Long-Term Project

과목명.	데이터사이언스
담 당.	김 상 욱 교수님
제출일.	2021년 06월 11일
공과대학	컴퓨터소프트웨어학부
학 번	2016025969
이 름.	정지훈

HANYANG UNIVERSITY

목차.

1. Summary of algorithm

- Overview

1	1	5	874965758
1	2	3	876893171
1	3	4	878542960
1	4	3	876893119
1	5	3	889751712
1	7	4	875071561
1	8	1	875072484
1	9	5	878543541
1	11	2	875072262

u1.base

1	6	5	887431973
1	10	3	875693118
1	12	5	878542960
1	14	5	874965706
1	17	3	875073198
1	20	4	887431883
1	23	4	875072895
1	24	3	875071713
1	27	2	876892946

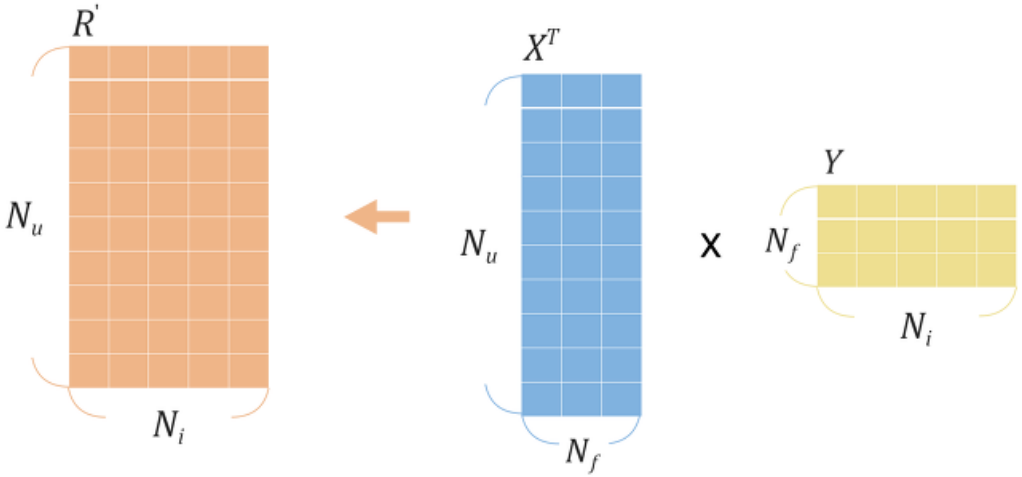
u1.test

1	6	3.169470508761394
1	10	3.623972421956292
1	12	4.785959713391555
1	14	4.219861957100557
1	17	3.577716250941956
1	20	3.387577156017737
1	23	4.259527659906447
1	24	3.55797935261278
1	27	3.8037606000922466

u1.prediction

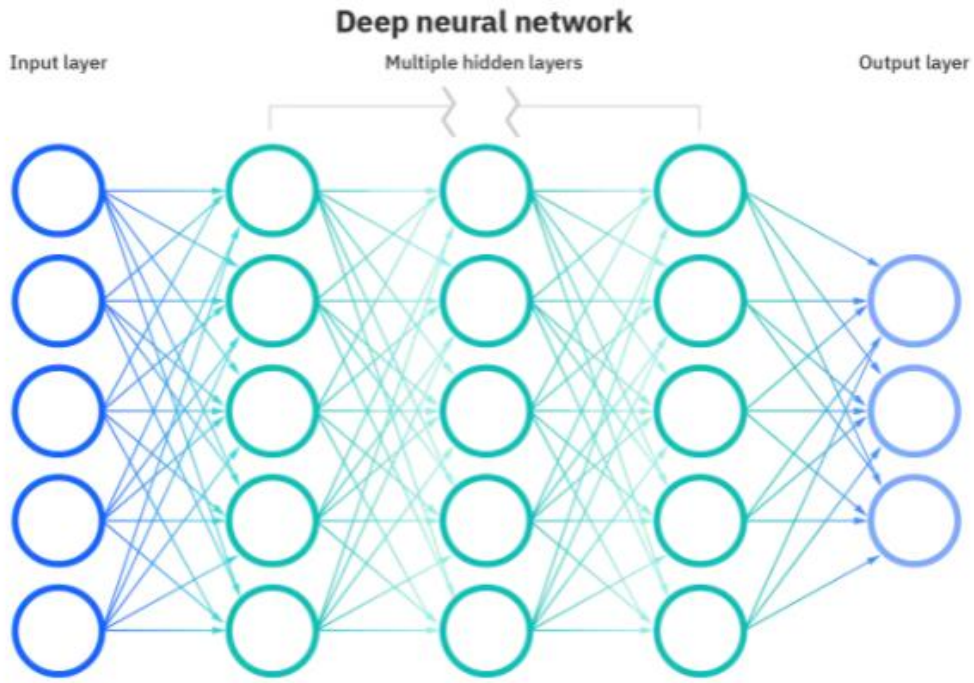
U1.base를 통해 주어진 user와 movie 그리고 평점에 대해서 Matrix Factorization 기법을 사용해 user와 movie의 Latent Factor Matrix를 얻고 두 Matrix 통해 평점을 예측합니다. 얻은 예측 값을 Neural Network Layer을 통과해 최종 값을 얻습니다. 최종 예측 값과 정답 값을 이용해 Loss를 구하고 이를 각 Layer의 weight과 latent vector를 학습합니다. 얻어진 Model은 이제 새로운 user와 movie에 대한 평점을 얻을 수 있습니다.

- Matrix Factorization



Matrix Factorization은 위의 그림과 같이 User와 Movie의 각각 개수를 row로 갖고 사전에 정의한 Factor의 숫자의 크기를 col으로 만드는 latent Factor Matrix를 User, Movie에 대해 각각 만듭니다. 이 두 Matrix에서 추론을 원하는 user-id, movie-id가 들어오면 해당하는 row를 각각 가져와 두 row의 Inner-product 값을 얻습니다. 하지만 User, Movie 별로 평균 등 특성이 다를 것입니다. 이 특성을 고려하기 위해 user-bias와 movie-bias에 대한 Matrix도 만들어 그 값을 예측 값에 더합니다. 이를 통해 얻은 값과 실제 평점에 대한 오차를 Loss로 설정해 Factor Matrix를 update 합니다.

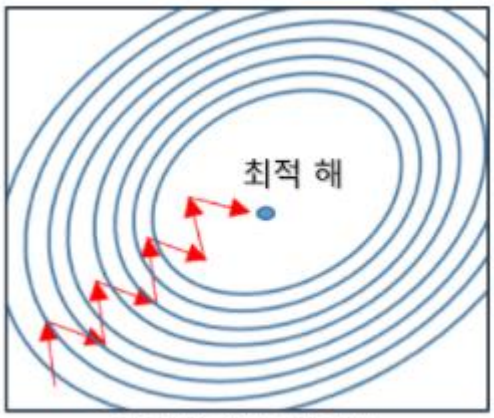
- Neural Network



Input으로 들어온 예측 값을 Neural Network를 통과시켜 기존 Matrix Factorization 방법 보다 더 많은 특성을 학습할 수 있도록 합니다. 각 Layer의 값은 다음 layer의 노드에 값이 전해질 때 input node와 output node 사이의 weight을 곱해주고 bias을 더한 값을 다음 node에 전해집니다. 이후 linear-regression에서 다양한 Feature를 반영하기 위한 Non-linear

regression을 위해 Activation Function인 ReLU 함수를 적용합니다. 이를 통해 얻은 값을 최종 결과 값과 비교해 얻은 Loss를 backward를 통해 Backpropagation을 적용, 각 Layer의 weight과 bias를 update 합니다.

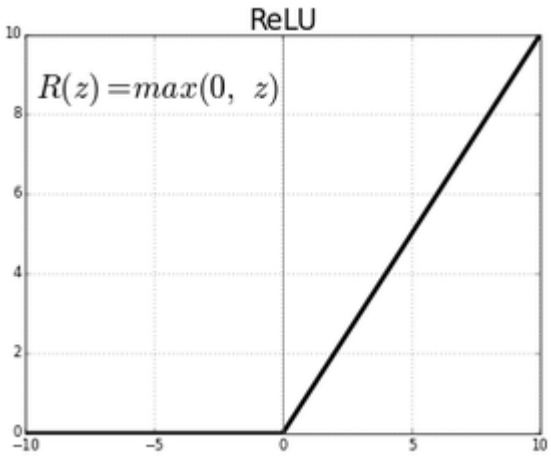
- SGD



확률적 경사 하강법

SGD(Stochastic Gradient Descent)는 기존에 있던 Gradient Descent에서 모든 Data에 대한 Loss를 이용하는 것이 아닌 확률적으로 선택된 data의 Loss만을 이용합니다. 이를 통해 더 빠른 학습이 가능하며 기존 방법에 있었던 Local Minimum에 빠지는 단점을 해결합니다.

- ReLU



ReLU 함수는 기존에 Linear regression에서 다양한 Feature를 분류하기 위한 Non-linear regression으로 변환해주는 Activation Function입니다. Input 값이 0 이하라면 0으로 초리하고 0 이상이라면 Input 값을 Output 값으로 처리합니다.

- Result Test

Data	RMSE
U1	0.9592
U2	0.9475
U3	0.9438
U4	0.9395
U5	0.9426

2. Detailed description of codes

- Data Read

```
train = sys.argv[1]
test = sys.argv[2]
log_dir = './model'
np.random.seed(42)

print("Load Data ... ", end="", flush=True)
df_ratings = pd.read_csv("./data-2/"+train, s
df_ratings.drop('timestamp', axis=1, inplace=

ratings = pd.pivot_table(data=df_ratings, val
dataset = MovieDataset(df_ratings)

users = df_ratings['user']
movies = df_ratings['movie']

user2idx = {user:idx for idx, user in enumera
movie2idx = {movie:idx for idx, movie in enum
n_users = len(df_ratings.user.unique())
n_movies = len(df_ratings.movie.unique())
```

compile에서 base 파일과 그에 대응하는 test 파일의 이름을 인자로 받습니다. 이후 base file에서 학습에 필요하지 않은 timestamp를 제거하고 user와 movie를 각각 row, column으로 해 rating을 value로 갖는 ratings pivot table을 만듭니다. 이후 user와 movie 번호가 중간에 비어 있을 수 있으므로 각각 비어 있지 않은 값으로 matching 하는 Dictionary를 만듭니다.

- RecDataset

```
class RecDataset(Dataset):
    def __init__(self, dataset):
        self.dataset = torch.from_numpy(datas

    def __getitem__(self, idx):
        return self.dataset[idx]

    def __len__(self):
        return len(self.dataset)
```

추후 Model을 학습할 때 Dataloader에서 data를 얻어 올 때 dataset class를 통해 얻습니다. Dataset에 u1.base 값이 담겨 있는 상태에서 index가 주어지면 해당 값을 얻게 됩니다.

- AverageMeter

```
class AverageMeter(object):
    def __init__(self):
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count
```

Model에서 얻은 예측 평점과 실제 점수인 Ground Truth를 가지고 MSE Loss를 구합니다. 이후 batch마다의 loss를 기록해 한 Epoch 안에서의 Loss 평균을 구하게 됩니다. 이 값은 추후에 Model이 학습되는지 그리고 Valid에서 Best Model 저장의 근거가 됩니다.

- Init-weight

```
def _init_weight_(self):
    nn.init.normal_(self.user_factors.weight, 0, 0.1)
    nn.init.normal_(self.movie_factors.weight, 0, 0.1)
    nn.init.normal_(self.user_biases.weight, 0, 0.1)
    nn.init.normal_(self.movie_biases.weight,0, 0.1)

    for m in self.modules():
        if isinstance(m, nn.Linear) and m.bias is not N
            m.bias.data.zero_()
```

만들어지는 latent Factor matrix와 user, movie 각 bias에 대해 값을 초기화 시킵니다.

- Model Init

```
ss MatrixFactorization(nn.Module):
 def __init__(self, n_users, n_movies, n_factors=20):
     super(MatrixFactorization, self).__init__()
     self.user_factors = nn.Embedding(n_users, n_factors, sparse=True)
     self.movie_factors = nn.Embedding(n_movies, n_factors, sparse=True)

     self.user_biases = nn.Embedding(n_users, 1, sparse=True)
     self.movie_biases = nn.Embedding(n_movies, 1, sparse=True)

     self.dropout = nn.Dropout(0.5)
     self.flatten = nn.Flatten()

     self.linear1 = nn.Linear(1, 32)
     self.linear2 = nn.Linear(32, 64)
     self.linear3 = nn.Linear(64, 64)
     self.linear4 = nn.Linear(64, 32)
     self.output = nn.Linear(32, 1)

     self.relu = nn.ReLU()
```

Matrix Factorization의 user, movie 각각 latent factor matrix와 bias matrix를 선언하는 역할을 합니다. User, movie 각각 latent factor matrix에서 index에 해당하는 row를 얻어 dot product를 진행해 얻은 예측 값을 다시 Neural Network에 통과해야 하기에 그를 위한 Network도 선언되어 있는 부분을 볼 수 있습니다. nn.Linear(1,32)는 기존에 input으로 1 node의 값이 들어오면 이것을 다음 Layer 32개의 node로 값을 보내는 신경망의 역할을 합니다. 활성 함수인 relu도 선언이 되어 있습니다.

- Forward

```
def forward(self, user, movie):
    b = self.user_biases(user)
    b += self.movie_biases(movie)

    user_factor = self.user_factors(user)
    movie_factor = self.movie_factors(movie

    x = ((user_factor * movie_factor).sum(d
    x = torch.reshape(x, (-1, 1))
```

Model에 구하고자 하는 평점에 대한 user, movie index가 들어오면 각 matrix에서 factor를 얻어와 product를 통해 값을 얻습니다. 추후 학습을 위해 dimension을 바꿉니다.

```
    x = self.linear1(x)
    x = self.relu(x)
    x = self.dropout(x)

    x = self.linear2(x)
    x = self.relu(x)
    x = self.dropout(x)

    x = self.linear3(x)
    x = self.relu(x)
    x = self.dropout(x)

    x = self.linear4(x)
    x = self.relu(x)

    x = self.output(x)
    x += b
    x = x.squeeze()
    return x
```

예측한 평점을 신경망을 통해 나온 Output에 사전에 정의한 User, Movie bias를 더해 최종 값으로 추론합니다. Train에서 학습할 때 loss는 미분의 chain rule에 따라 backpropagation을 통해 각 신경망과 latent factor vector를 학습합니다. 기존에 학습을 진행할 때 valid loss는 고정이고 train loss만 떨어지는 overfitting 현상이 일어나 dropout을 진행 각 layer에서 사전 정의된 확률만큼 의도적으로 학습이 되지 않게 설정했습니다.

- Train Hyper Parameter

```
fold, (train_ids, valid_ids) in enumerate(kfold.split(dataset
if fold == set_fold:
    break
train_sampler = SubsetRandomSampler(train_ids)
valid_sampler = SubsetRandomSampler(valid_ids)

train_loader = DataLoader(dataset, batch_size=32, sampler=tra
valid_loader = DataLoader(dataset, batch_size=32, sampler=val

best_rmse = 100

model = MatrixFactorization(n_users, n_movies, n_factors=50).
loss_fn = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.05)
rmse = [0] * 5
```

Generalization을 위해 Ensemble을 적용하기 위해 Dataset에 KFold를 적용했습니다. 다만 data가 부족해 충분한 학습이 안 될 것을 우려해 data split은 train : valid = 0.95 : 0.05로 나누었고 20번의 반복은 시간이 오래 걸리

기에 5번의 fold가 끝나면 Train을 끝내 얻어낸 5개의 model에 대해서만 ensemble을 적용했습니다. 이후 각 Fold에서 충분한 학습을 위해 Epochs는 30으로 설정했습니다. Mini-batch를 위해 batch size는 32로 설정하였고 latent Factor의 수는 50, Loss는 RMSE를 위한 MSELoss 그리고 optimizer는 SGD에 learning rate는 0.05로 설정하였습니다.

- Best Model

```
if best_rmse > valid_loss.avg:
    best_rmse = valid_loss.avg
    rmse[fold] = best_rmse
    if not os.path.exists(log_dir):
        os.mkdir(log_dir)

    print(f"Model Save : rmse - {best_rmse}")
    torch.save(model, f'{log_dir}/{train[:2]}_model{fold+1}.
```

학습을 하는 과정에서 개발자는 어떠한 Model이 최적의 model인지 판단을 할 수 없습니다. 따라서 Validation set을 이용해 MSE loss를 구하고 이를 이용해 각 epoch마다 best MSE를 기록, 해당 model을 fold와 train파일 기준으로 저장하도록 만들었습니다.

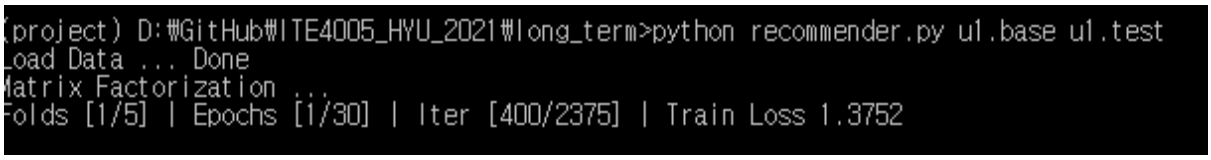
- Ensemble

```
            result[idx][kfold] = float(model(u, m))

    div = 0
    for i in range(set_fold):
        div += (1/rmse[i])
    for idx, row in result_df.iterrows():
        s = 0
        for i in range(set_fold):
            s += result[idx][i] / rmse[i]
        result_df.loc[idx, 'rating'] = s / div
    result_df.to_csv('./test/'+train[:2]+".base_predict
```

5개의 model을 통해 얻어진 값을 각 model의 MSE를 통해 Soft Voting Ensemble을 했습니다. MSE는 낮을수록 좋은 값이기에 MSE의 역수를 이용해서 구현했습니다.

3. Instructions for compiling source codes at TA's computer



파일 경로와 이름 설정을 위해 base 파일과 test파일은 해당 소스 파일이 위치한 곳 data-2 폴더 안에 있어야 하며 best model은 model 폴더 안에 base파일과 fold 정보를 가지고 저장이 되게 됩니다. 다만 이때 base파일은 u0~u9의 이름으로 해야 합니다. 그렇게 학습이 끝나고 prediction은 존재하는 test folder 안에 저장이 되게 됩니다.

4. Any other specification of implementation and testing

Library	Version
Python	3.8.10
Numpy	1.19.2
Pandas	1.2.4
Pytorch	1.4.0
Scikit-learn	0.24.2

Anaconda 가상 환경에서 실행했습니다.

5. References

- Matrix Factorization 그림 : https://yeomko.tistory.com/5

- Neural Network 그림 : https://www.ibm.com/cloud/learn/neural-networks

- SGD 그림 : https://twinw.tistory.com/247