Homework: GLM and MLE

Problem 1

Step 1: Preparation of data

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

G:\내 드라이브\Class\Programming_2023_Pytorch\New_2024_Chapter_5_MLE_and_GLM\HW

```
import pandas as pd
import torch
import numpy as np
from torch.utils.data import DataLoader, TensorDataset

df = pd.read_csv(r"/content/drive/MyDrive/24-2프로그래밍/competition_awards_data.csv",  sep
df.head()
```

	Awards	Math Score
0	0	43
1	0	38
2	0	41
3	0	33
4	0	39

다음 단계: df변수로 코드 생성 추천 차트 보기 New interactive sheet

```
x = torch.tensor(df['Math Score'].values, dtype=torch.float32)
n = len(x)
#정규화
x_mean = torch.mean(x)
x_std = torch.std(x)
xx = (x-x_mean)/x_std

y = df['Awards'].values

ones = torch.ones([n])
X = torch.stack([ones, xx], axis=1)
Y = torch.tensor(y.reshape([n,1]), dtype=torch.float32)

dataset = TensorDataset(X,Y)
trainloader = DataLoader(dataset=dataset, batch_size=32, shuffle=True)


```

```
print(X.shape, Y.shape)
```

```
torch.Size([200, 2]) torch.Size([200, 1])
```

```
from sklearn.model_selection import train_test_split

# Split the data into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42
# Convert the data to TensorDataset to use with DataLoader
```

```
train_dataset = TensorDataset(X_train, Y_train)
test_dataset = TensorDataset(X_test, Y_test)

train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=32, shuffle=True)
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

Step 2: Model 1

Using the data above, we want to build the poisson regression model to predict the number of awards.

Model 1

```
Awards ~ exp(beta0 + beta1 * MathScore)
```

Answer the following questions.

1. Train the model.
2. You want to predict the number of awards for people having `Math Score = 47, 40, 80`. Calculated the predicted number of awards for these people.

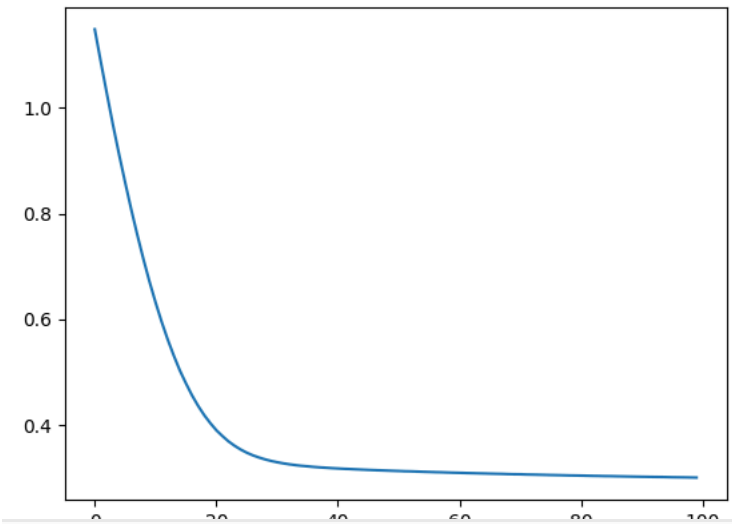
```
#step 1: definition of the model
beta=torch.tensor(torch.randn([2,1]), requires_grad=True)

def forward(xx):
  temp=torch.exp(xx@beta)
  return temp

def criterion(yhat,yy):
  temp=-torch.mean(-yhat+yy*torch.log(yhat))
  return temp

optimizer = torch.optim.SGD([beta], lr=0.01)
```

```
<ipython-input-6-80cbd61bf6cf>:2: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.cl
  beta=torch.tensor(torch.randn([2,1]), requires_grad=True)
```

```
#training
epochs=100
history=[]
n=len(train_loader.dataset)

for epoch in range(epochs):
  LOSS_sum=0
  for xx,yy in train_loader:
    yhat=forward(xx)
    loss=criterion(yhat,yy)
    LOSS_sum=LOSS_sum+loss*len(yy)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

  history.append(LOSS_sum.item()/n)

```

```
import matplotlib.pyplot as plt
plt.plot(history)

```

```
[<matplotlib.lines.Line2D at 0x7e2a58d41870>]
```




```
#step 2 mathscore= 47, 40, 80

forward(torch.tensor([[1,47],[1,40],[1,80]],dtype=torch.float32))
```

```
tensor([[4.5239e+24],
        [8.0004e+20],
        [       inf]], grad_fn=<ExpBackward0>)
```

```
beta
```

```
tensor([[-1.2415],
        [ 1.2343]], requires_grad=True)
```

```
#test MSE
MSE_sum=0
n_test=len(test_loader.dataset)
for xx, yy in test_loader:
  yhat=forward(xx)
  mse=torch.sum((yhat-yy)**2)
  MSE_sum+=mse.item()

MSE_sum/n_test
```

```
0.04800056889653206
```

Step 3: Model 2

You are also considering the following model

Model 2

```
Awards ~ exp(beta0 + beta1 * MathScore + beta2 * MathScore**2)
```

3. Train the model.
4. You want to predict the number of awards for people having `Math Score = 47, 40, 80`. Calculated the predicted number of awards for these people.
5. Calculate the Test MSE for each model. Which model is better in terms of Test MSE?

```
x = torch.tensor(df['Math Score'].values, dtype=torch.float32)
n = len(x)
#정규화
x_mean = torch.mean(x)
x_std = torch.std(x)
xx = (x-x_mean)/x_std

```

```
y = df['Awards'].values

x2=df['Math Score'].values

x2=torch.tensor(x2**2, dtype=torch.float32)
x2_mean = torch.mean(x2)
x2_std = torch.std(x2)
x3 = (x2-x2_mean)/x2_std

ones = torch.ones([n])
X = torch.stack([ones, xx, x3], axis=1)
Y = torch.tensor(y.reshape([n,1]), dtype=torch.float32)

dataset = TensorDataset(X,Y)
trainloader = DataLoader(dataset=dataset, batch_size=32, shuffle=True)
```

더블클릭 또는 Enter 키를 눌러 수정

```
X.shape
```

```
torch.Size([200, 3])
```

```
#데이터 셋 넣기
dataset = TensorDataset(X,Y)
trainloader = DataLoader(dataset=dataset, batch_size=32, shuffle=True)

from sklearn.model_selection import train_test_split

# Split the data into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42
# Convert the data to TensorDataset to use with DataLoader
train_dataset = TensorDataset(X_train, Y_train)
test_dataset = TensorDataset(X_test, Y_test)

train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=32, shuffle=True)
```

```
#model2

beta=torch.tensor(torch.randn([3,1]), requires_grad=True)

def forward(xx):
  temp=torch.exp(xx@beta)
  return temp

def criterion(yhat,yy):
  temp=-torch.mean(-yhat+yy*torch.log(yhat))
  return temp

optimizer = torch.optim.SGD([beta], lr=0.01)

```

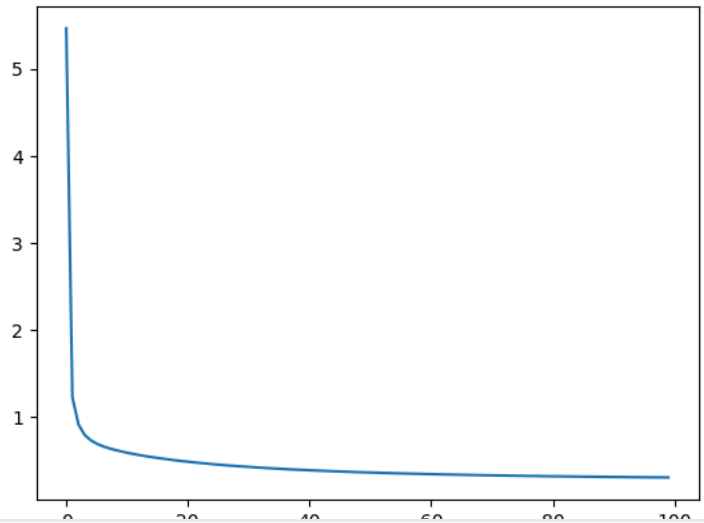
```
<ipython-input-102-5b449b6fcfd2>:3: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.
  beta=torch.tensor(torch.randn([3,1]), requires_grad=True)
```

```
#training
epochs=100
history=[]
n=len(train_loader.dataset)

for epoch in range(epochs):
  LOSS_sum=0
  for xx,yy in train_loader:
```

```
    yhat=forward(xx)
    loss=criterion(yhat,yy)
    LOSS_sum=LOSS_sum+loss*len(yy)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

  history.append(LOSS_sum.item()/n)
```

```
plt.plot(history)
```

```
[<matplotlib.lines.Line2D at 0x7e2a51220df0>]
```




```
beta
```

```
tensor([[-0.9299],
        [ 0.5249],
        [ 0.5035]], requires_grad=True)
```

```
 #Math Score = 47, 40, 80

# 정규화 되돌리기
x_ori=x_std*47+x_mean
x_ori1=x_std*40+x_mean
x_ori2=x_std*80+x_mean
forward(torch.tensor([[1,x_ori,x_ori**2],[1,x_ori1,x_ori1**2],[1,x_ori2,x_ori2**2]],dtyp

```

```
tensor([[inf],
        [inf],
        [inf]], grad_fn=<ExpBackward0>)
```

```
#test MSE
MSE_sum=0
n_test=len(test_loader.dataset)
for xx, yy in test_loader:
  yhat=forward(xx)
  mse=torch.sum((yhat-yy)**2)
  MSE_sum+=mse.item()

MSE_sum/n_test
```

```
0.03132881969213486
```

```
# 모델 2의 성능이 더 좋음

```

Problem 2:

Under the same dataset in Problem 1, answer the following questions.

Problem 2.1.

You want to use

```
torch.nn.Linear(... , ..., bias=True)
```

to modify `forward` function. Repeat procedure in Step 3 of Problem 1 using `torch.nn.Linear` . Redefine the train_loader and test_loader if necessary.

Problem 2.2.

You want to use

```
torch.nn.Linear(... , ..., bias=False)
```

to modify `forward` function. Repeat procedure in Step 3 of Problem 1 using `torch.nn.Linear` . Redefine the train_loader and test_loader if necessary.

Problem 2.3.

Repeat procedure in Step 3 in Problem 1 by modifying the following function. Furthermore, do not use `torch.distributions.poisson.Poisson` in defining the loss function. Redefine the train_loader and test_loader if necessary.

```
class linear_regression(torch.nn.Module):
  def __init__(self, input_size, output_size):
    super().__init__()
    ###
    self.linear = torch.nn.Linear(input_size, output_size)
    ###
  def forward(self,x):
    ###
    yhat = self.linear(x)
    return yhat
    ###

forward = linear_regression(2,1)
```

Problem 3.

You are given the following data. Answer the following questions.

```
!pip install ISLP
```

```
Collecting ISLP
  Downloading ISLP-0.4.0-py3-none-any.whl.metadata (7.0 kB)
Requirement already satisfied: numpy>=1.7.1 in /usr/local/lib/python3.10/dist-packages (from ISLP) (1.26.4)
Requirement already satisfied: scipy>=0.9 in /usr/local/lib/python3.10/dist-packages (from ISLP) (1.13.1)
Requirement already satisfied: pandas>=0.20 in /usr/local/lib/python3.10/dist-packages (from ISLP) (2.2.2)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from ISLP) (4.9.4)
Requirement already satisfied: scikit-learn>=1.2 in /usr/local/lib/python3.10/dist-packages (from ISLP) (1.5.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from ISLP) (1.4.2)
Requirement already satisfied: statsmodels>=0.13 in /usr/local/lib/python3.10/dist-packages (from ISLP) (0.14.4)
Collecting lifelines (from ISLP)
  Downloading lifelines-0.29.0-py3-none-any.whl.metadata (3.2 kB)
Collecting pygam (from ISLP)
  Downloading pygam-0.9.1-py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from ISLP) (2.5.0+cu121)
```

```
Collecting pytorch-lightning (from ISLP)
  Downloading pytorch_lightning-2.4.0-py3-none-any.whl.metadata (21 kB)
Collecting torchmetrics (from ISLP)
  Downloading torchmetrics-1.5.1-py3-none-any.whl.metadata (20 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.20->
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.20->ISLP) (202
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.20->ISLP) (2
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.13->ISLP
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.13->IS
Requirement already satisfied: matplotlib>=3.0 in /usr/local/lib/python3.10/dist-packages (from lifelines->ISLP) (3.7
Requirement already satisfied: autograd>=1.5 in /usr/local/lib/python3.10/dist-packages (from lifelines->ISLP) (1.7.0
Collecting autograd-gamma>=0.3 (from lifelines->ISLP)
  Downloading autograd-gamma-0.5.0.tar.gz (4.0 kB)
  Preparing metadata (setup.py) ... done
Collecting formulaic>=0.2.2 (from lifelines->ISLP)
  Downloading formulaic-1.0.2-py3-none-any.whl.metadata (6.8 kB)
Requirement already satisfied: progressbar2<5.0.0,>=4.2.0 in /usr/local/lib/python3.10/dist-packages (from pygam->ISL
Collecting scipy>=0.9 (from ISLP)
  Downloading scipy-1.11.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (60 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.4/60.4 kB 4.6 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.57.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning->ISLP
Requirement already satisfied: PyYAML>=5.4 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning->ISLP)
Requirement already satisfied: fsspec>=2022.5.0 in /usr/local/lib/python3.10/dist-packages (from fsspec[http]>=2022.5
Requirement already satisfied: typing-extensions>=4.4.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-ligh
Collecting lightning-utilities>=0.10.0 (from pytorch-lightning->ISLP)
  Downloading lightning_utilities-0.11.8-py3-none-any.whl.metadata (5.2 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->ISLP) (3.16.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch->ISLP) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->ISLP) (3.1.4)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch->ISLP) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->to
Collecting interface-meta>=1.2.0 (from formulaic>=0.2.2->lifelines->ISLP)
  Downloading interface_meta-1.3.0-py3-none-any.whl.metadata (6.7 kB)
Requirement already satisfied: wrapt>=1.0 in /usr/local/lib/python3.10/dist-packages (from formulaic>=0.2.2->lifelin
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.10/dist-packages (from fsspec[htt
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from lightning-utilities>=0.10
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0->li
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0->lifeli
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0->li
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0->li
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0->lifeli
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0->li
```

```
import ISLP
Bikeshare = ISLP.load_data("Bikeshare")
```

Problem 3.1.

You are given Model 1

$$Y_i \sim Pois(\mu_i)$$

where $\mu_i = \exp\left(b_0 + b_1 X_{1,i} + \cdots + b_4 X_{4,i}\right)$ and

$$\begin{cases} Y = bikers \\ X_1 = mnth \\ X_2 = hr \\ X_3 = temp \\ X_4 = weathersit \end{cases}$$

and Model 2:

$$Y_i \sim Pois(\mu_i)$$

where $\mu_i = \exp\left(b_0 + b_1 X_{1,i} + \cdots + b_4 X_{4,i}\right)$ and

$$\begin{cases} Y = bikers \\ X_1 = mnth \\ X_2 = hr \\ X_3 = workingday \\ X_4 = temp \\ X_5 = weathersit \end{cases}$$

Which is better model in terms of TEST MSE?

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```