

인공지능 개론

L07.1 Latent Factor Model with Pytorch

국민대학교

소프트웨어융합대학원

박하명

Contents

- ❖ **Pytorch Basic**
- ❖ Linear Regression with Pytorch

MovieLens 데이터

<https://grouplens.org/datasets/movielens/>

- 영화 평점 데이터셋
 - 크기별로 다양하게 존재: 100K, 1M, 10M, 20M ...
 - 실습에서 사용할 데이터셋: **100K**


데이터 준비

- ml-100k.zip 파일을 받아서 적절한 위치에 압축 풀기
- colab을 사용할 경우 압축을 풀어서 구글드라이브에 저장

older datasets

MovieLens 100K Dataset

MovieLens 100K movie ratings. Stable benchmark dataset. 100,000 ratings from 1000 users on 1700 movies. Released 4/1998.

- [README.txt](#) 
- [ml-100k.zip](#) (size: 5 MB, [checksum](#))
- [Index of unzipped files](#)

Permalink: <https://grouplens.org/datasets/movielens/100k/>

Colab에서 구글드라이브 연결하기

- Colab에서 구글 드라이브 mount (연결) 하기

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

클릭

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6c

Enter your authorization code:

복사 붙여넣기

이 코드를 복사하여 애플리케이션으로 전환한 다음 붙여넣으세요.

4/zQGt-

HvkjWMqyMKyvZ

zwJpb3z1Qfn83EQ



Colab에서 구글드라이브 연결하기

- 파일 확인하기

```
!ls "/content/gdrive/My Drive/ml-100k"
```

```
allbut.pl u1.base u2.test u4.base u5.test ub.base u.genre u.occupation  
mku.sh    u1.test u3.base u4.test ua.base ub.test  u.info    u.user  
README   u2.base u3.test u5.base ua.test u.data    u.item
```

Pytorch 사용하기

- 필요한 모듈 import 하기

```
import torch
import pandas as pd
import torch.nn.functional as F
import matplotlib.pyplot as plt
```

파일 불러오기

- 판다스를 이용하여 파일 불러오기

```
train = pd.read_csv("/content/gdrive/My Drive/ml-100k/ua.base",  
                    sep="\t", names=['user', 'movie', 'rating', 'timestamp'])  
test = pd.read_csv("/content/gdrive/My Drive/ml-100k/ua.test",  
                   sep="\t", names=['user', 'movie', 'rating', 'timestamp'])
```


파일 불러오기

- pytorch tensor 데이터로 변환

```
items = torch.LongTensor(train['movie'])
users = torch.LongTensor(train['user'])
ratings = torch.FloatTensor(train['rating'])
items_test = torch.LongTensor(test['movie'])
users_test = torch.LongTensor(test['user'])
ratings_test = torch.FloatTensor(test['rating'])
```

Latent Factor Model

- rank \rightarrow 사용자 vector, 아이템 vector의 차원
- numUsers \rightarrow 사용자 수
- numItems \rightarrow 아이템 수
- P \rightarrow 아이템 매트릭스
- Q \rightarrow 사용자 매트릭스

```
rank = 10
numItems = items.max() + 1
numUsers = users.max() + 1
P = torch.randn(numItems, rank, requires_grad=True)
Q = torch.randn(numUsers, rank, requires_grad=True)
```

Latent Factor Model

$$H(i, x) = p_i \cdot q_x$$

- 기본 Matrix Factorization 구현

$$\text{cost}(P, Q) = \sum_{(i,x) \in R} (r_{xi} - H(i, x))^2$$

```
optimizer = torch.optim.Adam([P, Q], lr= 0.1)

for epoch in range(1000):
    hypothesis = torch.sum(P[items] * Q[users], dim=1)
    cost = F.mse_loss(hypothesis, ratings)

    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    if epoch % 100 == 0:
        print("epoch: {}, cost: {:.6f}".format(epoch, cost.item()))
```

Latent Factor Model

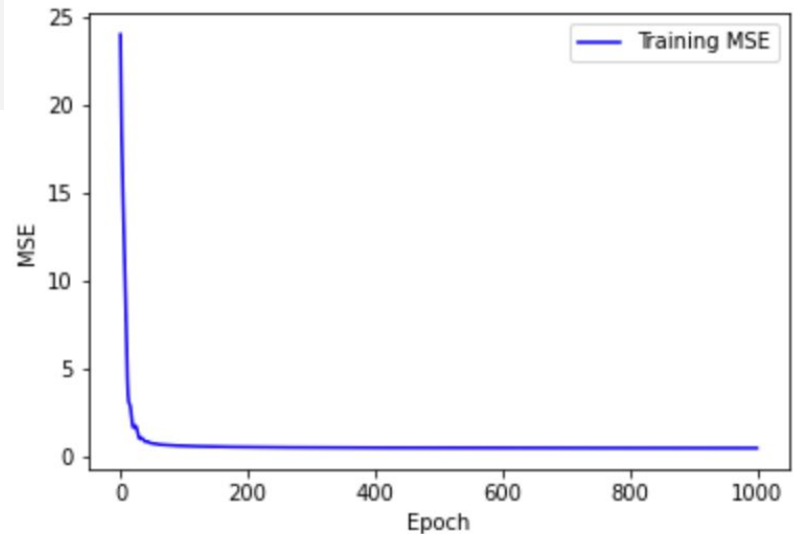
- Matplotlib으로 epoch 마다 **Training MSE** 그려보기

```
X = []  
Y = []  
for epoch in range(1000):  
    hypothesis = torch.sum(P[items] * Q[users], dim=1)  
    cost = F.mse_loss(hypothesis , ratings)  
    ...  
    X.append(epoch)  
    Y.append(cost)  
    if epoch % 100 == 0:  
        ...
```

Latent Factor Model

- Matplotlib으로 epoch 마다 **Training MSE** 그려보기

```
plt.ylabel("MSE")  
plt.xlabel("Epoch")  
plt.plot(X,Y, c="blue", label="Training MSE")  
plt.legend()  
plt.show()
```



Latent Factor Model

- Matplotlib으로 epoch 마다 Test MSE 그려보기

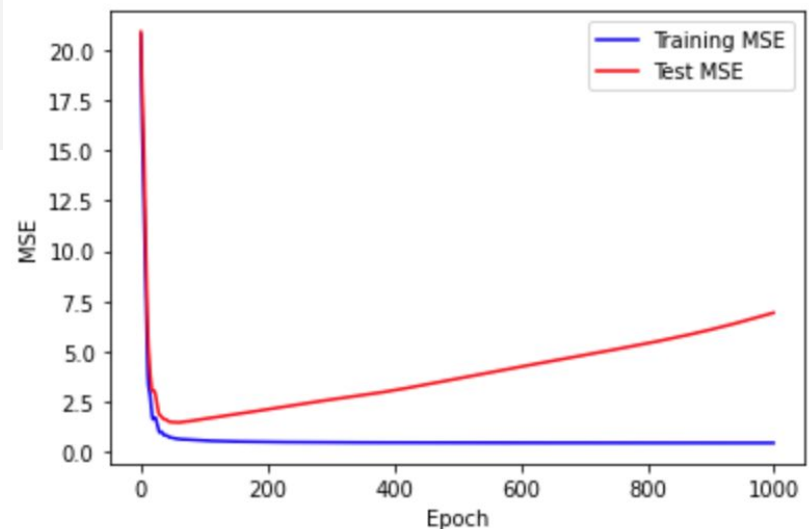
```
X = []
Y = []
Y_test = []
for epoch in range(1000):
    ...
    X.append(epoch)
    Y.append(cost)
    with torch.no_grad():
        hypo_test = torch.sum(P[items_test] * Q[users_test], dim=1)
        cost_test = F.mse_loss(hypo_test, ratings_test)
        Y_test.append(cost_test)

    if epoch % 100 == 0:
        ...
```

Latent Factor Model

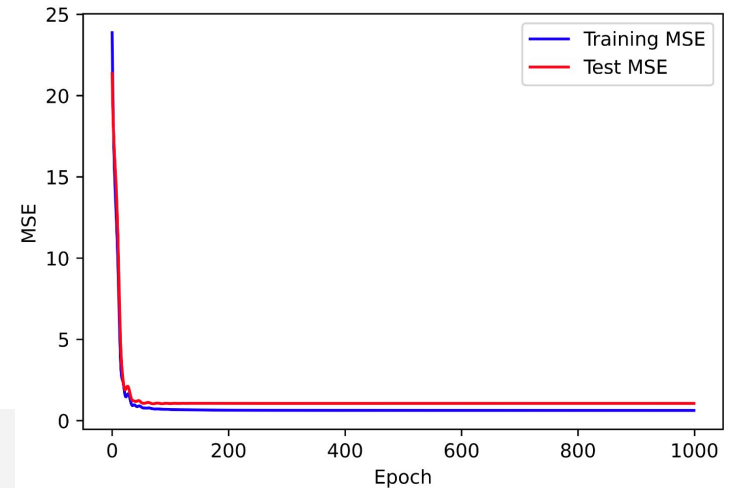
- Matplotlib으로 epoch 마다 **Test MSE** 그려보기

```
plt.ylabel("MSE")
plt.xlabel("Epoch")
plt.plot(X,Y, c="blue", label="Training MSE")
plt.plot(X,Y_test, c="red", label="Test MSE")
plt.legend()
plt.show()
```



Regularization

- Regularization 추가하기



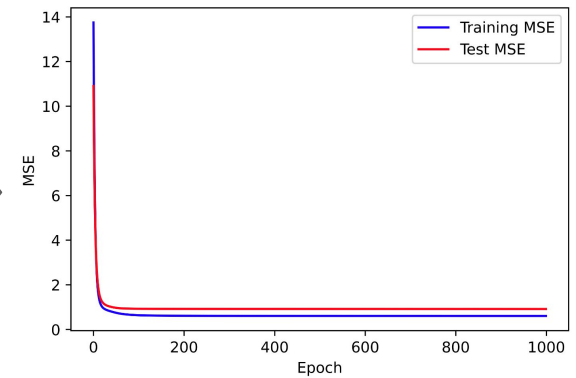
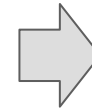
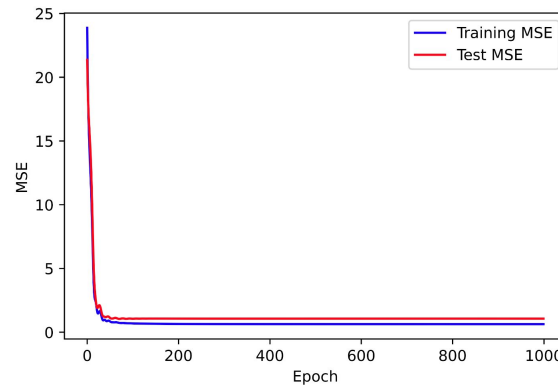
```
lambda1 = 0.0001
lambda2 = 0.0001

for epoch in range(1000):
    hypothesis = torch.sum(P[items] * Q[users], dim=1)
    cost = F.mse_loss(hypothesis, ratings)
    loss = cost + lambda1 * torch.sum(P ** 2) + lambda2 * torch.sum(Q ** 2)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    ...
    if epoch % 100 == 0:
        print("epoch: {}, cost: {:.6f}".format(epoch, cost.item()))
```


Bias

- Bias 추가하기



```
lambda3 = 0.001
lambda4 = 0.001
bias_item = torch.randn(numItems, requires_grad=True)
bias_user = torch.randn(numUsers, requires_grad=True)
mean = (ratings.sum() / len(ratings)).item()
```

```
hypothesis = torch.sum(P[items] * Q[users], dim=1)
```



```
hypothesis = torch.sum(P[items] * Q[users], dim=1) + mean + bias_item[items] +
bias_user[users]
```

```
loss = cost + lambda1 * torch.sum(P ** 2) + lambda2 * torch.sum(Q ** 2)
```



```
loss = cost + lambda1 * torch.sum(P ** 2) + lambda2 * torch.sum(Q ** 2) +
lambda3 * torch.sum(bias_item ** 2) + lambda4 * torch.sum(bias_user ** 2)
```

Question?