Recommendation System

목차.

1. Summary of algorithm

2. Detailed description of codes

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

4. Any other specification of implementation and testing

5. References

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

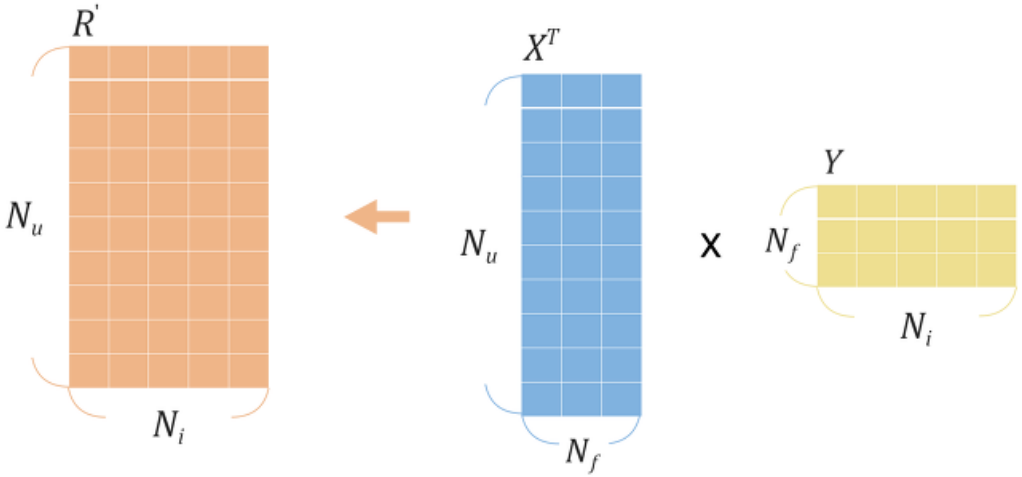
u1.test

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

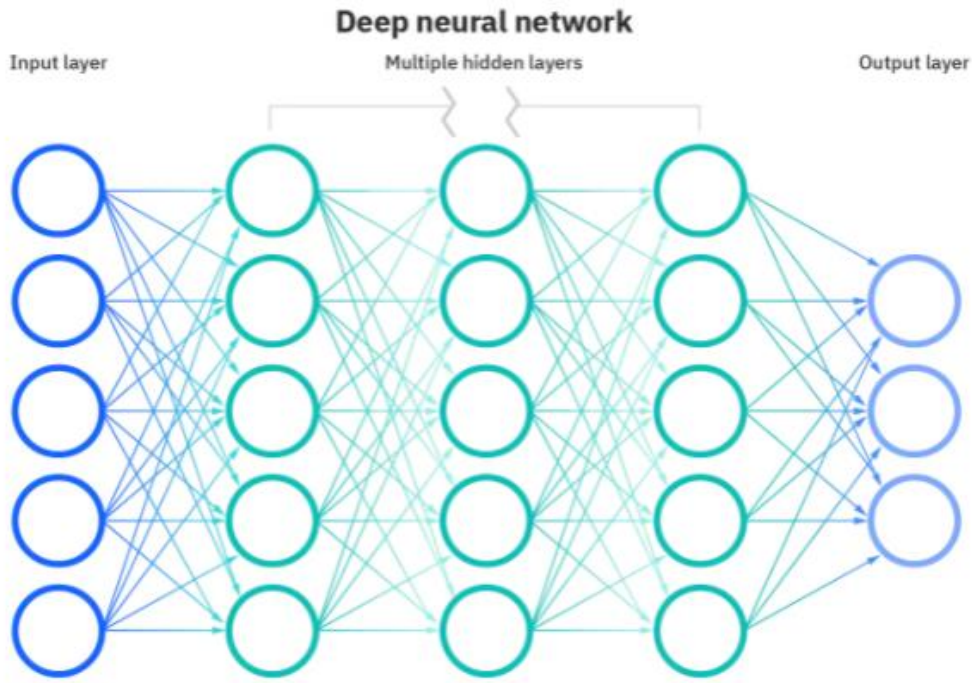
U1.base를 통해 주어진 user와 movie 그리고 평점에 대해서 Matrix Factorization 기법을 사용해 user와 movie의 Latent Factor Matrix를 얻고 두 Matrix 통해 평점을 예측합니다. 얻은 예측 값을 Neural Network Layer을 통과해 최종 값을 얻습니다. 최종 예측 값과 정답 값을 이용해 Loss를 구하고 이를 각 Layer의 weight과 latent vector를 학습합니다. 얻어진 Model은 이제 새로운 user와 movie에 대한 평점을 얻을 수 있습니다.

- 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.

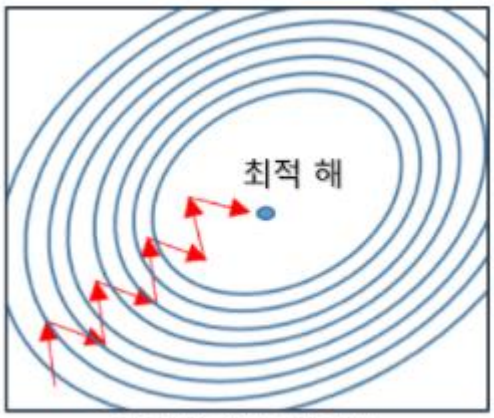
- Neural Network



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.

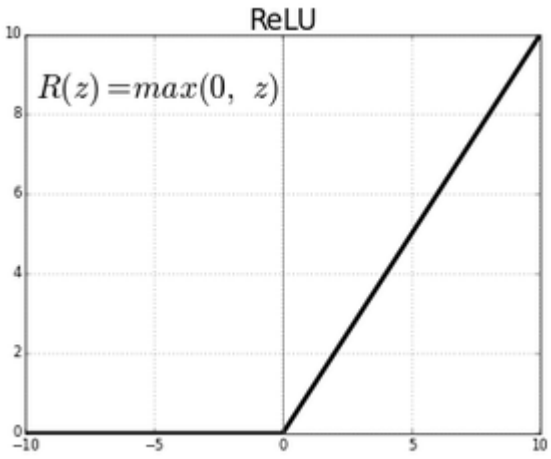
- SGD



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



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.

- Result Test

Data	RMSE
U1	0.9419
U2	0.9298
U3	0.9279
U4	0.9251
U5	0.9309

2. Detailed description of codes

- Data Read

```
input_file = sys.argv[1]
number = input_file[5:-4]
n = int(sys.argv[2])
eps = int(sys.argv[3])
minPts = int(sys.argv[4])

input_df = pd.read_csv('./data-3/'+input_file,
input = input_df.to_numpy()

n_points = len(input)
classify = [-1] * n_points
```

compile에서 주어지는 최대 Cluster 개수와 Eps, MinPts값을 저장한 후 input file을 읽습니다. 이때 input file을 읽을 때 주어진 object_id를 index로 사용하여 Dataframe을 만듭니다. 만들어진 Dataframe을 이용해 총 점의 개수와 그를 이용해 각 점이 무슨 Cluster에 속하는지 정보를 담는 classify 배열을 만듭니다.

- Distance

```
def dist(x, y):
    return np.linalg.norm(np.array(x)-np.array(y))
```

두 점이 주어지면 Euclidian Distance를 구하는 함수입니다.

- Get Neighbor

```
print("Calculate Neighbor ... ", end="", flush
neighbor_list = defaultdict(list)
for i in range(len(input)):
    for j in range(i+1, len(input)):
        if dist(input[i], input[j]) <= eps:
            neighbor_list[i].append(j)
            neighbor_list[j].append(i)
print("Done")
```

기존에 새롭게 추가되어지는 점이 있을 때마다 이웃을 구해주면 중복되는 시간이 많아지기에 처음 Input을 받고 각 점에 대해 Eps 안에 있는 모든 점들을 구해 준 배열을 만듭니다.

- DBSCAN iterative

```
print("Calculate DBSCAN ... ", end="", flush=True)
for idx in range(len(input)):
    if classify[idx] != -1:
        continue
    if _assign(input, idx, classify, cur_cluster, neighbor_list):
        cur_cluster += 1
print("Done")
```

Cluster를 시작하는 점을 선택합니다. 이때 Core Point나 Outlier라고 Assign 되지 않은 점을 선택합니다.

- Assign Cluster

```
def _assign(array, cidx, classify, cur, ndict):
    global minPts

    nlist = ndict[cidx]
    if len(nlist) < minPts-1:
        classify[cidx] = None
        return False

    classify[cidx] = cur
    while nlist:
        nidx = nlist.pop()

        if classify[nidx] == None:
            classify[nidx] = cur
        elif classify[nidx] == -1:
            classify[nidx] = cur
            pnew_neighbor = ndict[nidx]

            if len(pnew_neighbor) >= minPts-1:
                nlist = list(set(nlist) | set(pnew_neighbor))

    return True
```

임의로 선택된 Point의 index를 이용해 Point의 이웃 점 집합을 구합니다. 만약 해당 점 포함 집합이 MinPts를 넘지 못한다면 Outlier 또는 Border Point이므로 모르겠다는 None을 할당한 뒤 새로운 Cluster를 만들지 않으므로 False를 반환합니다. 만약 넘었다면 해당 점은 Core Point가 되고 그 점은 Clustering 됩니다 이후 그 점을 통해 얻어지는 directly density-reachable된 모든 점들을 같은 Cluster에 속하게 정해줍니다. 이때 기존 None이었다면 Border Point이므로 Assign만 해주고 -1이었다면 새로운 점의 정보이므로 그 점을 기준으로 다시 directly density reachable을 neighbor list에 넣어 줌으로써 시작 점을 통한 모든 density-reachable 점들을 Assign합니다. 이 때 시간을 줄이기 위해 새로운 Neighbor 배열을 만들 때 중복을 제거함으로써 연산을 줄입니다.

- Plot Cluster

```
unique_labels = set(classify)
print(f"{len(unique_labels)} clusters created")
colors = [plt.cm.gist_rainbow(each) for each in np.linspace(0, 1, len(unique_labels))]

for cluster_idx, col in zip(unique_labels, colors):
    if cluster_idx == None:
        col = [0, 0, 0, 1]
    class_mask = list(filter(lambda x : classify[x] == cluster_idx, range(n_points)))
    plt.plot(input_df.values[class_mask][:, 0], input_df.values[class_mask][:, 1],
        'o', markerfacecolor=tuple(col), markeredgecolor=tuple(col),
        markersize=1)
plt.show()
```

DBSCAN을 통해 구한 Cluster의 결과를 확인하기 위해 해당 점들의 좌표와 classify에 담긴 cluster 정보를 합쳐 Cluster Figure를 얻습니다.

- Data Write

```
print("Make Result File ... ", end="", flush=True)
path = "./result"
if not os.path.isdir(path):
    os.mkdir(path)
cnt = 0
for cls, num in counter:
    if cls == None:
        continue
    if cnt >= n:
        break
    li = list(filter(lambda x : classify[x] == cls, range(n_points)))
    li_df = pd.DataFrame(li)
    li_df.to_csv(path+f'/input{number}_cluster_{cnt}.txt', index=False,header=False)
    cnt += 1
print("Done")
```

Classify 배열의 결과를 통해 Cluster에 포함된 점의 개수가 많은 순으로 Output file을 만듭니다.

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

```
(project) D:\GitHub\ITE4005_HYU_2021\Assignment3>python clustering.py input2.txt 5 2 7
Calculate Neighbor ... Done
Calculate DBSCAN ... Done
7 clusters created
[(0, 640), (3, 489), (1, 386), (4, 235), (2, 192), (None, 54), (5, 4)]
Make Result File ... Done
```

주어진 clustering.py 파일을 python을 통해 compile 하면서 이때 Input 파일과 cluster 개수, Eps, MinPts를 차례대로 입력해야 합니다. 파일은 상대경로를 통해 받으므로 해당 python파일이 위치한 곳에 있는 data-3 folder 안에 input.txt가 있어야 합니다. 또한 Result File은 python 파일이 위치한 경로에 result folder가 없다면 생기는 folder 안에 생기게 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