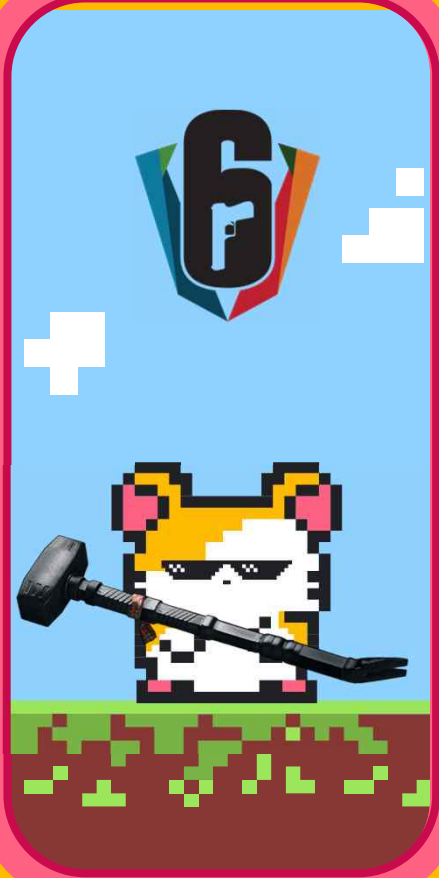
Today's
8조 게임
Hamster

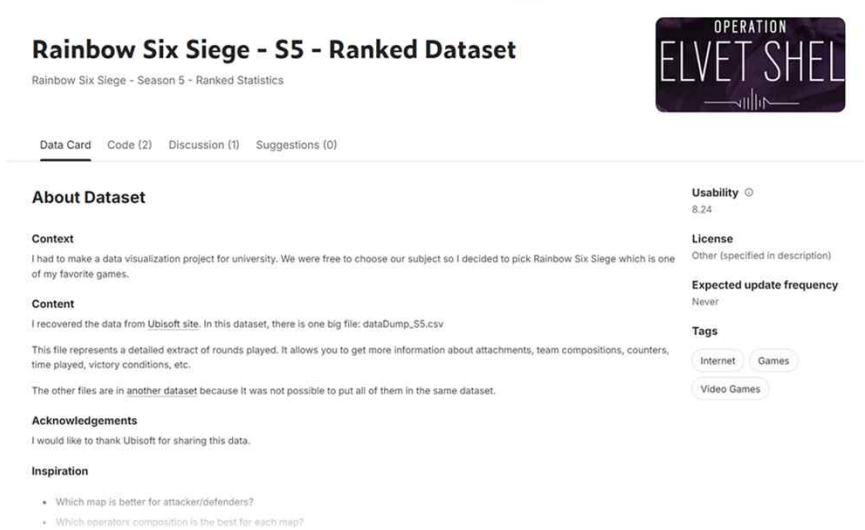


02

RAINBOW SIX SIEGE

레인보우식스 시즈 승부 예측

데이터 출처



Rainbow Six Siege - S5 - Ranked Dataset

Rainbow Six Siege - Season 5 - Ranked Statistics

OPERATION ELVET SHEL

Data Card | Code (2) | Discussion (1) | Suggestions (0)

About Dataset

Context

I had to make a data visualization project for university. We were free to choose our subject so I decided to pick Rainbow Six Siege which is one of my favorite games.

Content

I recovered the data from Ubisoft site. In this dataset, there is one big file: dataDump_S5.csv

This file represents a detailed extract of rounds played. It allows you to get more information about attachments, team compositions, counters, time played, victory conditions, etc.

The other files are in another dataset because it was not possible to put all of them in the same dataset.

Acknowledgements

I would like to thank Ubisoft for sharing this data.

Inspiration

- Which map is better for attacker/defenders?
- Which operators composition is the best for each map?

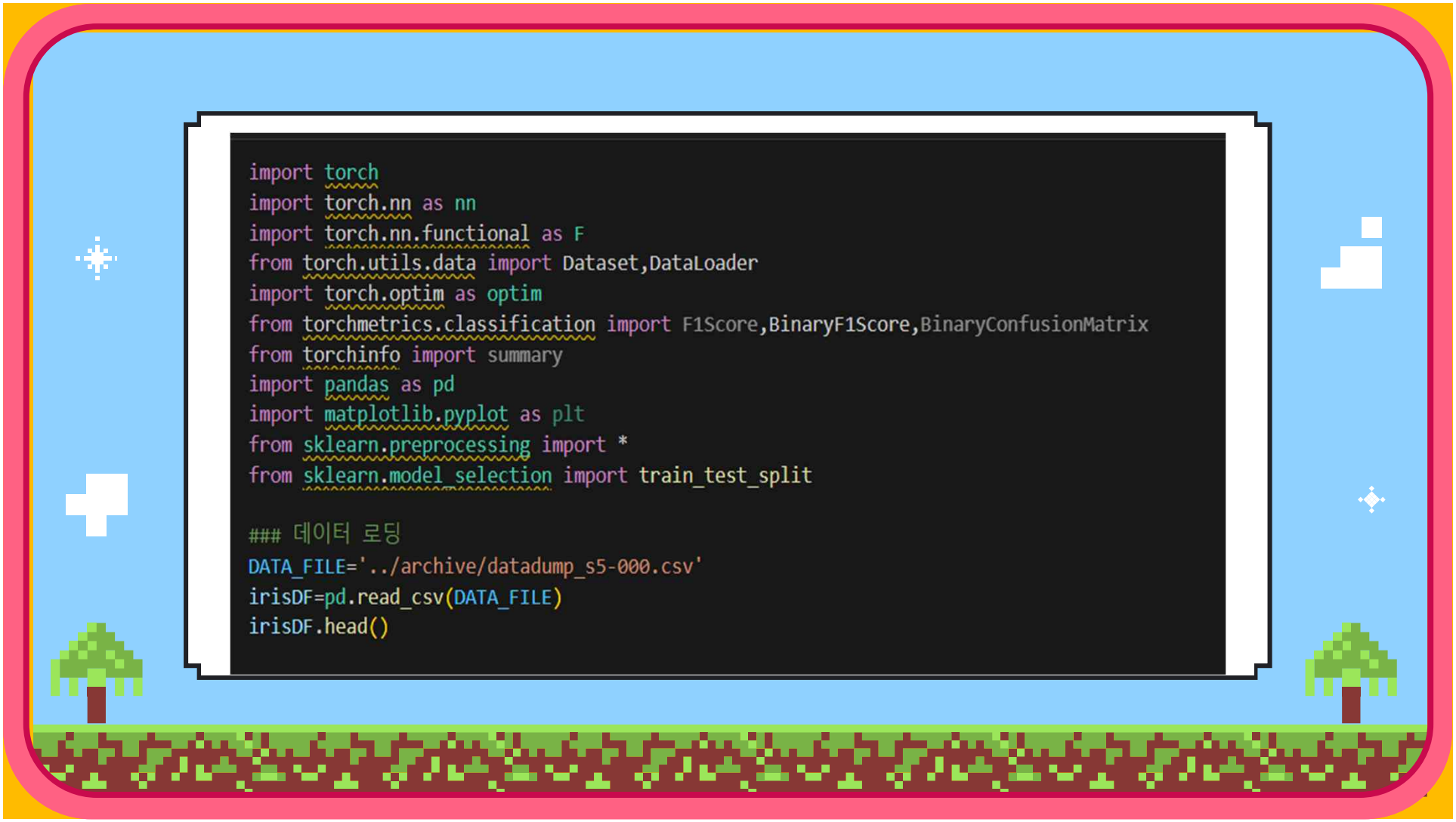
Usability 8.24

License Other (specified in description)

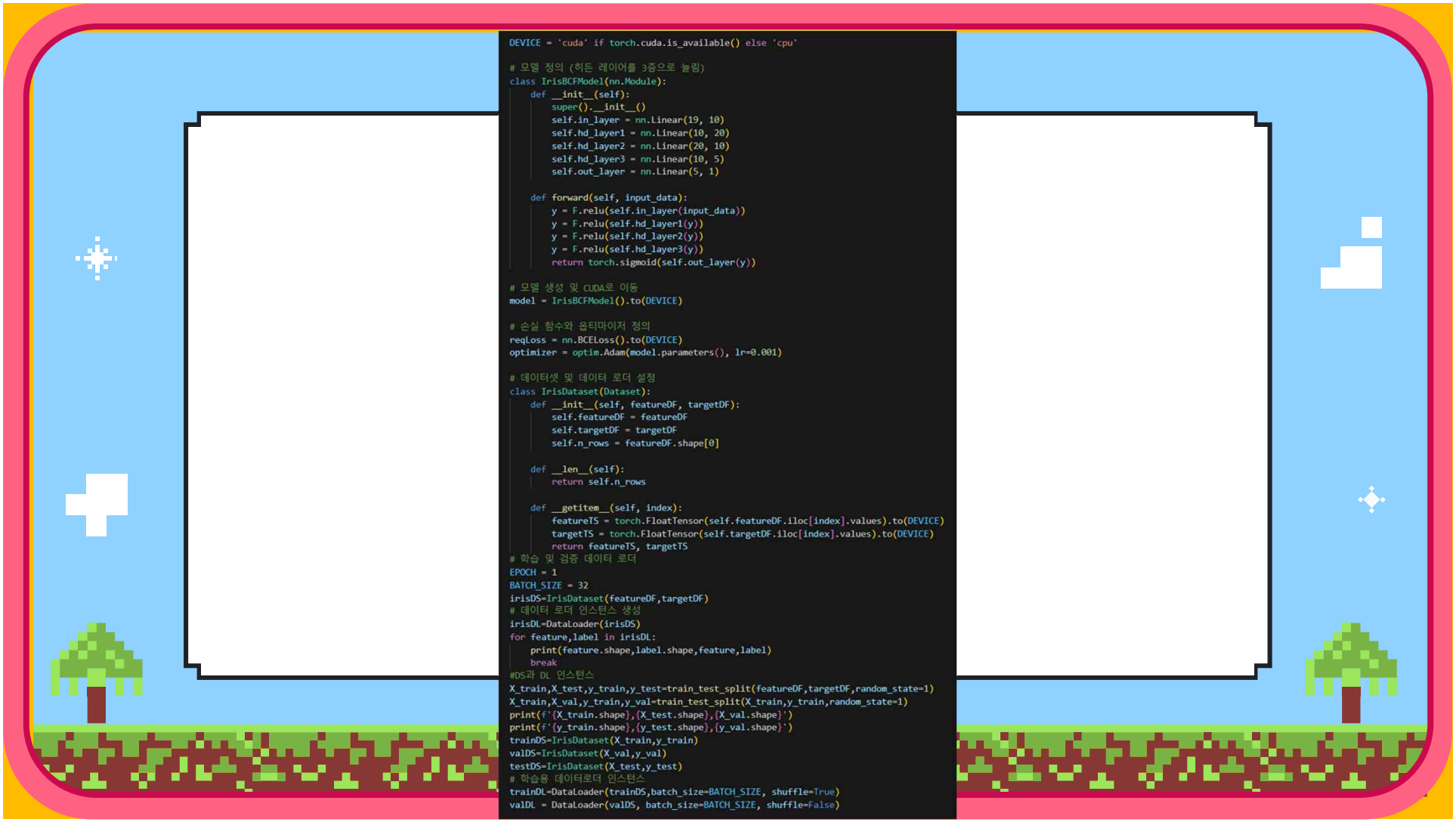
Expected update frequency Never

Tags Internet, Games, Video Games

https://www.kaggle.com/datasets/maxcobra/rainbow-six-siege-s5-ranked-dataset

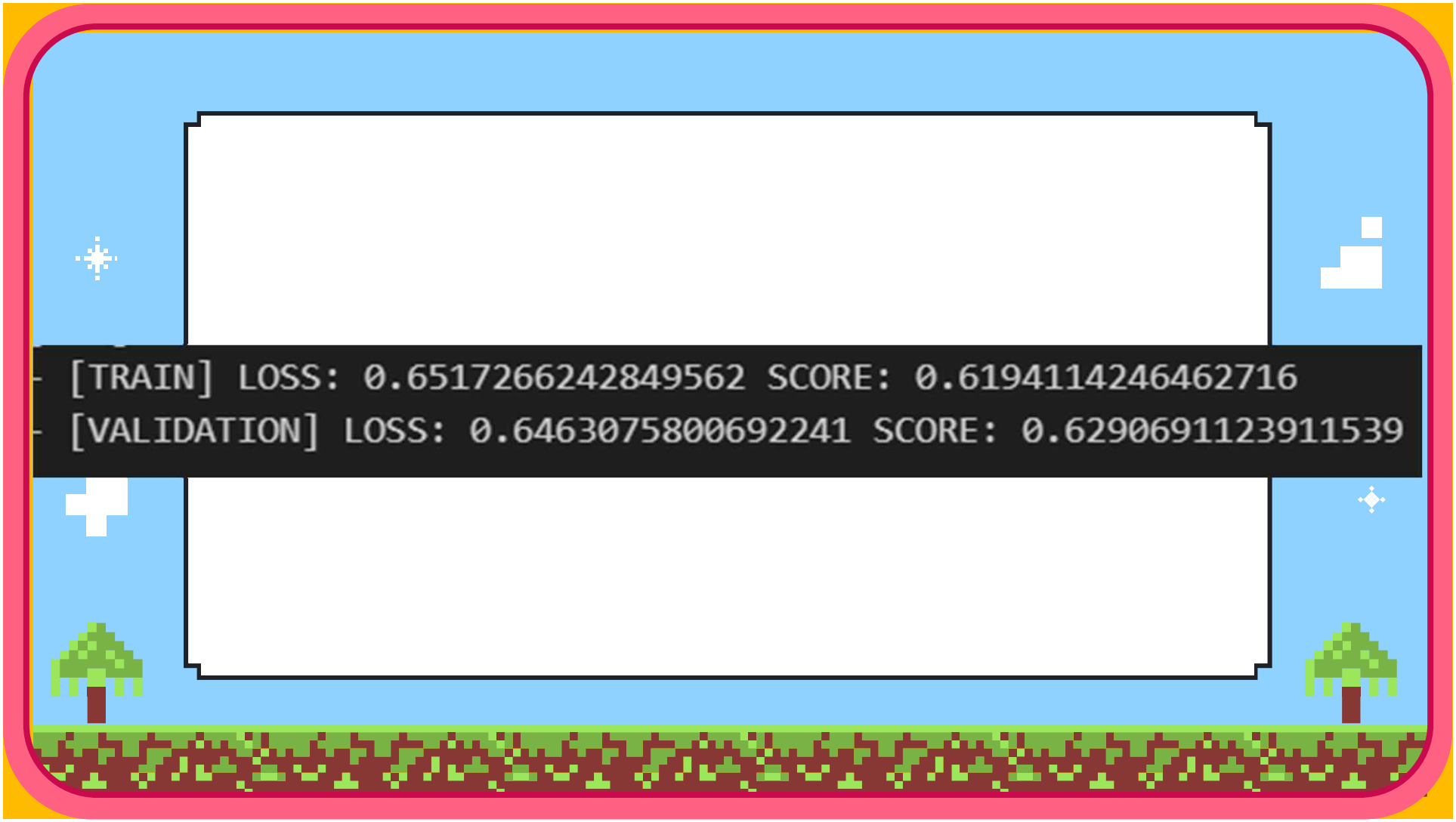

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset,DataLoader
import torch.optim as optim
from torchmetrics.classification import F1Score,BinaryF1Score,BinaryConfusionMatrix
from torchinfo import summary
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import *
from sklearn.model_selection import train_test_split
데이터 로딩
DATA_FILE='../archive/datadump_s5-000.csv'
irisDF=pd.read_csv(DATA_FILE)
irisDF.head()

```python
# 타겟 (승률 예측)
targetDF = irisDF[['haswon']]

# 피처 (승률에 영향을 줄 수 있는 중요한 요소들)
feature_columns = ['platform', 'gamemode', 'mapname', 'objectivelocation', 'roundnumber',
                   'roundduration', 'clearancelevel', 'skillrank', 'role', 'operator',
                   'primaryweapon', 'primaryweapontype', 'primarysight', 'primarygrip',
                   'primaryunderbarrel', 'primarybarrel', 'secondaryweapon',
                   'secondaryweapontype', 'secondarygadget']

featureDF = irisDF[feature_columns]

# 카테고리형 변수들을 숫자로 인코딩
for col in feature_columns:
    if featureDF[col].dtype == 'object':  # 카테고리형 데이터라면
        featureDF[col] = featureDF[col].astype('category').cat.codes
featureDF.info()
```



```
DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'

# 모델 정의 (히든 레이어를 3층으로 늘림)
class IrisBCFModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.in_layer = nn.Linear(19, 10)
        self.hd_layer1 = nn.Linear(10, 20)
        self.hd_layer2 = nn.Linear(20, 10)
        self.hd_layer3 = nn.Linear(10, 5)
        self.out_layer = nn.Linear(5, 1)

    def forward(self, input_data):
        y = F.relu(self.in_layer(input_data))
        y = F.relu(self.hd_layer1(y))
        y = F.relu(self.hd_layer2(y))
        y = F.relu(self.hd_layer3(y))
        return torch.sigmoid(self.out_layer(y))

# 모델 생성 및 CUDA로 이동
model = IrisBCFModel().to(DEVICE)

# 손실 함수와 옵티마이저 정의
reqLoss = nn.BCELoss().to(DEVICE)
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 데이터셋 및 데이터 로더 설정
class IrisDataset(Dataset):
    def __init__(self, featureDF, targetDF):
        self.featureDF = featureDF
        self.targetDF = targetDF
        self.n_rows = featureDF.shape[0]

    def __len__(self):
        return self.n_rows

    def __getitem__(self, index):
        featureTS = torch.FloatTensor(self.featureDF.iloc[index].values).to(DEVICE)
        targetTS = torch.FloatTensor(self.targetDF.iloc[index].values).to(DEVICE)
        return featureTS, targetTS
# 학습 및 검증 데이터 로더
EPOCH = 1
BATCH_SIZE = 32
irisDS=IrisDataset(featureDF,targetDF)
# 데이터 로더 인스턴스 생성
irisDL=DataLoader(irisDS)
for feature,label in irisDL:
    print(feature.shape,label.shape,feature,label)
    break
#DS과 DL 인스턴스
X_train,X_test,y_train,y_test=train_test_split(featureDF,targetDF,random_state=1)
X_train,X_val,y_train,y_val=train_test_split(X_train,y_train,random_state=1)
print(f'{X_train.shape},{X_test.shape},{X_val.shape}')
print(f'{y_train.shape},{y_test.shape},{y_val.shape}')
trainDS=IrisDataset(X_train,y_train)
valDS=IrisDataset(X_val,y_val)
testDS=IrisDataset(X_test,y_test)
# 학습용 데이터로더 인스턴스
trainDL=DataLoader(trainDS,batch_size=BATCH_SIZE, shuffle=True)
valDL = DataLoader(valDS, batch_size=BATCH_SIZE, shuffle=False)
```

```python
# 학습 및 검증 루프
def train_model(trainDL, valDL, EPOCH, BATCH_SIZE):
    LOSS_HISTORY, SCORE_HISTORY = [[], []], [[], []]
    for epoch in range(EPOCH):
        model.train()
        loss_total, score_total = 0, 0

        for featureTS, targetTS in trainDL:
            featureTS, targetTS = featureTS.to(DEVICE), targetTS.to(DEVICE)

            # 예측 및 손실 계산
            pre_y = model(featureTS)
            loss = reqLoss(pre_y, targetTS)
            loss_total += loss.item() * featureTS.size(0)

            # 성능 평가
            pre_y_binary = (pre_y > 0.5).float()
            score = BinaryF1Score().to(DEVICE)(pre_y_binary, targetTS)
            score_total += score.item() * featureTS.size(0)

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

        # 검증 루프
        model.eval()
        with torch.no_grad():
            val_loss_total, val_score_total = 0, 0
            for val_featureTS, val_targetTS in valDL:
                val_featureTS, val_targetTS = val_featureTS.to(DEVICE), val_targetTS.to(DEVICE)

                pre_val = model(val_featureTS)
                loss_val = reqLoss(pre_val, val_targetTS)
                val_loss_total += loss_val.item() * val_featureTS.size(0)

                pre_val_binary = (pre_val > 0.5).float()
                score_val = BinaryF1Score().to(DEVICE)(pre_val_binary, val_targetTS)
                val_score_total += score_val.item() * val_featureTS.size(0)

        # 학습 및 검증 손실/성능 평균 계산
        train_loss_avg = loss_total / len(trainDL.dataset)
        train_score_avg = score_total / len(trainDL.dataset)
        val_loss_avg = val_loss_total / len(valDL.dataset)
        val_score_avg = val_score_total / len(valDL.dataset)

        # 결과 출력
        print(f'[{epoch + 1}/{EPOCH}]\n- [TRAIN] LOSS: {train_loss_avg} SCORE: {train_score_avg}')
        print(f'- [VALIDATION] LOSS: {val_loss_avg} SCORE: {val_score_avg}')
# 학습 시작
train_model(trainDL, valDL, EPOCH, BATCH_SIZE)
```


- [TRAIN] LOSS: 0.6517266242849562 SCORE: 0.6194114246462716
- [VALIDATION] LOSS: 0.6463075800692241 SCORE: 0.6290691123911539