Soongsil University

FISM : Factored Item Similarity Models for Top – N Recommender System

CheolHee Jung

DM Lab

05.23.2024

Santosh Kabbur, Xia Ning, George Karypis

KDD '13

Outline

- Introduction
- Motivation
- Proposed Method
- Experiments
- Conclusion

Top – N Recommender Systems

- Collaborative Filtering
  - 이웃 기반 방법
    - 사용자 또는 아이템 간의 유사성을 계산하여 추천
  - 모델 기반 방법
    - 행렬 분해 ex. SVD
- Content – Based Filtering
  - 아이템의 속성을 사용하여 추천, 사용자가 과거에 선호했던 아이템들의 속성을 분석하여 유사한 속성을 가진 아이템 추천
- However, **데이터가 희소할 경우 성능이 저하됨**

Sparse Linear Methods(SLIM)

- **SLIM**
  - 기존 아이템 기반 이웃 협업 필터링을 개선하여 데이터로부터 직접 희소행렬 학습, 구체적으로는 SLIM은 아이템 아이템 유사성 행렬 S(희소행렬)를 학습하여 이를 통해 추천
- **S 행렬 학습**
  - $min_s \frac{1}{2}\|R - RS\|_F^2 + \frac{\beta}{2}\|S\|_F^2 + \lambda\|S\|_1$
  - $s.t\ S \geq 0, diag(S) = 0$
    - R과 RS의 차이를 최소화하여 예측의 정확성을 높임
    - $\frac{\beta}{2}\|S\|_F^2$ 항을 통해 행렬 S의 크기를 제어하여 과적합 방지
    - $\lambda\|S\|_1$ 행렬 S의 희소성을 유도
    - $diag(S) = 0$ 아이템이 자기 자신을 추천하지 않도록
    - 비음수성 제약 : 행렬 S의 모든 요소가 음수가 되지 않도록, 가중치가 양수일때만 의미가 있음을 보장

- However, SLIM은 일부 사용자들에 의해 공동 구매/평가된 아이템 간의 관계만 모델링함 이에 따라 희소한 데이터셋에서 좋은 성능을 위해 필요한 아이템 간의 전이 관계를 포착할 수 없다.

Neighborhood Singular Value Decomposition(NSVD)

❒ NSVD

- 사용자 아이템 평점 행렬의 잠재 요인을 학습하여 사용자에게 아이템을 추천하는 방법으로 SVD를 기반(두 개의 저차원 행렬의 곱)으로 한 협업 필터링

  - 각 사용자와 아이템은 잠재 요인 벡터로 표현
    - 사용자 u는 잠재 요인 벡터 p_u로 아이템 i는 잠재요인 벡터 q_i로 표현
  - 평점 예측 : 사용자 u가 아이템 i에 대해 줄 평점
    - $\hat{r}_{ui} = b_u + b_i + \sum_{j \in R_u^+} p_j^T q_i$
    - 즉 두 저차원의 행렬곱으로 아이템 간 유사성 학습

❒ However, 여전히 희소 데이터 문제 존재함

Outline

- Introduction
- Motivation
- Proposed Method
- Experiments
- Conclusion

Motivation

- 현실 세계에서, 사용자는 수백만 개의 아이템 중 극히 일부에만 피드백을 제공하기 때문에 사용자 아이템 행렬이 매우 희소해지는 결과를 초래
- SLIM 과 같은 방법은 공동 평가되지 않은 아이템 간의 의존성 포착하지 못함
- 이를 해결하기 위해 FISM은 NSVD및 SVD++와 유사한 팩터화된 아이템 유사성 모델을 사용하여 관계 학습

Outline

- Introduction
- Motivation
- Proposed Method
- Experiments
- Conclusion

FISM

- **Factored Item Similarity Methods**
  - **FISM에서 사용자 u가 평가하지 않은 아이템i에 대한 추천 점수는 u가 평가한 아이템과 P에서 학습된 p_j 백터와 Q에서 학습된 q_i 벡터의 곱의 집합으로 계산됨**
    - $\hat{r}_{ui} = b_u + b_i + (n_u^+)^{-\alpha} \sum_{j \in R_u^+} p_j q_i^T$

- **FISM에서 사용자가 평가하지 않은 아이템에 대한 추천 점수는 학습된 잠재 요인 벡터의 곱을 통해 계산됨**

- **FISM은 SLIM의 아이템 - 아이템 행렬을 직접 학습하는 아이디어 채택, NSVD의 두 개의 저차원 행렬 곱 아이디어 채택**

FISMrmse

- FISMrmse
  - 사용자 u가 평가하지 않은 아이템 i에 대한 추천 점수를 계산하기 위해 제곱 오차 손실 함수를 사용함
    - $L(\cdot) = \sum_{i \in D} \sum_{u \in C} (r_{ui} - \hat{r}_{ui})^2$
  - 최적화 문제를 최소화 하여 행렬 P와 Q를 학습함
    - $$\underset{\mathbf{P},\mathbf{Q}}{\text{minimize}} \ \frac{1}{2} \sum_{u \in \mathcal{C}} \sum_{i \in \mathcal{R}_u^+} \|r_{ui} - \hat{r}_{ui}\|_F^2 + \frac{\beta}{2} (\|\mathbf{P}\|_F^2 + \|\mathbf{Q}\|_F^2)$$
    - 최적화 문제는 확률적 경사 하강법(SGD) 알고리즘을 사용하여 해결됨
  - 0 항목 샘플링 전략
    - FISMrmse의 학습 과정에서 계산을 줄이기 위해, 평가되지 않은 항목(0 항목)을 샘플링하여 최적화에 사용

FISMrmse

❒ 알고리즘 1

Algorithm 1 FISMrmse:Learn.

procedure FISMrmse_LEARN
  $\eta \leftarrow$ learning rate
  $\beta \leftarrow \ell_F$ regularization weight
  $\rho \leftarrow$ sample factor
  $iter \leftarrow 0$
  Init $\mathbf{P}$ and $\mathbf{Q}$ with random values in (-0.001, 0.001)

  while $iter < maxIter$ or error on validation set decreases **do**
    $\mathcal{R}' \leftarrow \mathbf{R} \cup SampleZeros(\mathbf{R}, \rho)$
    $\mathcal{R}' \leftarrow RandomShuffle(\mathcal{R}')$

    for all $r_{ui} \in \mathcal{R}'$ **do**
      $\mathbf{x} \leftarrow (n_u^+ - 1)^{-\alpha} \sum_{j \in \mathcal{R}_u^+ \setminus \{i\}} \mathbf{p}_j$

      $\tilde{r}_{ui} \leftarrow b_u + b_i + \mathbf{q}_i^\top \mathbf{x}$
      $e_{ui} \leftarrow r_{ui} - \tilde{r}_{ui}$
      $b_u \leftarrow b_u + \eta \cdot (e_{ui} - \lambda \cdot b_u)$
      $b_i \leftarrow b_i + \eta \cdot (e_{ui} - \gamma \cdot b_i)$
      $\mathbf{q}_i \leftarrow \mathbf{q}_i + \eta \cdot (e_{ui} \cdot \mathbf{x} - \beta \cdot \mathbf{q}_i)$

      for all $j \in \mathcal{R}_u^+ \setminus \{i\}$ **do**
        $\mathbf{p}_j \leftarrow \mathbf{p}_j + \eta \cdot (e_{ui} \cdot (n_u^+ - 1)^{-\alpha} \cdot \mathbf{q}_i - \beta \cdot \mathbf{p}_j)$
      end for
    end for
    $iter \leftarrow iter + 1$
  end while

  return $\mathbf{P}, \mathbf{Q}$
end procedure

FISMauc

- FISMauc는 FISM의 또 다른 변형으로, 순위 오류 기반 손실 함수로 아이템을 올바른 순서로 정렬하는 데 중점을 둔다
  - $L(\cdot) = \sum_{u \in C} \sum_{i \in R_u^+} \sum_{j \in R_u^-} ((r_{ui} - r_{uj}) - (\hat{r}_{ui} - \hat{r}_{uj}))^2$

- 최적화 문제를 최소화하여 행렬 P와 Q를 학습함
  - $$\underset{\mathbf{P},\mathbf{Q}}{\text{minimize}} \ \frac{1}{2} \sum_{u \in \mathcal{C}} \sum_{i \in \mathcal{R}_u^+, j \in \mathcal{R}_u^-} \|(r_{ui} - r_{uj}) - (\hat{r}_{ui} - \hat{r}_{uj})\|_F^2 + \frac{\beta}{2}(\|\mathbf{P}\|_F^2 + \|\mathbf{Q}\|_F^2) + \frac{\gamma}{2}(\|\mathbf{b_i}\|_2^2)$$

FISMauc

❒ 알고리즘 2

Algorithm 2 FISMauc:Learn.

procedure FISMauc_LEARN
  $\eta \leftarrow$ learning rate
  $\beta \leftarrow \ell_F$ regularization weight
  $\rho \leftarrow$ number of sampled zeros
  $iter \leftarrow 0$
  Init $\mathbf{P}$ and $\mathbf{Q}$ with random values in (-0.001, 0.001)

  while $iter < maxIter$ or error on validation set decreases **do**
    for all $u \in \mathcal{C}$ **do**
      for all $i \in \mathcal{R}_u^+$ **do**
        $\mathbf{x} \leftarrow 0$
        $\mathbf{t} \leftarrow (n_u^+ - 1)^{-\alpha} \sum_{j \in \mathcal{R}_u^+ \setminus \{i\}} \mathbf{p}_j$
        $\mathcal{Z} \leftarrow SampleZeros(\rho)$

        for all $j \in \mathcal{Z}$ **do**
          $\tilde{r}_{ui} \leftarrow b_i + \mathbf{t} \cdot \mathbf{q}_i^\top$
          $\tilde{r}_{uj} \leftarrow b_j + \mathbf{t} \cdot \mathbf{q}_j^\top$
          $r_{uj} \leftarrow 0$
          $e \leftarrow (r_{ui} - r_{uj}) - (\tilde{r}_{ui} - \tilde{r}_{uj})$
          $b_i \leftarrow b_i + \eta \cdot (e - \gamma \cdot b_i)$
          $b_j \leftarrow b_j - \eta \cdot (e - \gamma \cdot b_j)$
          $\mathbf{q}_i \leftarrow \mathbf{q}_i + \eta \cdot (e \cdot \mathbf{t} - \beta \cdot \mathbf{q}_i)$
          $\mathbf{q}_j \leftarrow \mathbf{q}_j - \eta \cdot (e \cdot \mathbf{t} - \beta \cdot \mathbf{q}_j)$
          $\mathbf{x} \leftarrow \mathbf{x} + e \cdot (\mathbf{q}_i - \mathbf{q}_j)$
        end for
      end for

      for all $j \in \mathcal{R}_u^+ \setminus \{i\}$ **do**
        $\mathbf{p}_j \leftarrow \mathbf{p}_j + \eta \cdot (\frac{1}{\rho} \cdot (n_u^+ - 1)^{-\alpha} \cdot \mathbf{x} - \beta \cdot \mathbf{p}_j)$
      end for
    end for

    $iter \leftarrow iter + 1$
  end while

  return $\mathbf{P}, \mathbf{Q}$
end procedure

Outline

- Introduction
- Motivation
- Proposed Method
- Experiments
- Conclusion

Experiments

- Dataset
  - 각 데이터 세트에 대해 희소성 수준이 다른 세 가지 버전
    - FISM의 희소한 데이터 세트에서의 성능을 평가하기 위해

Table 1: Datasets.

Dataset	#Users	#Items	#Ratings	Rsize	Csize	Density
ML100K-1	943	1,178	59,763	63.99	50.73	5.43%
ML100K-2	943	1,178	39,763	42.57	33.75	3.61%
ML100K-3	943	1,178	19,763	21.16	16.78	1.80%
Netflix-1	6,079	5,641	429,339	70.63	76.11	1.25%
Netflix-2	6,079	5,641	221,304	36.40	39.23	0.65%
Netflix-3	6,079	5,641	110,000	18.10	19.50	0.32%
Yahoo-1	7,558	3,951	282,075	37.32	71.39	0.94%
Yahoo-2	7,558	3,951	149,050	19.72	37.72	0.50%
Yahoo-3	7,558	3,951	75,000	9.92	18.98	0.25%

The "#Users", "#Items" and "#Ratings" columns are the number of users, items and ratings respectively, in each of the datasets. The "Rsize" and "Csize" columns are the average number of ratings for each user and for each item (i.e., row and column density of the user-item matrix), respectively, in each of the datasets. The "Density" column is the density of each dataset (i.e., density = #Ratings/(#Users × #Items)).

Experiments

❒ Evaluation Methodology

- **모델 성능 평가**
  - **5 fold Leave-One-Out-Validation 사용**

- **추천 품질 평가**
  - **Hit Rate와 Average Reciprocal Hit Rank 사용**
    - $HR = \frac{\#hits}{\#users}$
    - **테스트 세트에 있는 사용자가 추천 목록에서 테스트 아이템을 성공적으로 회상한 비율**
    - $ARHR = \frac{1}{\#users}\sum_{i=1}^{\#hits}\frac{1}{pos_i}$
    - **HR의 가중 버전으로, 추천된 아이템의 순위를 고려하여 품질을 측정**

- **ItemKNN, PureSVD, BPRkNN, BPRMF, SLIM등과 비교**

Experiments

❒ Effect of Neighborhood Agreement

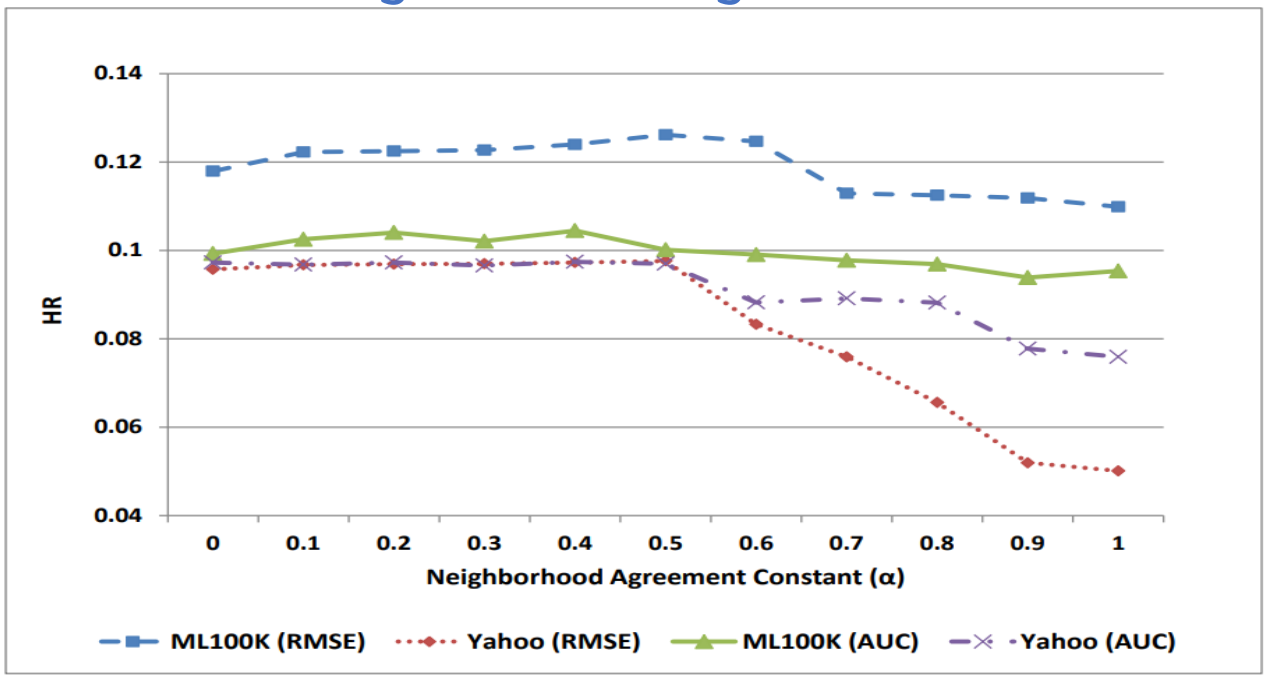


Figure 1: Effect of neighborhood agreement on performance.

- α값이 0.4에서 0.5범위 일 때 가장 좋은 성능
  - 추천되는 아이템이 높은 평점을 받기 위해서는 많은 이웃 아이템이 높은 유사성을 가져야 함을 의미

Experiments

❒ Performance of Induced Sparsity on S

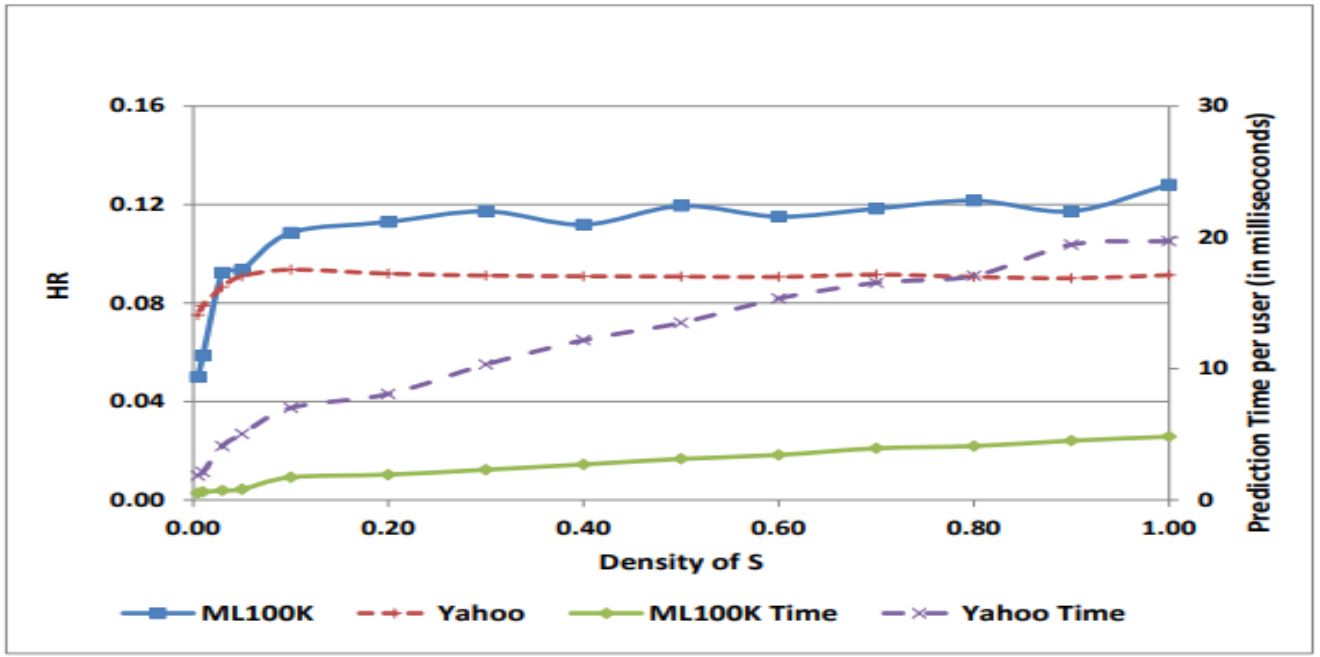
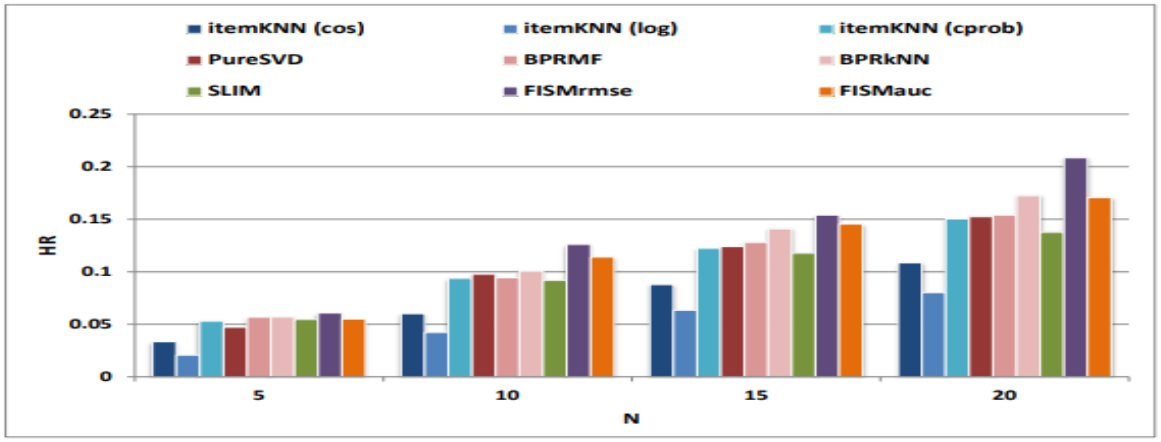


Figure 2: Performance of induced Sparsity on S.

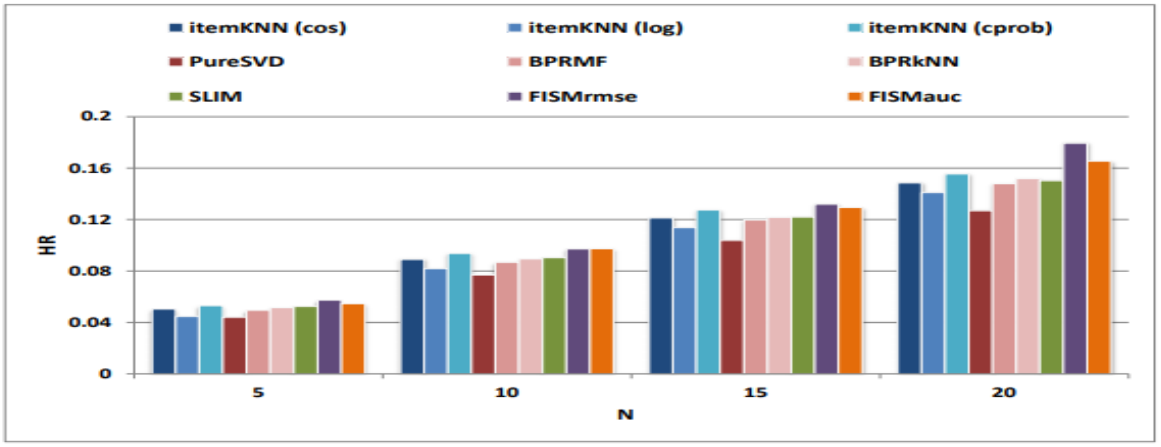
- 밀도가 0.1에서 0.15 범위일 때 추천 성능의 감소가 최소화되는 반면, 추천 계산 시간은 크게 줄어듬
  - 추천 효율성을 높이는 데 작은 성능 손실만을 감수하면서도 유용할 수 있음

Experiments

- Comparison With Other Approaches



(a) ML100K dataset.



(b) Yahoo dataset.

Figure 5: Performance for different values of N.

Outline

- ❒ Introduction
- ❒ Motivation
- ❒ Proposed Method
- ❒ Experiments
- ❒ Conclusion

Conclusion

- 데이터셋의 희소성이 증가할수록 FISM의 상대적 성능이 더욱 향상되었고, FISM의 추정 접근법이 NSVD와 SVD++와 같은 기존 방법보다 우수함

- Weak Points

  - FISM은 SGD를 사용하여 최적화를 수행하는데, 각 반복(iteration)마다 행렬의 업데이트가 필요하기 때문에 시간이 많이 소요

  - FISM은 아이템 임베딩에 집중하므로, 사용자 개별의 선호도를 충분히 반영하지 못할 수 있음

Thank you!