인공지능특강(추천시스템) Project 2 Report

2022-26772 장석운

2023.04.24

Task 1. Item-based Collaborative Filtering

"ratings_train.csv" 파일에서 모든 영화에 대해 rating 한 유저들의 집합을 찾고, 각 영화 m 에 대해 rating 한 유저들은 rating 한 값을 넣고 아니면 0 으로 채워서 Item-User sparse matrix 를 만들었다. 이 Item-User sparse matrix 에서 Cosine similarity 를 계산하여 movie-movie 사이의 similarity 를 구하였다. 이를 통해 공식을 적용하여 각 유저별로 영화에 대한 추정 점수를 계산하였다.

```
            n += 1
        if diff == 0: # movie가 rating_val에는 있고 rating_train에 없으
          continue
        else:
          item_based_rmse[u] = math.sqrt((diff / n))

    print("userID 73의 item-based 기법 RMSE: ", item_based_rmse[73])
```

```
userID 73의 item-based 기법 RMSE:  0.7322648432687335
```

"Ratings_val.csv"에 존재하는 유저가 rating한 실제 rating 값과 앞서 구한 추천 점수 사이의 RMSE를 구하였다. userID 73의 실제 rating 값들과 구현한 item-based 기법에서 얻은 추천 점수 사이의 RMSE는 약 0.7322이다.

Task 2. Matrix Factorization

Task 1과 같은 방법으로 이번에는 movie의 평균 평점으로 채워서 Item-User sparse matrix를 만들었다. 그렇게 구한 Item-User sparse matrix 에 numpy.linalg.svd()를 적용하면 svd에 따라 U, s, Vt의 세 matrix가 나오게 된다.

```
[10] U, s, Vt = np.linalg.svd(iu_sparse_mat_with_avg, full_matrices=False)
     print(U.shape, s.shape, Vt.shape)

     # Singular value 중 가장 큰 K = 400개만 사용
     k = 400
     Uk = U[:,:k]
     sk = np.diag(s[:k]) # 대각 행렬로 만들어줌
     Vtk = Vt[:k,:]
     print(Uk.shape, sk.shape, Vtk.shape)
```

```
Uksk = np.matmul(Uk, sk)
matfac_recommend_score = np.matmul(Uksk, Vtk)
print(matfac_recommend_score.shape)
```

이 때, k=400으로 정의한 후 U, s, Vt에서 k만큼 slicing을 한 후 slicing한 U, s, Vt를 차례로 matmul시키면 Matrix Factorization 기반의 추천 점수를 얻을 수 있다.

```
      rmse = math.sqrt(diff / n)
      matfac_rmse[u] = rmse
print("userID 73의 matrix factorization 기법 RMSE: ", matfac_rmse[73])
```

```
userID 73의 matrix factorization 기법 RMSE:  0.7854786063906751
```

마찬가지로 "Ratings_val.csv"에 존재하는 유저가 rating한 실제 rating 값과 앞서 구한 추천 점수 사이의 RMSE를 구하였다. userID 73의 실제 rating 값들과 구현한 Matrix Factorization 기법에서 얻은 추천 점수 사이의 RMSE는 약 0.7854이다.

Task 3. Optimization

앞서 두 가지 기법 중 Matrix Factorization 기법을 기반으로 최적화를 진행해보았다. Singular Value의 개수 k를 변화시켜가며 성능을 비교해보았다. Latent factor의 개수에 따라 모델 성능이 차이가 날 수 있다고 생각하였다. 너무 적은 Singular Value를 사용하면 정보의 손실이 발생하여 전체 데이터를 잘 표현하지 못할 가능성이 있고, 반대로 너무 큰 Singular Value를 사용한다면 의미가 없는 factor가 포함되어 데이터를 factorize하는데 방해가 될 수 있을 것이라고 판단하였다. 따라서 적절한 Singular Value의 개수 k를 찾는 것이 Matrix Factorization에서 성능을 높이는데 도움이 될 수 있을 것으로 판단하였다. "ratings_val.csv"의 모든 유저에 대해서 ("ratings_train.csv"에 존재하지 않는 movie와 user는 제외) 실제 rating과 추천 점수의 RMSE를 계산하여 평균을 성능 metric으로 정하였다.

1) K = 50

```
for u in optimized_rmse:
  total_optimized_rmse += optimized_rmse[u]

print("optimized based RMSE : ", total_optimized_rmse/len(matfac_rmse))
print("matrix factorization based RMSE : ", total_matfac_rmse/len(optimized_rmse))
```

```
optimized based RMSE :  0.9521327223348579
matrix factorization based RMSE :  0.9527140378742626
```

Optimization method : 0.9521

Matrix factorization based method (k = 400) : 0.9527

2) K = 100

```
for u in optimized_rmse:
  total_optimized_rmse += optimized_rmse[u]

print("optimized based RMSE : ", total_optimized_rmse/len(matfac_rmse))
print("matrix factorization based RMSE : ", total_matfac_rmse/len(optimized_rmse))
```

```
optimized based RMSE :  0.9519133010920591
matrix factorization based RMSE :  0.9527140378742626
```

Optimization method : 0.9519

Matrix factorization based method (k = 400) : 0.9527

3) K = 200

```
for u in optimized_rmse:
  total_optimized_rmse += optimized_rmse[u]

print("optimized based RMSE : ", total_optimized_rmse/len(matfac_rmse))
print("matrix factorization based RMSE : ", total_matfac_rmse/len(optimized_rmse))
```

```
optimized based RMSE :  0.9521457639318366
matrix factorization based RMSE :  0.9527140378742626
```

Optimization method : 0.9521

Matrix factorization based method (k = 400) : 0.9527

4) K = 300

```
for u in optimized_rmse:
  total_optimized_rmse += optimized_rmse[u]

print("optimized based RMSE : ", total_optimized_rmse/len(matfac_rmse))
print("matrix factorization based RMSE : ", total_matfac_rmse/len(optimized_rmse))
```

```
optimized based RMSE :  0.9530798887819195
matrix factorization based RMSE :  0.9527140378742626
```

Optimization method : 0.9531

Matrix factorization based method (k = 400) : 0.9527

앞서 네 가지 경우에서 모두 k=400일 경우보다 성능이 근소하게 좋았다. 또한 네 가지 경우 중 k=100일 때가 가장 성능이 좋았다.

Optimization 방법에 대해 고민하면서 기타 다른 접근들 역시 고려하였다. Item-User sparse matrix를 생성하는데 평균 평점으로 대체하는 것이 아닌 중간값이다 다른 value로 대체해보기도 하였고, 유저마다 rating하는 경향이 다를 수 있으므로 그에 대한 weight를 고려하여 대체해보는 방식도 시도해보았다. 그러나 모든 경우에서 단순히 Singular Value k를 변화시키는 방법보다 성능이 안 좋았고, Task2의 matrix factorization 기법보다도 성능이 좋지 않았다. 따라서 본 과제에서는 단순히 Singular Value k를 변화시키는 방법으로 Optimization을 진행하였다.

프로젝트에 대한 피드백 의견

본 프로젝트에서 RMSE를 구할 때, "rating_val.csv"에는 있고 "rating_train.csv"에 없는 movie와 user는 계산에서 제외하였는데, 이를 해결할 수 있는 기법에 대해 궁금하였다. 즉 학습 시 사용하지 않은 movie와 user에 대해서 추천할 때에는 어떤 방식으로 해결하는지에 대한 내용도 과제로 주어진다면 추천 시스템을 이해하는데 도움이 될 것 같다.