<협업 필터링(Collaborative Filtering)>

* 종류

1.최근접 이웃 기반

: 사용자가 평가하지 않은 item 예측

1. User-based : 비슷한 성향을 가진 사람이 선택한 item 추천
2. **Item-based** : 과거 구매 item과 유사한 item 추천

2. 잠재 요인 기반 Matrix Factorization

: user-item 행렬 분해 -> latent factor 추출

latent factor 기반으로 평가하지 않은 item 예측

장점 : 계산 효율성, 저장 공간 절약

단점 : 행렬 분해에 의해 추출되는 '잠재 요인'이 정확히 어떤 것인지는 알 수 없다. 대략 우리의 데이터셋은 동아리별 키워드 (음악, 취미, 자기계발)이 영향을 끼쳤을 것으로 예상.

<사용한 모델>

* **SVD**
  + latent factor 통해 평점 없는 item의 평점 예측
* **SVD++**
  + SVD에 implicit feedback까지 반영
* **NMF(Non-negative Matrix Factorization)**
  + 행렬 분해시, 음수를 포함하지 않는 행렬이 되도록 함 -> 분해 결과 행렬 찾기가 쉬움
* 한계

1. Cold Start : 신규 사용자, 신규 item에 대한 추천이 어려움
2. Long Tail : 사용자들이 많은 호감도를 보이는 소수의 item이 추천 item으로 보이는 비율이 높음 (비대칭적 쏠림 현상), 관심이 저조한 item은 추천되지 못함

3. Neural Collaborative Filtering

: MF는 linear한 방식으로, 복잡한 구조를 표현하는 데에 한계

다만, NCF 적용을 위해서 필요한 문장 데이터가 없는 관계로, 하나하나 생성해야 한다는 문제

우리 방향에 적합한지 의문

2007년 넷플릭스 분석 경연대회에 참가한 벨코의 실용적 혼돈팀이 가장 먼저 사용한 알고리즘은 군집 분석의 일종인 k-NN이었다. 이후 RBM, SVD를 사용, 혼합 모델에는 선형 회귀 방법을 사용해서 마지막 결과물 제출했다. K-NN 알고리즘은 서로 근접해 있는 그룹들에 대해서는 예측력이 높으나, 멀리 떨어져 있는 그룹에 대한 예측력은 떨어지기 때문에 이를 보완하기 위해 SVD를 사용하였다. K-NN, SVD, 혼합 모델 순으로 예측력이 높았다. 2008년에는 SVD를 빼고 NNMF 알고리즘을 추가해서 차원축소를 하여 혼합 모델을 만들었다. 마지막으로 제출한 모델은 700개가 넘는 예측 알고리즘 모델을 GBDT 알고리즘으로 혼합한 모델이었다. 하지만 현업 운영을 고려하면 너무 많은 알고리즘을 혼합하는 것은 적합하지 않다. 몇 개의 알고리즘만 사용한 단순한 모델도 많은 알고리즘을 사용한 모델의 90% 정도는 예측할 수 있다.

투입한 변수가 많을수록 RMSE 값이 작아진다.

단순히 모델의 오차를 나타내는 RMSE만 평가하는 것은 현업에 적용할 모델을 평가하는 데는 적당하지 않다. 지나치게 정교화, 특수화되어 과적합 문제가 존재하기 때문이다.

2009년에 우승한 벨코의 실용적 혼돈 팀은 다른 팀과의 협업을 통해 데이터 현황과 사용자 및 영화에 대한 속성을 더 깊게 이해함으로써 모델의 예측력를 높일 수 있었다. 결국 경험과 비즈니스 영역에 대한 지식이 중요하다는 것을 입증한 셈이다.

**<콘텐츠 기반 필터링(Contents-based Filtering)>**

협업 필터링의 문제 해결

* 한계

1. 메타 정보의 한정성으로 인해 정밀성 저하

다른 머신러닝 기법을 사용해서 예측 가능

=> but, 그럴경우 방향성에 맞는지? Sparse matrix라서 정확도가 높지 않을 것으로 보임

<하이브리드(Hybrid) 추천시스템>

신규 콘텐츠는 콘텐츠 기반 필터링, 충분한 데이터가 쌓인 후에는 협업 필터링

=> 현재 두 방법을 모두 사용한 것으로, 이 장점을 충분히 살린 것으로 보임

<머신러닝 추천시스템>

사용자에게 추천 후보군을 보여준 후, 그에 대한 반응을 학습

=> 현재 하고 있는 것과 매우 유사

최근접 이웃 기반 대신 넷플릭스 대회에서 우승한 잠재요인기반 모델 등장 후 대부분 이 알고리즘을 사용. 따라서 우리 코드도 잠재요인기반 알고리즘 구현을 먼저 시도해봄.

하지만 아마존의 경우 아직 최근접 이웃 기반의 item-based 모델을 사용 중이므로 최근접 이웃 기반 알고리즘도 시도하고 비교해 볼 예정. 고객기반 알고리즘보다 아이템 기반 알고리즘의 정확도가 주로 높게 나오므로 이를 염두에 두고 비교해보자. (비슷한 고객을 토대로 추천하는 것보다 비슷한 아이템을 토대로 추천하는 것이 일반적으로 정확도가 높다고 한다.)

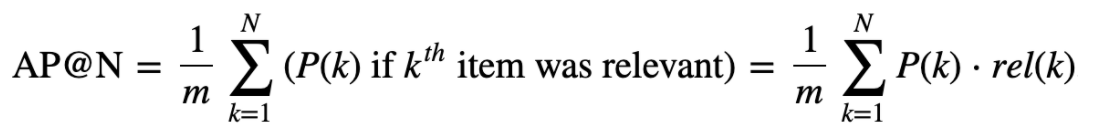
<결과물 및 해석 방안>

1. Long-tail plot (가능)

: 평점수 ~ item 그래프를 통해 인기있는 item 확인 가능

보통, 극히 일부의 item만 많은 평점수를 가짐

1. MAP@K (rel k에 대한 결과가 없어서 어려울 것으로 보임)

: AP의 평균 

* N : 전체 제품 개수가 예를 들어 2N개 있을 때, 몇 개의 제품을 사용자에게 추천해줄지, N값이 곧 cuttoff 'K'의 'K'가 된다고 보면 된다.
* m : 실제로 유저의 취향에 맞는 제품들 개수(즉, 추천 시스템에서 일종의 '정답'이 되겠다)
* k : N개 중 k번째로 추천된 제품
* P(k) : k번째 추천 제품의 Precision 값
* rel(k) : binary값으로, k번째 추천된 제품이 실제 유저의 취향 제품과 동일하면 1, 아니면 0

3. Coverage

: **Train 데이터에 기반해 도출된 Ranked list에 있는 추천 제품들이 Test 데이터에서도 추천 제품인 퍼센티지**

4. Personalization (가능)

: 사용자들의 추천 list간 dissimilarity를 구함

user X item matrix -> user 간 코사인 유사도 -> (1- upper triangle 평균값) 높으면 user 간 추천 item이 다름. 즉, 개인화된 정보 제공

5. Intra Similarity

: **제품들 간의 코사인 유사도 행렬 -> 평균 -> 각 유저 평균**

**이 값이 높다면 추천 시스템이 각 유저에게 매우 비슷한 성격의 제품들을 추천 해주고 있다는 것이며 결국 추천 시스템의 성능이 좋다는 것을 의미한다.**

<https://techblog-history-younghunjo1.tistory.com/133>

이페이지 참고!