1 [Linear Regression]

2019/0243 서재은

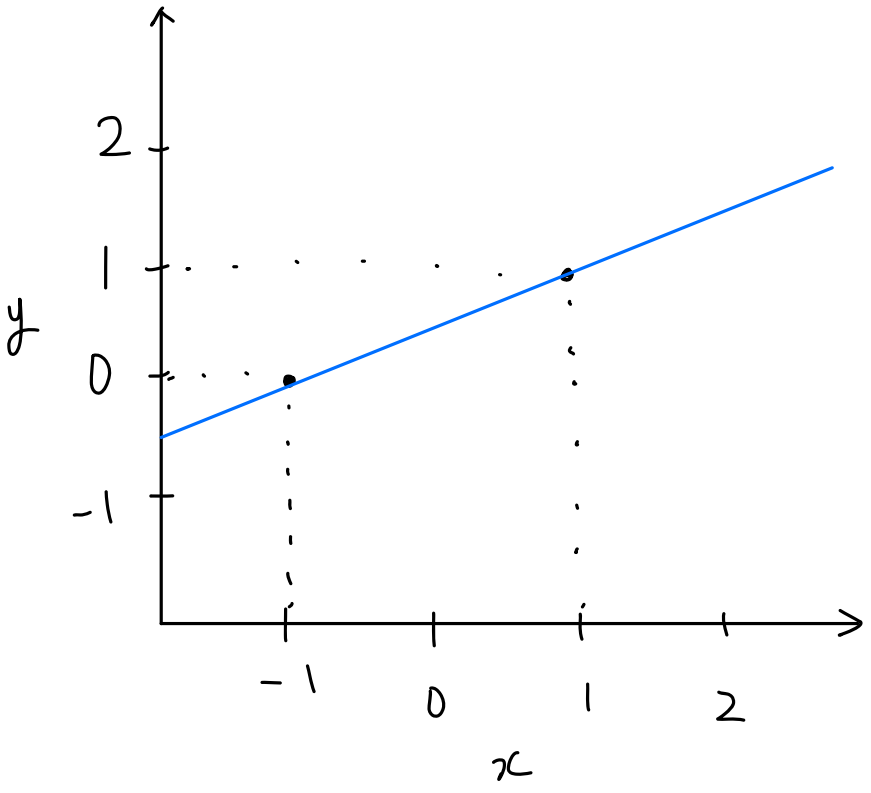
dataset $D = \{(-1, 0), (1, 1)\}$ containing two pairs (x, y) with $x \in \mathbb{R}$ and $y \in \mathbb{R}$

goal : find the parameters $w = \begin{bmatrix} w_1 \\ w_0 \end{bmatrix} \in \mathbb{R}^2$ of a linear regression model $y = w_1 x + w_0$ using

$$\min \frac{1}{2} \sum_{(x,y) \in D} \left(y - w^T \begin{bmatrix} x \\ 1 \end{bmatrix} \right)^2$$

(a)

plot the given dataset & find the optimal w^* by inspection.



$y = w_1 x + w_0$

$0 = -w_1 + w_0 \rightarrow w_0 = w_1$

$1 = w_1 + w_0 \rightarrow w_0 = w_1 = \frac{1}{2}$

$\therefore w^* = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$

(b)

$y \in \mathbb{R}^2$, $X \in \mathbb{R}^{2\times 2}$

goal : $\min_w \frac{1}{2}\|y - Xw\|_2^2 = \min_w \frac{1}{2}\sum_{(x,y)\in D}\left(y - w^T\begin{bmatrix} x \\ 1 \end{bmatrix}\right)^2$

$\downarrow$

$y = w_1 x + w_0$

$\rightarrow \quad y = \begin{bmatrix} y_1 \\ y_0 \end{bmatrix}$ & $X = \begin{bmatrix} x_1 & 1 \\ x_0 & 1 \end{bmatrix}$

So, $y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ & $X = \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}$

(c)

goal : ① derive the general analytical solution for

$$\min_w \frac{1}{2}\|y - Xw\|_2^2$$

② plug in the values for the given dataset D and compute the solution numerically.

① $\frac{1}{2}\|y-Xw\|_2^2 = \frac{1}{2}(y-Xw)^T(y-Xw)$

$$\frac{\partial L}{\partial w} = \frac{\partial\left(\frac{1}{2}(y-Xw)^T(y-Xw)\right)}{\partial w}$$

$$= \frac{\partial\left(\frac{1}{2}(y^T - w^TX^T)(y-Xw)\right)}{\partial w}$$

$$= \frac{\partial\left(\frac{1}{2}(y^Ty - y^TXw - w^TX^Ty + w^TX^TXw)\right)}{\partial w} = 0$$

$$-y^TX - y^TX + w^T(X^TX + X^TX) = 0$$

$$2y^TX = 2w^TX^TX$$

$$X^TXw = X^Ty$$

$$w = (X^TX)^{-1}X^Ty$$

② $X=\begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}$ $y=\begin{bmatrix} 0 \\ 1 \end{bmatrix}$

$$X^TX=\begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}=\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

$$\left[\begin{array}{cc|cc} 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \end{array}\right] \rightarrow \left[\begin{array}{cc|cc} 1 & 0 & 1/2 & 0 \\ 0 & 1 & 0 & 1/2 \end{array}\right] \rightarrow (X^TX)^{-1}=\begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix}$$

$$(X^TX)^{-1}X=\begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix}\begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}=\begin{bmatrix} -1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix}$$

$$\therefore (X^TX)^{-1}X^Ty=\begin{bmatrix} -1/2 & 1/2 \\ 1/2 & 1/2 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix}=\begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}=\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

(4) Linear Regression . py

```python
import torch

x = torch.Tensor([[-1],[1]])
y = torch.Tensor([[0],[1]])
print(x.size())

X = torch.cat((x, torch.ones_like(x)),1)
print(X)
print(X[1,0])
print(torch.matmul(X, y))


#[Your task1]
################################
## Fill in the arguments
res1 = torch.lstsq(y, X)
################################
print("Solution 1:")
print(res1[0])

# Hint:
print(torch.matmul(torch.transpose(X, 0, 1),X))
print(torch.matmul(torch.transpose(X, 0, 1),y))

#[Your task2]
################################
## What should l and r be?
## Dimensions: l (2x2); r (2x1)
l = (torch.matmul(torch.transpose(X, 0, 1), X))
r =    (torch.matmul(torch.transpose(X, 0, 1), y))
################################
res2 = torch.solve(r,l)
print("Solution 2:")
print(res2[0])


#[Your task3]
################################
## What should l and r be?
## Dimensions: l (2x2); r (2x1)
l = (torch.matmul(torch.transpose(X, 0, 1), X))
r = (torch.matmul(torch.transpose(X, 0, 1), y))
################################
res3 = torch.matmul(torch.inverse(l),r)
print("Solution 3:")
print(res3)
```

$\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$

$\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$

$\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$

all the same with my answer 1-(c).

```
Solution 1:
tensor([[0.5000],
        [0.5000]])
tensor([[2., 0.],
        [0., 2.]])
tensor([[1.],
        [1.]])
Solution 2:
tensor([[0.5000],
        [0.5000]])
Solution 3:
tensor([[0.5000],
        [0.5000]])
```

(e)

given a dataset $D' = \{(-2,-2), (-1,0), (1,1), (2,1)\}$ of pairs (x,y) with $x \in \mathbb{R}$ and $y \in \mathbb{R}$

goal: fit a quadratic model $\hat{y} = w_2 x^2 + w_1 x + w_0$ using

$$\min \frac{1}{2} \| y - Xw \|_2^2 .$$

① specify the dimensions of the matrix X and vector y.

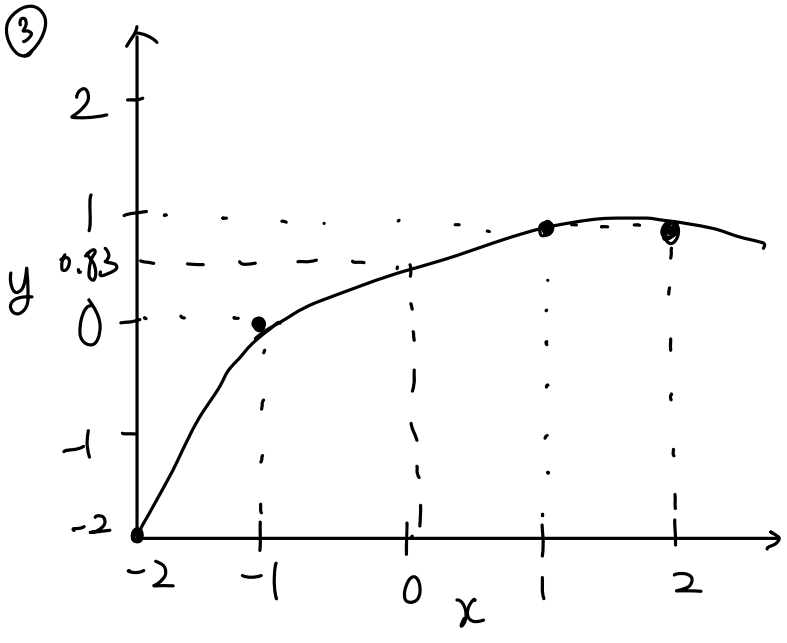
② write down explicitly the matrix and vector using the values in D'

③ Find the optimal solution w^* and draw it together with the dataset into a plot.

① $X \in \mathbb{R}^{4 \times 3}$ $\quad y \in \mathbb{R}^4$

② $X = \begin{bmatrix} 4 & -2 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \\ 4 & 2 & 1 \end{bmatrix}$ $\quad y = \begin{bmatrix} -2 \\ 0 \\ 1 \\ 1 \end{bmatrix}$ $\quad w = \begin{bmatrix} w_2 \\ w_1 \\ w_0 \end{bmatrix}$

③



$$w = \begin{bmatrix} w_2 \\ w_1 \\ w_0 \end{bmatrix} = (X^T X)^{-1} X^T y$$

$$X^T X = \begin{bmatrix} 4 & 1 & 1 & 4 \\ -2 & -1 & 1 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 4 & -2 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \\ 4 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 34 & 0 & 10 \\ 0 & 10 & 0 \\ 10 & 0 & 4 \end{bmatrix}$$

$$(X^T X)^{-1} = \begin{bmatrix} 1/9 & 0 & -5/18 \\ 0 & 1/10 & 0 \\ -5/18 & 0 & 17/18 \end{bmatrix}$$

$$X^T y = \begin{bmatrix} 4 & 1 & 1 & 4 \\ -2 & -1 & 1 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -3 \\ 7 \\ 0 \end{bmatrix}$$

$$(X^T X)^{-1} X^T y = \begin{bmatrix} 1/9 & 0 & -5/18 \\ 0 & 1/10 & 0 \\ -5/18 & 0 & 17/18 \end{bmatrix} \begin{bmatrix} -3 \\ 7 \\ 0 \end{bmatrix} = \begin{bmatrix} -1/3 \\ 7/10 \\ 5/6 \end{bmatrix}$$

$$\therefore w^* = \begin{bmatrix} -1/3 \\ 7/10 \\ 5/6 \end{bmatrix} = \begin{bmatrix} -0.33 \\ 0.7 \\ 0.83 \end{bmatrix}$$

(f)

```
import torch

# [your task1]
##############################
## Specify the matrix X and vector y
X = torch.Tensor([[4, -2, 1], [1, -1, 1], [1, 1, 1], [4, 2, 1]])
y = torch.Tensor([[-2], [0], [1], [1]])
##############################
print(X)
print(y)

# [your task2]
##############################
## Use one of the ways to compute the result, c.f., LinearRegression.py
l = (torch.matmul(torch.transpose(X, 0, 1), X))
r = (torch.matmul(torch.transpose(X, 0, 1), y))
res = torch.matmul(torch.inverse(l), r)
##############################
print("Solution:")
print(res)
```

```
Solution:
tensor([[-0.3333],
        [ 0.7000],
        [ 0.8333]])
```

X = torch.Tensor([[4,-2,1], [1,-1,1], [1,1,1], [4,2,1]])

y = torch.Tensor([[-2], [0], [1], [1]])

same with my answer 1-(e).

2 [Binary Logistic Regression]

dataset $D = \{(-2,-1), (1,1), (2,1)\}$ containing three pairs (x,y), where each $x \in \mathbb{R}$ denotes a real-valued point and a $y \in \{-1, +1\}$ is the points class label.

assumption : the samples in D to be i.i.d. & using maximum likelihood

goal : train a logistic regression model parameterized by $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \in \mathbb{R}^2$:

$$p(y|x) = \frac{1}{1 + \exp\left(-y w^T \begin{bmatrix} x \\ 1 \end{bmatrix}\right)}$$

(a)

goal : minimization problem for the model given in

$$p(y|x) = \frac{1}{1 + \exp\left(-y w^T \begin{bmatrix} x \\ 1 \end{bmatrix}\right)}$$

$$\underset{w}{\arg\min} \sum_n \log\left(1 + \exp\left(-y w^T \begin{bmatrix} x \\ 1 \end{bmatrix}\right)\right)$$

(b)

goal: ① compute the derivative of the negative log-likelihood objective in (a)

② sketch a simple gradient-descent algorithm using pseudo-code

$$① \; L(w) = \sum_n \log\left(1 + \exp\left(-y_n w^T \begin{bmatrix} x_n \\ 1 \end{bmatrix}\right)\right)$$

$$\frac{\partial L}{\partial w} = \sum_n \frac{\exp\left(-y_n w^T \begin{bmatrix} x_n \\ 1 \end{bmatrix}\right)(-y_n)\begin{bmatrix} x_n \\ 1 \end{bmatrix}}{1 + \exp\left(-y_n w^T \begin{bmatrix} x_n \\ 1 \end{bmatrix}\right)}$$

$$= \sum_n \frac{-y_n \cdot \exp\left(-y_n \cdot w^T \begin{bmatrix} x_n \\ 1 \end{bmatrix}\right)\begin{bmatrix} x_n \\ 1 \end{bmatrix}}{1 + \exp\left(-y_n \cdot w^T \begin{bmatrix} x_n \\ 1 \end{bmatrix}\right)}$$

②

$$w_{t+1} \leftarrow w_t - \alpha \nabla_w L(w)$$

(c)

```python
import torch

torch.manual_seed(1)
X = torch.Tensor([[-2, 1, 2],[1, 1, 1]])
y = torch.Tensor([-1, 1, 1])
w = torch.Tensor([[0.1],[0.1]])
alpha = 1

for iter in range(100):
    tmp = torch.exp(torch.matmul(torch.transpose(w,0,1),X)*(-y))

    # [your task]
    ##############################
    ## Use tmp to compute f and g. Instead of summing we average the result, i.e.,
    ## complete only inside torch.mean(...) and don't remove this function
    ## Dimensions: f (scalar); g (2)
    f = torch.mean((torch.log(1 + tmp)))
    g = torch.mean((-y * tmp / (1 + tmp) * X), 1)
    ##############################

    print("Loss: %f; ||g||: %f" % (f, torch.norm(g)))
    g = g.view(-1,1)
    w = w - alpha*g

print(w)
```

```
Loss: 0.010605; ||g||: 0.014524
Loss: 0.010396; ||g||: 0.014235
Loss: 0.010195; ||g||: 0.013957
Loss: 0.010002; ||g||: 0.013690
Loss: 0.009816; ||g||: 0.013433
Loss: 0.009638; ||g||: 0.013186
Loss: 0.009465; ||g||: 0.012948
Loss: 0.009299; ||g||: 0.012718
Loss: 0.009139; ||g||: 0.012497
Loss: 0.008984; ||g||: 0.012283
Loss: 0.008834; ||g||: 0.012076
Loss: 0.008690; ||g||: 0.011876
Loss: 0.008550; ||g||: 0.011683
Loss: 0.008415; ||g||: 0.011496
Loss: 0.008283; ||g||: 0.011315
Loss: 0.008156; ||g||: 0.011140
Loss: 0.008033; ||g||: 0.010970
Loss: 0.007914; ||g||: 0.010806
Loss: 0.007798; ||g||: 0.010646
Loss: 0.007685; ||g||: 0.010491
Loss: 0.007576; ||g||: 0.010340
Loss: 0.007470; ||g||: 0.010194
Loss: 0.007367; ||g||: 0.010052
Loss: 0.007266; ||g||: 0.009914
Loss: 0.007169; ||g||: 0.009779
Loss: 0.007074; ||g||: 0.009648
Loss: 0.006981; ||g||: 0.009521
Loss: 0.006891; ||g||: 0.009397
Loss: 0.006804; ||g||: 0.009277
Loss: 0.006718; ||g||: 0.009159
Loss: 0.006635; ||g||: 0.009044
Loss: 0.006553; ||g||: 0.008932
Loss: 0.006474; ||g||: 0.008823
Loss: 0.006397; ||g||: 0.008717
Loss: 0.006321; ||g||: 0.008613
Loss: 0.006247; ||g||: 0.008512
Loss: 0.006175; ||g||: 0.008413
Loss: 0.006105; ||g||: 0.008316
Loss: 0.006036; ||g||: 0.008222
Loss: 0.005969; ||g||: 0.008129
Loss: 0.005903; ||g||: 0.008039
Loss: 0.005839; ||g||: 0.007951
Loss: 0.005776; ||g||: 0.007865
Loss: 0.005715; ||g||: 0.007780
Loss: 0.005654; ||g||: 0.007697
Loss: 0.005596; ||g||: 0.007617
Loss: 0.005538; ||g||: 0.007537
Loss: 0.005481; ||g||: 0.007460
Loss: 0.005426; ||g||: 0.007384
Loss: 0.005372; ||g||: 0.007309
Loss: 0.005319; ||g||: 0.007237
Loss: 0.005266; ||g||: 0.007165
Loss: 0.005215; ||g||: 0.007095
Loss: 0.005165; ||g||: 0.007026
tensor([[3.3747],
        [1.0860]])
```

$$w^* = \begin{bmatrix} 3.3747 \\ 1.0860 \end{bmatrix}$$

(d)

fourth datapoint (100, 1)

Would this influence the solution of maximum likelihood estimate w much? How about if we had used linear regression to fit D as opposed to logistic regression?

→ Logistic regression is more robust than linear regression when some data is added. It means that the logistic regression is less influenced by outliers. When some data is added to the dataset, the parameters of logistic regression will not be changed a lot. Because that data is already in the correct region of decision boundary. In conclusion, L2-loss can influence the result than logistic-loss.

(e)

```
import torch
import torch.optim as optim

torch.manual_seed(1)
X = torch.Tensor([[-2, 1, 2],[1, 1, 1]])
y = torch.Tensor([-1, 1, 1])
w = torch.Tensor([[0.1],[0.1]])
w.requires_grad = True
alpha = 1

optimizer = optim.SGD([w], lr=alpha)
optimizer.zero_grad()

for iter in range(100):
    tmp = torch.exp(torch.matmul(torch.transpose(w,0,1),X)*(-y))

    # [your task1]
    ##############################
    ## loss is the same as f in LogisticRegression.py
    ## Dimensions: loss (scalar)
    loss = torch.mean((torch.log(1 + tmp)))
    ##############################

    loss.backward()
    print("Loss: %f; ||g||: %f" % (loss, torch.norm(w.grad)))

    # [your task2]
    ##############################
    ## Use two functions within the optimizer instance to perform the update step
    optimizer.step()
    optimizer.zero_grad()
    ##############################


print(w)
```

```
Loss: 0.009816; ||g||: 0.013433
Loss: 0.009638; ||g||: 0.013186
Loss: 0.009465; ||g||: 0.012948
Loss: 0.009299; ||g||: 0.012718
Loss: 0.009139; ||g||: 0.012497
Loss: 0.008984; ||g||: 0.012283
Loss: 0.008834; ||g||: 0.012076
Loss: 0.008690; ||g||: 0.011876
Loss: 0.008550; ||g||: 0.011683
Loss: 0.008415; ||g||: 0.011496
Loss: 0.008283; ||g||: 0.011315
Loss: 0.008156; ||g||: 0.011140
Loss: 0.008033; ||g||: 0.010970
Loss: 0.007914; ||g||: 0.010806
Loss: 0.007798; ||g||: 0.010646
Loss: 0.007685; ||g||: 0.010491
Loss: 0.007576; ||g||: 0.010340
Loss: 0.007470; ||g||: 0.010194
Loss: 0.007367; ||g||: 0.010052
Loss: 0.007266; ||g||: 0.009914
Loss: 0.007169; ||g||: 0.009779
Loss: 0.007074; ||g||: 0.009648
Loss: 0.006981; ||g||: 0.009521
Loss: 0.006891; ||g||: 0.009397
Loss: 0.006804; ||g||: 0.009277
Loss: 0.006718; ||g||: 0.009159
Loss: 0.006635; ||g||: 0.009044
Loss: 0.006553; ||g||: 0.008932
Loss: 0.006474; ||g||: 0.008823
Loss: 0.006397; ||g||: 0.008717
Loss: 0.006321; ||g||: 0.008613
Loss: 0.006247; ||g||: 0.008512
Loss: 0.006175; ||g||: 0.008413
Loss: 0.006105; ||g||: 0.008316
Loss: 0.006036; ||g||: 0.008222
Loss: 0.005969; ||g||: 0.008129
Loss: 0.005903; ||g||: 0.008039
Loss: 0.005839; ||g||: 0.007951
Loss: 0.005776; ||g||: 0.007865
Loss: 0.005715; ||g||: 0.007780
Loss: 0.005654; ||g||: 0.007697
Loss: 0.005596; ||g||: 0.007617
Loss: 0.005538; ||g||: 0.007537
Loss: 0.005481; ||g||: 0.007460
Loss: 0.005426; ||g||: 0.007384
Loss: 0.005372; ||g||: 0.007309
Loss: 0.005319; ||g||: 0.007237
Loss: 0.005266; ||g||: 0.007165
Loss: 0.005215; ||g||: 0.007095
Loss: 0.005165; ||g||: 0.007026
tensor([[3.3747],
        [1.0860]], requires_grad=True)
```

The result is same as the result that we optimized same loss from same iterations.

(f)

```python
import torch
import torch.optim as optim
import torch.nn as nn

torch.manual_seed(1)
X = torch.Tensor([[-2, 1, 2],[1, 1, 1]])
# [your task1]
##############################
## modify the dataset so that it can be used here and is equivalent to the one
## used in LogisticRegression.py and LogisticRegression2.py
## Dimensions: y (3)
y = torch.Tensor([0, 1, 1])
##############################


alpha = 1

class ShallowNet(nn.Module):
    def __init__(self):
        super(ShallowNet, self).__init__()
        self.fc1 = nn.Linear(2,1, bias=False)

    def forward(self, X):
        return self.fc1(X)

net = ShallowNet()
print(net)

net.fc1.weight.data = torch.Tensor([[0.1, 0.1]])

print(net(torch.transpose(X,0,1)).squeeze())

optimizer = optim.SGD(net.parameters(), lr=alpha)
optimizer.zero_grad()

criterion = nn.BCEWithLogitsLoss()

for iter in range(100):
    netOutput = net(torch.transpose(X,0,1)).squeeze()

    # [your task2]
    ##############################
    ## provide the arguments for the criterion function
    loss = criterion(netOutput, y)
    ##############################

    loss.backward()
    gn = 0
    for f in net.parameters():
        gn = gn + torch.norm(f.grad)
    print("Loss: %f; ||g||: %f" % (loss, gn))
    optimizer.step()
    optimizer.zero_grad()

for f in net.parameters():
    print(f)
```

```
Loss: 0.008415; ||g||: 0.011496
Loss: 0.008283; ||g||: 0.011315
Loss: 0.008156; ||g||: 0.011140
Loss: 0.008033; ||g||: 0.010970
Loss: 0.007914; ||g||: 0.010806
Loss: 0.007798; ||g||: 0.010646
Loss: 0.007685; ||g||: 0.010491
Loss: 0.007576; ||g||: 0.010340
Loss: 0.007470; ||g||: 0.010194
Loss: 0.007367; ||g||: 0.010052
Loss: 0.007266; ||g||: 0.009914
Loss: 0.007169; ||g||: 0.009779
Loss: 0.007074; ||g||: 0.009649
Loss: 0.006981; ||g||: 0.009521
Loss: 0.006891; ||g||: 0.009397
Loss: 0.006804; ||g||: 0.009276
Loss: 0.006718; ||g||: 0.009159
Loss: 0.006635; ||g||: 0.009044
Loss: 0.006553; ||g||: 0.008932
Loss: 0.006474; ||g||: 0.008823
Loss: 0.006397; ||g||: 0.008717
Loss: 0.006321; ||g||: 0.008613
Loss: 0.006247; ||g||: 0.008512
Loss: 0.006175; ||g||: 0.008413
Loss: 0.006105; ||g||: 0.008316
Loss: 0.006036; ||g||: 0.008222
Loss: 0.005969; ||g||: 0.008129
Loss: 0.005903; ||g||: 0.008039
Loss: 0.005839; ||g||: 0.007951
Loss: 0.005776; ||g||: 0.007865
Loss: 0.005715; ||g||: 0.007780
Loss: 0.005654; ||g||: 0.007697
Loss: 0.005596; ||g||: 0.007617
Loss: 0.005538; ||g||: 0.007537
Loss: 0.005481; ||g||: 0.007460
Loss: 0.005426; ||g||: 0.007384
Loss: 0.005372; ||g||: 0.007309
Loss: 0.005319; ||g||: 0.007237
Loss: 0.005266; ||g||: 0.007165
Loss: 0.005215; ||g||: 0.007095
Loss: 0.005165; ||g||: 0.007026
Parameter containing:
tensor([[3.3747, 1.0860]], requires_grad=True)
```

$$p(y=1|x) = \frac{1}{1+\exp\left(-w^T\begin{bmatrix}x\\1\end{bmatrix}\right)}$$

$$p(y=0|x) = 1 - p(y=1|x)$$

$$= 1 - \frac{1}{1+\exp\left(-w^T\begin{bmatrix}x\\1\end{bmatrix}\right)}$$

$$= 1 - \frac{\exp\left(w^T\begin{bmatrix}x\\1\end{bmatrix}\right)}{1+\exp\left(w^T\begin{bmatrix}x\\1\end{bmatrix}\right)}$$

$$= \frac{1+\exp\left(w^T\begin{bmatrix}x\\1\end{bmatrix}\right) - \exp\left(w^T\begin{bmatrix}x\\1\end{bmatrix}\right)}{1+\exp\left(w^T\begin{bmatrix}x\\1\end{bmatrix}\right)}$$

$$= \frac{1}{1+\exp\left(w^T\begin{bmatrix}x\\1\end{bmatrix}\right)}$$

$$p(y|x) = \frac{1}{1+\exp\left((1-2y)w^T\begin{bmatrix}x\\1\end{bmatrix}\right)}$$

$1-2y$:

1 : −1

0 : +1

And the result is same as previous result.

[3] [Support Vector Machine]

D of 4 samples :

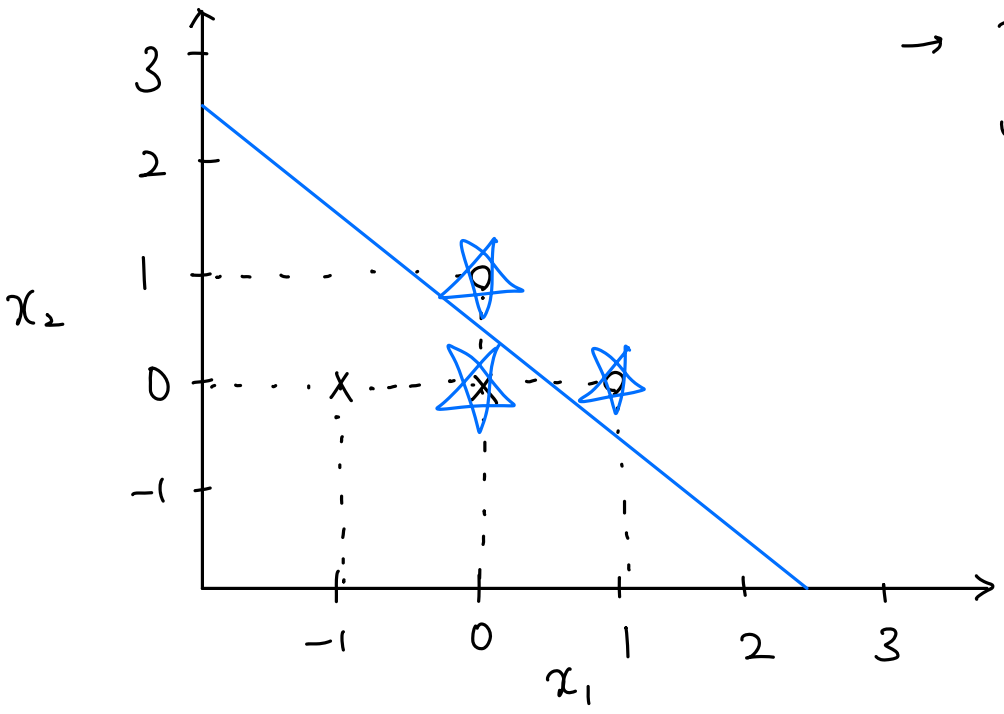
x_1	x_2	label
1	0	+
0	1	+
0	0	−
-1	0	−

binary classification problem of learning decision boundary.

use the hard-margin SVM to find the max-margin hyperplane :

$$\min_{w,b} \frac{1}{2}\|w\|_2^2 \quad \text{s.t.} \quad y^{(i)}(w_1 x_1^{(i)} + w_2 x_2^{(i)} + b) \geq 1 \quad \forall i \in \{1, 2, \dots, 6\}$$

(a)



→ three support vectors with star

(b)

support vectors $\begin{cases} +1 & (1,0) \quad (0,1) \\ -1 & (0,0) \end{cases}$

Set of support vectors $S_{s.v} = \{(1,0;+1), (0,1;+1), (0,0;-1)\}$

where the element of $S_{s.v}$ is $(x_1, x_2; y)$.

goal: calculate the values of $w_1, w_2, b \in \mathbb{R}$ for hard-margin SVM

$$y^{(i)}\left(w^T x^{(i)} + b\right) = 1 \quad \forall i$$

① $(1,0;+1)$: $1 \cdot w_1 + 0 \cdot w_2 + b = 1 \rightarrow w_1 + b = 1$

② $(0,1;+1)$: $0 \cdot w_1 + 1 \cdot w_2 + b = 1 \rightarrow w_2 + b = 1$

③ $(0,0;-1)$: $0 \cdot w_1 + 0 \cdot w_2 + b = -1 \rightarrow b = -1$

$w_1 - 1 = 1 \rightarrow w_1 = 2$

$w_2 - 1 = 1 \rightarrow w_2 = 2$

optimal solution : $w = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$, $b = -1$

(c)

L_1 soft-margin SVM formulation:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{N}\xi_i \quad \text{s.t. } y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i \text{ (g)},$$

$$\xi_i \geq 0 \text{ (h)}, \quad \forall i \in \{1, 2, \cdots, N\}$$

(J)

goal: Lagrangian of the L_1 norm soft-margin SVM using Lagrangian multipliers α_i's and β_i's

$$y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i \rightarrow 1 - \xi_i - y^{(i)}(w^T x^{(i)} + b) \leq 0$$

$$\therefore \mathcal{L}(w, b, \xi) = \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{N}\xi_i + \sum_{i=1}^{N}\alpha_i\{1 - \xi_i - y^{(i)}(w^T x^{(i)} + b)\} - \sum_{i=1}^{N}\beta_i \cdot \xi_i$$

α, β : dual variable

(d)

KKT stationary condition for the soft-margin SVM

$w, b, \zeta, \alpha, \beta$ satisfy the KKT stationary condition

→ w, b, ζ is optimal solution of primal problem and

α, β is optimal solution of dual problem.

complementary slackness condition :

$$\beta_i \zeta_i = 0 \quad , \quad i = 1, \cdots, N$$

$$\alpha_i \{ y^{(i)} (w^T x^{(i)} + b) - 1 + \zeta_i) = 0 \quad , \quad i = 1, \cdots, N$$

stationary condition :

$$\frac{\partial L}{\partial w} = 0 \rightarrow w - \sum_{i=1}^{N} \alpha_i y^{(i)} x^{(i)} = 0 \rightarrow w = \sum_{i=1}^{N} \alpha_i y^{(i)} x^{(i)}$$

$$\frac{\partial L}{\partial b} = 0 \rightarrow \sum_{i=1}^{N} \alpha_i y^{(i)} = 0$$

$$\frac{\partial L}{\partial \zeta_i} = 0 \rightarrow C - \alpha_i - \beta_i = 0 \quad , \text{where } i = 1, \cdots, N$$

(e)

Using the KKT stationary condition, find the dual of the soft-margin SVM as an optimization problem w.r.t. α_i's.

α_i, β_i is dual variable $\rightarrow$ $\alpha_i, \beta_i \geq 0$. $\quad C - \alpha_i \geq 0 \rightarrow \alpha_i \leq C$

From $\beta_i \geq 0$, constraint of α_i is $\quad 0 \leq \alpha_i \leq C$.

$$\text{Lagrangian } \mathcal{L}(w, b, \zeta, \alpha, \beta) = \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{N}\zeta_i + \sum_{i=1}^{N}\alpha_i\{1 - \zeta_i - y^{(i)}(w^T x^{(i)} + b)\} - \sum_{i=1}^{N}\beta_i \cdot \zeta_i$$

Using the KKT stationary condition $\left(w = \sum_{i=1}^{N}\alpha_i y^{(i)} x^{(i)}\right)$,

$$\rightarrow \frac{1}{2}\left\|\sum_{i=1}^{N}\alpha_i y^{(i)} x^{(i)}\right\|_2^2 + C\sum_{i=1}^{N}\zeta_i + \sum_{i=1}^{N}\alpha_i\left[1 - \zeta_i - y^{(i)}\left\{\left(\sum_{j=1}^{N}\alpha_j y^{(j)} x^{(j)}\right)^T x^{(i)} + b\right\}\right] - \sum_{i=1}^{N}\beta_i\zeta_i$$

$$= \frac{1}{2}\left(\sum_{i=1}^{N}\alpha_i y^{(i)} x^{(i)}\right)^T\left(\sum_{j=1}^{N}\alpha_j y^{(j)} x^{(j)}\right) + C\sum_{i=1}^{N}\zeta_i + \sum_{i=1}^{N}\alpha_i\left[1 - \zeta_i - y^{(i)}\left\{\left(\sum_{j=1}^{N}\alpha_j y^{(j)} x^{(j)}\right)^T x^{(i)} + b\right\}\right] - \sum_{i=1}^{N}\beta_i\zeta_i$$

$$= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)T} x^{(j)} + C\sum_{i=1}^{N} \xi_i$$

$$+ \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \alpha_i \xi_i - \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)T} x^{(j)} + b\sum_{i=1}^{N} \alpha_i y^{(i)} - \sum_{i=1}^{N} \beta_i \xi_i$$

$$= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)T} x^{(j)} + C\sum_{i=1}^{N} \xi_i$$

$$+ \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \alpha_i \xi_i + b\sum_{i=1}^{N} \alpha_i y^{(i)} - \sum_{i=1}^{N} \beta_i \xi_i$$

$$= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)T} x^{(j)} + \sum_{i=1}^{N} (\alpha_i + \beta_i) \xi_i$$

$$+ \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \alpha_i \xi_i + b\sum_{i=1}^{N} \alpha_i y^{(i)} - \sum_{i=1}^{N} \beta_i \xi_i \quad \cdots \left(\text{using } \sum_{i=1}^{N} \alpha_i y^{(i)} = 0, \; C - \alpha_i - \beta_i = 0 \right)$$

$$= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)T} x^{(j)} + \sum_{i=1}^{N} \alpha_i$$

∴ dual of the soft-margin SVM as an optimization problem w.r.t. α_i's :

$$\min_{\alpha} \; -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)T} x^{(j)} + \sum_{i=1}^{N} \alpha_i$$

$$\text{s.t.} \quad \sum_{i=1}^{N} \alpha_i y^{(i)} = 0$$

$$0 \le \alpha_i \le C \qquad \forall_i \in \{1, \ldots, N\}.$$