실험결과(Autoencoder)

사용한 라이브러리 (OneClassSVM, Isolation Forest도 같은 라이브러리 사용)

- Pandas : 1.5.0
- Numpy : 1.23.2
- Pytorch : 1.11.0

대략적 설명

정상 데이터만으로 구성된 Train data를 Autoencoder모델에 입력으로 넣고 그대로 복원하는 방식으로 모델을 학습시킨다.

```
def fit(self, ):
    self.model.to(self.device)
    best_score = 0
    best_loss = 1000000000
    final_threshold = 0
    for epoch in range(EPOCHS):
        self.model.train()
        train_loss = []
        for x, y in iter(self.train_loader):
            x = x.float().to(self.device)
            self.optimizer.zero_grad()
            _x = self.model(x)
            loss = self.criterion(x, _x)
            loss.backward()
            self.optimizer.step()
            train_loss.append(loss.item())

        mean_train_loss = np.mean(train_loss)
```

이상이라고 판단하는 방식은 특정 threshold 값 이상 mse 오차가 발생할 시 해당 데이터를 이상이라고 판단하는 정책을 사용하였다.

```
error = x-_x
loss = self.criterion(x,_x)
val_loss.append(loss.item())
power = torch.pow(error,2)
tmp_mse = power.mean(axis=1).cpu()
mse += tmp_mse
true += y.tolist()
```

threshold 값은 충분한 실험 후에 아래와 같은 범위로 300개를 잡아주었다.

In [86]:
```
threshold = []
for i in range(300):
    threshold.append(1e-4*(i+1))
```

각 에폭 마다 검증을 수행하는데, 이 때 threshold값 범위내에서 가장 높은 F1 score, 라벨, 예측라벨, validation loss, 가장 높은 f1_score를 얻은 threshold값을 반환하여 준다.

```
def validation(self, eval_model, threshold):
    cos = nn.CosineSimilarity(dim=1, eps=1e-6)
    eval_model.eval()
    pred = []
    true = []
    mse = []
    val_loss = []
    best_score = 0
    best_threshold = 0
    with torch.no_grad():
        for x, y in iter(self.val_loader):
            x = x.float().to(self.device)
            _x = eval_model(x)
            error = x-_x
            loss = self.criterion(x,_x)
            val_loss.append(loss.item())
            power = torch.pow(error,2)
            tmp_mse = power.mean(axis=1).cpu()
            mse += tmp_mse
            true += y.tolist()

    mse_arr = np.array(mse)
    for i in threshold:
        pred = np.where(mse_arr>i, 1,0).tolist()
        f1 = f1_score(true, pred, average='macro')
        if f1 > best_score:
            best_threshold = i
            best_score = f1

    best_pred = np.where(mse_arr>best_threshold, 1,0).tolist()

    #plt.plot(mse)
    #plt.axhline(threshold, color='r', linewidth=1)
    #plt.show()
    return f1_score(true, best_pred, average='macro'), true, best_pred, np.mean(val_loss),best_threshold
```

반환 받은 f1 score 점수를 기준으로 Learning rate scheduler를 수행하고, f1 score가 가장 높았을 때 모델을 저장하고, threshold값을 저장한다

```
    score, true, pred, val_loss, best_threshold = self.validation(self.model, self.threshold)

    print(f'Epoch : [{epoch+1}] Train loss : [{mean_train_loss}]\nVal Score : [{score}])\nVal Loss : [{val_
    print(confusion_matrix(true, pred))
    print(f'Best threshold : [{best_threshold}]')

    self.scheduler.step(score)

    '''if best_loss > val_loss:
        best_loss = val_loss
        torch.save(model.module.state_dict(), '/Users/cain/Desktop/Study/ML,DL_Project/Fraud_Detection/mode
        print("Save Model~_~")'''

    if best_score < score:
        best_score = score
        final_threshold = best_threshold
        torch.save(model.module.state_dict(), '/Users/cain/Desktop/Study/ML,DL_Project/Fraud_Detection/mode
        print("Save Model~_~")
```

위에서 얻은 최고의 threshold값을 testing 함수의 인자로 넣어 테스트한다.

In [*]:
```
def testing(model, threshold, test_loader, device):
    model.to(device)
    model.eval()
    pred = []
    true = []
    with torch.no_grad():
        for x, y in iter(test_loader):
            x = x.float().to(device)
            _x = model(x)
            error = x-_x
            power = torch.pow(error,2)
            tmp_mse = power.mean(axis=1).cpu()
            batch_pred = np.where(np.array(tmp_mse)>threshold, 1,0).tolist()
            pred += batch_pred
            true += y.tolist()
    print(f1_score(true, pred, average='macro'))
    print(confusion_matrix(true, pred))
```

In [79]:
```
load_model = AutoEncoder()
load_model.load_state_dict(torch.load('/Users/cain/Desktop/Study/ML,DL_Project/Fraud_Detection/model/best_model.pth
load_model = nn.DataParallel(load_model)
load_model.eval()
testing(load_model,final_threshold, test_loader, device)
```

Autoencoder

- Dense Layer로 구성
- epoch = 100
- Learning Rate = 1e-3
- Batch size : 32
- Optimizer : Adam
- Loss function : MSE
- Learning rate scheduler : ReduceLROnPlateau
- Activation function : ReLU, Leaky ReLU
- Weight intializatinon : he normal

Layer	Activation Function	Threshold	F1-score
input - 32 - 8 - 32 - output	ReLU	0.0125	0.8526
input - 32 - 8 - 32 - output	LeakyReLU	0.0055	0.9047
input - 64 - 8 - 64 - output	ReLU	0.0090	0.9022
input - 64 - 8 - 64 - output	LeakyReLU	0.0039	0.9333
input - 128 - 8 - 128 - output	ReLU	0.0113	0.9269
input - 128 - 8 - 128 - output	LeakyReLU	0.0047	0.9276
input - 32 - 4 - 32 - output	ReLU	0.0135	0.8986
input - 32 - 4 - 32 - output	LeakyReLU	0.0089	0.8766
input - 64 - 4 - 64 - output	ReLU	0.0131	0.8817
input - 64 - 4 - 64 - output	LeakyReLU	0.0148	0.8625
input - 128 - 4 - 128 - output	ReLU	0.0137	0.8768
input - 128 - 4 - 128 - output	LeakyReLU	0.0101	0.9172