45199, width=0.73994
iteration 20000: loss=0.43374, width=0.56652
iteration 30000: loss=0.43003, width=0.51176
iteration 40000: loss=0.42903, width=0.48717
iteration 50000: loss=0.42872, width=0.47456
iteration 60000: loss=0.42862, width=0.46767
iteration 70000: loss=0.42859, width=0.46378
```

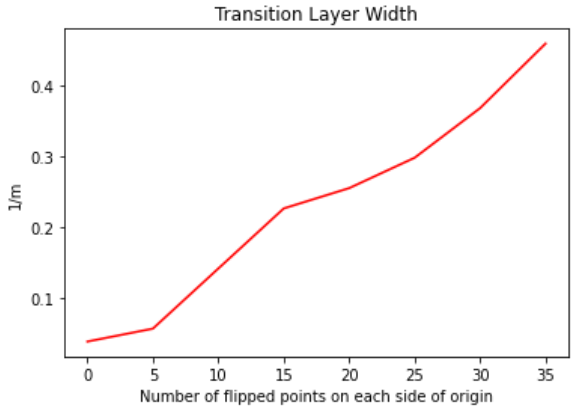
```
iteration 80000: loss=0.42858, width=0.46155
iteration 90000: loss=0.42857, width=0.46027
The width of the transition layer for the model with 35 points
 flipped on each side of 0 is 0.45954
================================================
```

The transition layer width vs number of points with the 'wrong' label on each side of the origin is ploted below. As the number of 'wrong' label points increases, the trainsition layer width increases, indicating that the training data with the wrong labels make the trained logistic regression model have extended transition layer in which the prediction accuracy may decrease.

In [16…

```
plt.figure()
plt.plot(np.delete(np.arange(0, 40, 5), 2), width_result,
         label="Width",
         color="red")

title = []
title.append("Transition Layer Width")
plt.title("\n".join(title))
plt.xlabel("Number of flipped points on each side of origin")
plt.ylabel("1/m")
plt.show()
plt.close()
```



In [ ]: