Deep Learning

```
import pandas as pd
import os
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
```

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

Data import

```
traindata_url = 'https://bitbucket.org/hyuk125/lg_dic/raw/889649d1bc273bf53967
testdata_url = 'https://bitbucket.org/hyuk125/lg_dic/raw/889649d1bc273bf53967c
train_data = pd.read_csv(traindata_url)
test_data = pd.read_csv(testdata_url)
```

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

le.fit(train_data.label == 5)

train_data.label = le.transform(train_data.label == 5)
test_data.label = le.transform(test_data.label == 5)
```

Deep learning - classification 모델

Pytorch 모델에 입력하기 위한 데이터 변환

```
train_data = torch.from_numpy(train_data.values).float()
test_data = torch.from_numpy(test_data.values).float()
```

```
BATCH_SIZE = 15
epochs = 2
learning_rate = 0.001
drop_prob = 0.5
```

```

data_loader = torch.utils.data.DataLoader(train_data,
                                          batch_size=BATCH_SIZE,
                                          shuffle=True,
                                          num_workers=0)
```

Deep learning 모델 정의 with Xavier initializer & dropout

```
class DNNModel(nn.Module):
    def __init__(self):
        super(DNNModel, self).__init__()
        self.layer1 = nn.Linear(28 *28, 300)
        self.layer2 = nn.Linear(300, 300)
        self.layer3 = nn.Linear(300, 2)
        self.relu = nn.ReLU()

        self.dropout = nn.Dropout(p=drop_prob)

        nn.init.xavier_uniform_(self.layer1.weight)
        nn.init.xavier_uniform_(self.layer2.weight)
        nn.init.xavier_uniform_(self.layer3.weight)


    def forward(self, x):

        layers = nn.Sequential(self.layer1, self.relu, self.dropout,
                               self.layer2, self.relu, self.dropout,
                               self.layer3).to(device)
        out = layers(x)
        return out

model = DNNModel()
model
```

```
DNNModel(
  (layer1): Linear(in_features=784, out_features=300, bias=True)
  (layer2): Linear(in_features=300, out_features=300, bias=True)
  (layer3): Linear(in_features=300, out_features=2, bias=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.5, inplace=False)
)
```

학습 시작

```
criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.Adam(model.parameters(), lr = learning_rate)
```

```
for epoch in range(epochs):
```

```
    running_cost = 0.0

    for step, (batch_data) in enumerate(data_loader):
        batch_x = batch_data[:, 1:].view(-1, 28*28).to(device)
        batch_y = batch_data[:, 0].to(device).long()

        optimizer.zero_grad()

        outputs = model(batch_x)
        cost = criterion(outputs, batch_y)

        cost.backward()
        optimizer.step()

        running_cost += cost.item()
        if step % 200 == 199:
            print('[%d, %5d] cost: %.3f' % (epoch + 1, step + 1, running_cost
            running_cost = 0.0

```

```
[1,   200] cost: 13.980
[1,   400] cost: 4.697
[1,   600] cost: 2.456
[1,   800] cost: 2.250
[1,  1000] cost: 1.086
[1,  1200] cost: 1.153
[1,  1400] cost: 0.812
[1,  1600] cost: 0.751
[1,  1800] cost: 0.519
[1,  2000] cost: 0.516
[1,  2200] cost: 0.383
[1,  2400] cost: 0.303
[1,  2600] cost: 0.226
[1,  2800] cost: 0.199
[1,  3000] cost: 0.184
[1,  3200] cost: 0.204
[1,  3400] cost: 0.199
[1,  3600] cost: 0.148
[1,  3800] cost: 0.188
[1,  4000] cost: 0.190
[2,   200] cost: 0.146
[2,   400] cost: 0.146
[2,   600] cost: 0.141
[2,   800] cost: 0.179
[2,  1000] cost: 0.217
[2,  1200] cost: 0.154
[2,  1400] cost: 0.097
[2,  1600] cost: 0.162
[2,  1800] cost: 0.132
[2,  2000] cost: 0.177
[2,  2200] cost: 0.241
[2,  2400] cost: 0.195
[2,  2600] cost: 0.183
[2,  2800] cost: 0.251
[2,  3000] cost: 0.265
[2,  3200] cost: 0.174
[2,  3400] cost: 0.176
[2,  3600] cost: 0.162
```

```
[2,  3800] cost: 0.194
[2,  4000] cost: 0.107
```

정확도 판단

Confusion matrix

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score
```

```
with torch.no_grad():
    X_test = test_data[:, 1:].view(-1, 28 * 28).float().to(device)
    y_test = test_data[:, 0].float()

    prediction = model(X_test).cpu()
    print(confusion_matrix(torch.argmax(prediction, 1), y_test))
    print("==Precision==")
    print(precision_score(torch.argmax(prediction, 1), y_test, average=None))
    print(precision_score(torch.argmax(prediction, 1), y_test, average='weight
    print("Recall")
    print(recall_score(torch.argmax(prediction, 1), y_test, average=None))
    print(recall_score(torch.argmax(prediction, 1), y_test, average='weighted'
```

```
[[9046  301]
 [  62  591]]
==Precision==
[0.9931928  0.66255605]
0.9716022181751223
Recall
[0.96779715 0.9050536 ]
0.9637
```

✓ 0초 오전 11:42에 완료됨 ● ×