Long Term Project (Recommender)

2016025105
컴퓨터공학과
강재훈

(i) Summary of your algorithm

주어진 데이터를 가지고 recommender을 만들기 위해 Collaborative Filtering(CF)을 이용하였습니다. CF는 취향이 비슷한 사람인 neighbor를 찾아내고 이 neighbor들이 공통적으로 높게 점수를 준 아이템을 추천하는 방식입니다. 구체적으로 3개의 step으로 구성됩니다. 첫번째 step에선 타겟 유저와 비슷한 취향을 갖는 neighbor를 찾고 두번째 step에선 이 neighbor들이 아직 내가 점수를 주지 않은 item들에 준 점수를 토대로 내가 해당 item들에 몇 점의 점수를 줄 지를 예측합니다. 마지막 step에선 점수들을 토대로 타겟 유저에게 item들을 추천합니다. 나와 비슷한 취향을 가지는 유저들을 찾기 위해 similarity를 측정해야 하는데 Pearson correlation coefficient(PCC)나 cosine similarity 등이 이용될 수 있습니다. 또한 neighbor들이 item에 준 점수들을 합산하기 위해 여러가지 방법을 사용할 수 있는데 대표적으로 단순히 평균을 내는 방법과 가중치를 주어서 평균을 내는 방법이 있습니다. 제 코드에선 similarity를 계산하기 위해 PCC를 선택했고 각각의 점수를 합산할 때는 가중치를 주어서 평균을 내는 방법을 사용하였습니다.

Pearson correlation coefficient (PCC)

$$w(a,i) = \frac{\sum_j (v_{a,j} - \bar{v}_a)(v_{i,j} - \bar{v}_i)}{\sqrt{\sum_j (v_{a,j} - \bar{v}_a)^2 \sum_j (v_{i,j} - \bar{v}_i)^2}}$$

$$(b)\ r_{c,s} = k \sum_{c' \in \hat{C}} sim(c,c') \times r_{c',s}$$

(ii) Detailed description of your codes

(모든 기능들은 1개의 클래스 내에서 구현되었습니다. (class model))

- def __init__(self, TRAINING_DATA_PATH, TEST_DATA_PATH, RESULT_FILE_PATH)

모델을 초기화 하며 training dataset, test dataset을 load 합니다. 또한 result file의 이름을 포함한 위치를 저장하여 이후 예측결과를 어디에 저장할지도 저장해둡니다..

```
def __init__(self, TRAINING_DATA_PATH, TEST_DATA_PATH, RESULT_FILE_PATH):
    self.training_set = defaultdict(lambda : defaultdict(float))
    self.test_set = []
    self.RESULT_FILE_PATH = RESULT_FILE_PATH
    with open(TRAINING_DATA_PATH, 'r' ) as train_f :
        trxs = train_f.readlines()
        for trx in trxs :
            trx = trx.strip().split('\t')
            u_id, i_id, rating = trx[0],trx[1],trx[2]
            self.training_set[u_id][i_id] = float(rating)

    with open(TEST_DATA_PATH, 'r') as test_f :
        trxs = test_f.readlines()
        for trx in trxs :
            trx = trx.strip().split('\t')
            u_id, i_id = trx[0],trx[1]
            self.test_set.append((u_id, i_id))
```

- def pearson_cor_coe(self, me, other)

me와 other의 pearson 상관계수를 계산하는 함수입니다. 만약 겹치는 item이 0개 이거나 pearson 상관계수의 계산식에서 분모가 0이 되는 경우 0을 return 합니다.

```
def pearson_cor_coe(self, me, other) :
    intersection = set()
    for item in self.training_set[me] :
        if item in self.training_set[other] :
            intersection.add(item)
    if intersection :
        mean_1 = np.mean(list(self.training_set[me].values()))
        mean_2 = np.mean(list(self.training_set[other].values()))

        sum_1, ssum_1 = 0,0
        sum_2, ssum_2 = 0,0

        for item in intersection :
            sum_1 += self.training_set[me][item] - mean_1
            ssum_1 += np.power(self.training_set[me][item]-mean_1,2)

        for item in intersection :
            sum_2 += self.training_set[other][item] - mean_2
            ssum_2 += np.power(self.training_set[other][item]-mean_2,2)

        numerator, denominator = sum_1*sum_2, np.sqrt(ssum_1*ssum_2)
        if denominator == 0 : return 0
        else : return numerator/denominator
    return 0
```

- def make_cf_row(self, me)

CF 에서 me로 들어온 user의 row를 만들어 줍니다. Neighbor들의 rating을 aggregation 하는 방법으로는 (i)에서 언급한 방법을 사용하였습니다.

```
def make_cf_row(self, me):
    user_item_rating, user_item_sim = defaultdict(int), defaultdict(int)
    for other in self.training_set :
        if me == other : continue
        sim = self.pearson_cor_coe(me, other)
        if sim > 0 :
            for i_id in self.training_set[other]:
                if i_id not in self.training_set[me] or self.training_set[me][i_id] == 0 :
                    user_item_rating[i_id] += self.training_set[other][i_id] * sim
                    user_item_sim[i_id] += sim

    my_row = {i_id : sigma/user_item_sim[i_id] for i_id, sigma in user_item_rating.items()}
    for i_id, rating in self.training_set[me].items() :
        if i_id not in my_row :
            my_row[i_id] = rating

    self.cf[me] = my_row
```

- def make_cf_row(self, me)

Test 파일에 들어있는 user를 순차적으로 탐색하며 해당 user가 CF에 없다면 해당 user의 row를 만들어줍니다. 이후 CF로 예측이 가능한 item이라면 해당 결과를 result에 넣어주고 그렇지 않다면 rating을 3으로 설정하여 result에 넣어줍니다.

```
def collaborative_filtering(self):
    self.cf = defaultdict(dict)
    self.result = []
    for trx in self.test_set :
        u_id, i_id = trx[0], trx[1]
        if u_id not in self.cf :
            self.make_cf_row(u_id)
        try :
            self.result.append( (u_id, i_id, self.cf[u_id][i_id] ) )
        except KeyError:
            self.result.append( (u_id, i_id, 3) )
```

- def save(self)

Init시 저장해둔 result file에 CF로 추천한 결과들을 써줍니다.

```
def save(self) :
    with open(self.RESULT_FILE_PATH, 'w') as f :
        for trx in self.result :
            trx_text = f"{trx[0]}\t{trx[1]}\t{trx[2]}\n"
            f.write(trx_text)
```

(iii) Instructions for compiling your source codes at TA's computer

python version == Python 3.9.5

실행법 == python recommender.py u#.base u#.test

1. Base파일과 test파일 모두 recommender.py와 동일한 위치에 있어야 합니다.
2. 결과파일은 recommender.py가 있는 위치에 생성됩니다.

테스트실행법 == PA4.exe u#

(iii) Any other specification of your implementation and testing