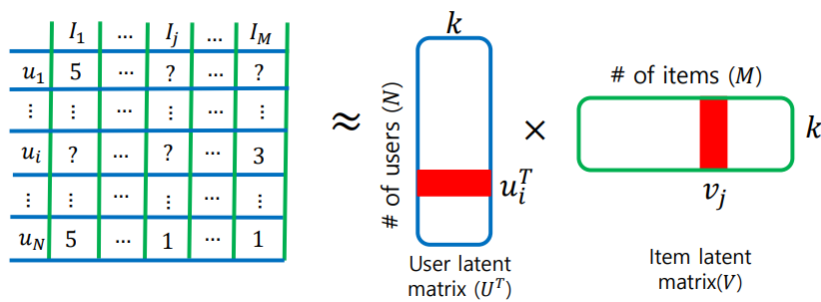


1. Matrix Factorization

User와 Item 간의 평가 정보를 나타내는 Rating Matrix를 User Latent Matrix와 Item Latent Matrix로 분해하는 기법이다. Rating Matrix는 (User의 수) * (Item의 수)로 구성된 행렬이다. 각 칸에는 각 유저가 해당 아이템에 기록한 평가이다. 비어 있는 부분이 더 많기 때문에, 보통 Sparse Matrix가 된다. 이러한 유저가 평가하지 않은 아이템에 대한 평가를 예측하는 것이 목표이다. 따라서 행렬 분해를 통해 평점을 예측할 수 있다.



- User Latent Matrix(U) = (User의 수) * K
- Item Latent Matrix(V) = (Item의 수) * K
- Rating Matrix(R) = ((User의 수) * K) * (K * (Item의 수)) = (User의 수) * (Item의 수)

2. 모델

```
class ModelClass(nn.Module):

    def __init__(self, num_users=610, num_items=193610, rank=16):
        super().__init__()
        self.U = torch.nn.Parameter(torch.randn(num_users + 1, rank))
        self.V = torch.nn.Parameter(torch.randn(num_items + 1, rank))

    def forward(self, users, items):
        ratings = torch.sum(self.U[users] * self.V[items], dim=-1)
        return ratings
```

Model은 일반적인 Matrix Factorization을 활용하여 구현하였다.

3. 학습방법

강의자료에 나와있는 코드를 참고하였다.

```
train_data = RecommendationDataset(f"{args.dataset}/ratings.csv", train=True)
num_train = int(len(train_data) * 0.9)
split_train, split_valid = random_split(train_data,[num_train,
len(train_data)-num_train])
train_loader = DataLoader(split_train, batch_size=args.batch_size,
shuffle=True)
valid_loader = DataLoader(split_valid, batch_size=args.batch_size)
```

train_data와 valid_data를 9:1 비율로 나누었다.

```
X = []
Y = []
Y_test = []
for epoch in range(20):
    cost = 0
    for users, items, ratings in train_loader:
        optimizer.zero_grad()
        ratings_pred = model(users,items)
        loss = criterion(ratings_pred, ratings)
        loss.backward()
        optimizer.step()
        cost += loss.item() * len(ratings)

    cost /= n_ratings

    X.append(epoch+1)
    Y.append(cost)

    print(f"Epoch: {epoch}")
    print("train cost: {:.6f}" .format(cost))

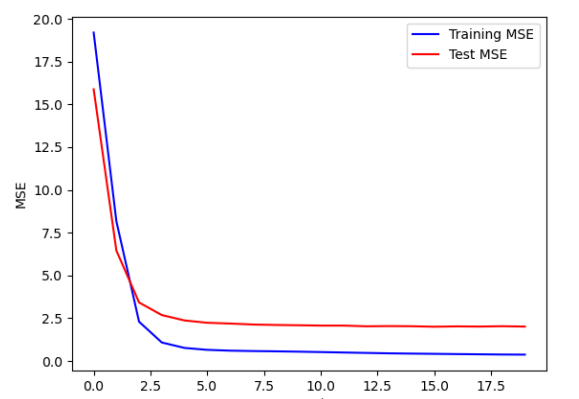
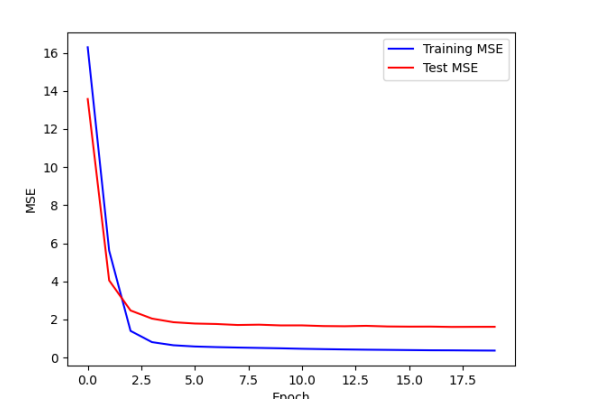
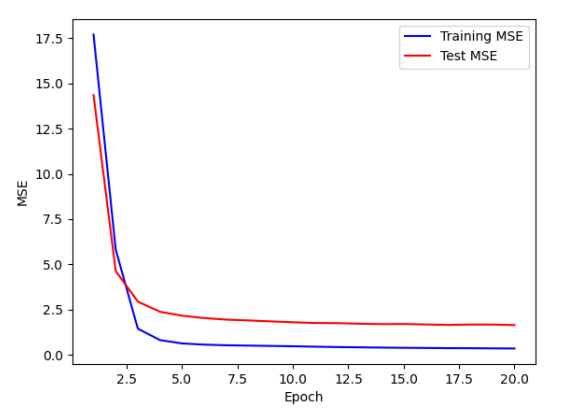
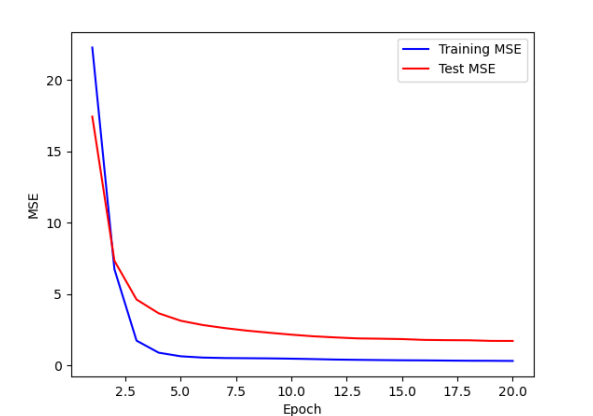
    Y_test.append(calculate_valid(valid_loader,model,n_ratings_pred))

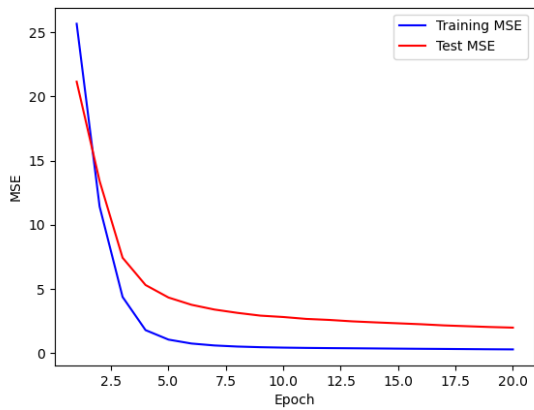
plt.ylabel("MSE")
plt.xlabel("Epoch")
plt.plot(X,Y, c="blue", label="Training MSE")
plt.plot(X,Y_test, c="red", label="Test MSE")
plt.legend()
plt.show()
```

train_data로 학습을 진행하고, valid_data로 cost가 얼마나 떨어지는지 plot을 그려서 확인해 본다.

4. 모델 비교 및 하이퍼파라미터 튜닝

Batchsize, epoch, rank, Learning-rate 등 하이퍼파라미터를 다양하게 적용해보며 실험하였다.

	
<p>Batchsize : 128 Epoch : 20 K : 16 Optimizer : Adam (Lr = 0.01) Valid-cost : 2.014892</p>	<p>Batchsize : 128 Epoch : 20 K : 16 Optimizer : Adam (Lr = 0.01 weight-decay = 1e-5) Valid-cost : 1.614478</p>
	
<p>Batchsize : 128 Epoch : 20 K : 20 Optimizer = Adam (Lr = 0.01, weight-decay = 1e-5) Valid-cost : 1.649236</p>	<p>Batchsize : 128 Epoch : 20 K : 32 Optimizer : adam (Lr = 0.01 weight-decay = 1e-5) Valid-cost : 1.731159</p>



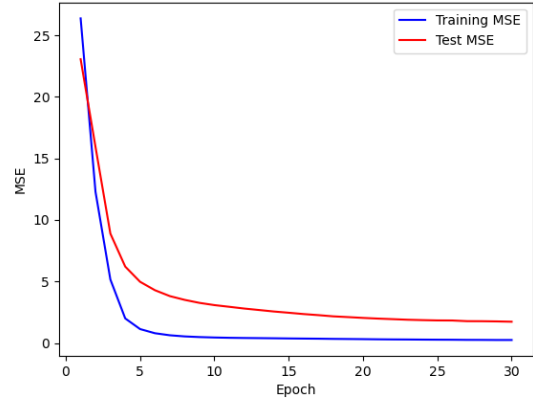
Batchsize : 256

Epoch : 20

K : 32

Optimizer : adam (Lr = 0.01 weight-decay = 1e-5)

Valid-cost : 2.003536



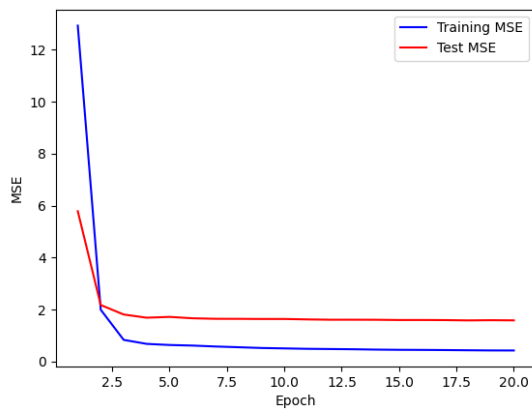
Batchsize : 256

Epoch : 30

K : 32

Optimizer : Adam (Lr = 0.01 weight-decay = 1e-5)

Valid-cost : 1.728883



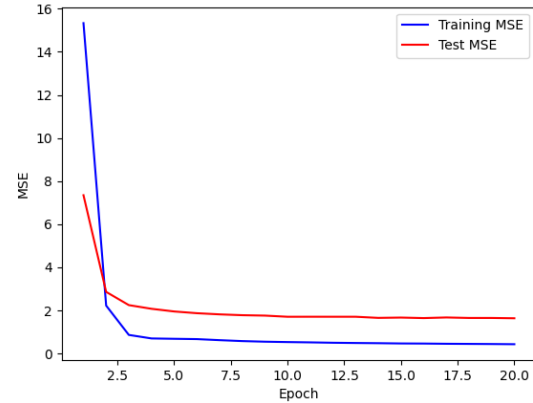
Batchsize : 64

Epoch : 20

K : 16

Optimizer : Adam (Lr = 0.01 weight-decay = 1e-5)

Valid-cost : 1.586613



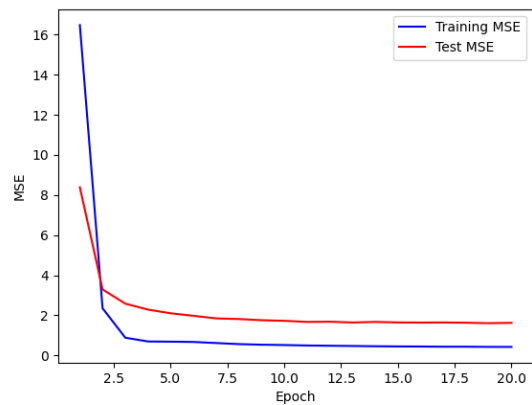
Batchsize : 64

Epoch : 20

K : 24

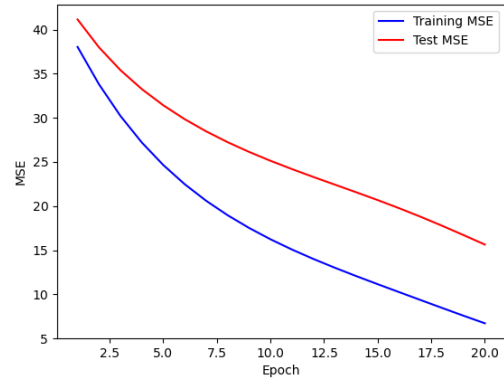
Optimizer : Adam (Lr = 0.01 weight-decay = 1e-5)

Valid-cost : 1.634246



Batchsize : 64

Epoch : 20



Optimizer의 Learning-rate을 0.001로 적용해보

K : 28 Optimizer : Adam (Lr = 0.01 weight-decay = 1e-5) Valid-cost : 1.624341	있을 때의 모습이다. 느리게 cost가 떨어지기 때문에 원하던 cost만큼 값이 떨어지지 않았다. 따라서 Learning-rate 값은 0.01로 사용하게 되었다. 뿐만 아니라, Optimizer로 SGD도 사용해 보았다. 빠르게 계산되긴 했지만, 검색과 적용 결과 비효율적이라고 판단되어 Adam을 사용하게 되었다.
---	--

5. 최종 선정 모델

K(rank) 값이 작다면, user의 Item에 대한 평가의 정보를 담을 수 있는 정도가 낮아지기 때문에, 좋지 못한 결과를 얻을 것이라고 생각했다. 하지만, K=16일 때 Valid_data에 대한 cost가 가장 낮게 측정되었다. 이는 test_data와 valid_data의 비율 문제일 수도 있을 것이라고 생각한다. K값이 낮으면, 학습을 할 때 마다 편차가 커지는 경향도 있었다. 따라서 최종모델로 Batchsize : 64 Epoch : 20 K : 28 Optimizer : Adam (Lr = 0.01 weight-decay = 1e-5) Valid-cost : 1.624341 으로 선정하였다.

6. 결론 및 소감

이렇게 큰 데이터로 모델을 만들어 본적이 없었는데, 이번 프로젝트에서 실데이터와 유사한 데이터로 모델을 만들어 볼 수 있어서 좋은 경험이 되었다. 데이터의 양이 정말 크기 때문에, 학습하는 데에도 시간이 오래 걸렸다. 시간이 오래 걸려서 다양한 실험을 해보는 데에도 많은 시간이 소요되었다. 하지만, 실제로 더 큰 데이터들도 많이 있고, 실생활에 사용되는 데이터들은 이보다 훨씬 더 클 것이기 때문에, 훨씬 더 많은 시간이 소요될 것이라고 생각했다. 이에 자연스럽게, 컴퓨터 성능에 대한 생각도 하게 되었다. 또, 직접 코드를 짜며, 모델을 만들어 보는 것의 중요성을 알게 되었다. 수업시간에 이론 수업과 간단한 실습만으론 이해되지 않던 것이, 직접 코드를 짜보며 공부하니 훨씬 더 깊이 있는 이해를 할 수 있었다. Recommender System 외에도 다른 모델들에 대해 공부해보고 다양한 모델들을 구현해보고 싶다.