FISM: Factored Item Similarity Models for Top-N Recommender Systems

By Santosh Kabbur, Xia Ning, and George Karypis

Presented by Brandon Norick

Overview

- Introduction
- Preliminaries
- Related work
- FISM
- Results
- Conclusions

Introduction

- Information overload – services like Amazon, Yelp, Netflix, and Last.fm give users access to a huge amount of information and product choices. They can't possibly process it all, so how can we help?

Recommender Systems

- Recommenders provide an alternative to search as another way for users to find items of interest
- There are many varieties of recommenders
  - Top-N recommendation vs rating prediction
  - Collaborative filtering (CF) vs content based recommenders

Recommender Systems

- For CF, models using matrix factorization are the state-of-the-art
- Sparsity of the data set is a key challenge
  - In real world datasets there are many items and typical users only provide ratings on a very small subset
  - Many methods rely on the co-rating relationship to learn item similarities

Preliminaries

- C and D represent the sets of user and items, respectively
  - $|C| = n$, $|D| = m$
- $\boldsymbol{R}$ is the n x m user-item implicit feedback matrix, a binary matrix having r_{ui} as 1 if user u has provided feedback on item i and 0 otherwise

Related Work

- SLIM (Sparse Linear Models) learns a sparse matrix ***S***, the aggregation coefficient matrix, and recommendation scores for a user are obtained using the equation

$$\tilde{\mathbf{r}}_u = \mathbf{r}_u \mathbf{S}$$

- While similar in nature to a traditional item-based nearest neighbor approach, SLIM learns ***S*** from the data by minimizing the following

$$\underset{\mathbf{S}}{\text{minimize}} \ \frac{1}{2}\|\mathbf{R} - \mathbf{R}\mathbf{S}\|_F^2 + \frac{\beta}{2}\|\mathbf{S}\|_F^2 + \lambda\|\mathbf{S}\|_1$$

$$\text{subject to} \ \mathbf{S} \geq 0 \ , \ \ \text{diag}(\mathbf{S}) = 0$$

Related Work

- A few properties of SLIM
  - $\boldsymbol{S}$ >= 0 ensures only positive aggregations
  - diag($\boldsymbol{S}$) = 0 constraint ensures that r_{ui} is not used to compute the recomendation score for item i
  - the L1-norm of $\boldsymbol{S}$ is used to learn a sparse matrix
- SLIM can only learn relationships between items which have been co-rated, and therefore misses the transitive nature of such relations

Related Work

- NSVD learns an item-item similarity matrix
  - Low rank matrices *P* and *Q* are learned to approximate the similarity between items as a product of their latent features
  - Ratings are predicted using

$$\hat{r}_{ui} = \tilde{r}_{ui} = b_u + b_i + \sum_{j \in \mathcal{R}_u^+} \mathbf{p}_j \mathbf{q}_i^\mathsf{T}$$

  - *P* and *Q* are learned by optimizing

$$\underset{\mathbf{P},\mathbf{Q}}{\text{minimize}} \ \frac{1}{2} \sum_{u \in \mathcal{C}} \sum_{i \in \mathcal{R}_u^+} \|r_{ui} - \hat{r}_{ui}\|_F^2 + \frac{\beta}{2} (\|\mathbf{P}\|_F^2 + \|\mathbf{Q}\|_F^2)$$

FISM

- Motivation: overcome the sparsity related issue inherent in SLIM by incorporating a matrix factorization model for item similarity similar to NSVD
  - Projection into a latent space helps FISM to learn transitive relationships implicitly
- Recommendation score is computed with

$$\tilde{r}_{ui} = b_u + b_i + (n_u^+)^{-\alpha} \sum_{j \in \mathcal{R}_u^+} \mathbf{p}_j \mathbf{q}_i^T$$

FISM

- The term $(n_u^+)^{-\alpha}$ controls the degree of agreement between items rated by user *u* with regard to their similarity to item *i*
  - Alpha is a dataset dependent parameter set empirically between 0 and 1, at the extremes

$$\alpha = 0$$

$$\tilde{r}_{ui} = b_u + b_i + \sum_{j \in \mathcal{R}_u^+} \mathbf{p}_j \mathbf{q}_i^\top$$

$$\alpha = 1$$

$$\tilde{r}_{ui} = b_u + b_i + \frac{\sum_{j \in \mathcal{R}_u^+} \mathbf{p}_j \mathbf{q}_i^\top}{n_u^+}$$

FISM

- Two variations
  - FISMrmse: loss computed using RMSE error loss function

$$\mathcal{L}(\cdot) = \sum_{i \in \mathcal{D}} \sum_{u \in \mathcal{C}} (r_{ui} - \hat{r}_{ui})^2$$

  - FISMauc: ranking error based loss function based on Bayesian Personalized Ranking

$$\mathcal{L}(\cdot) = \sum_{u \in \mathcal{C}} \sum_{i \in \mathcal{R}_u^+, j \in \mathcal{R}_u^-} ((r_{ui} - r_{uj}) - (\hat{r}_{ui} - \hat{r}_{uj}))^2$$

FISM

- Estimated value of r_{ui} explicitly excludes the current item, *i*

$$\hat{r}_{ui} = b_u + b_i + (n_u^+ - 1)^{-\alpha} \sum_{j \in \mathcal{R}_u^+ \setminus \{i\}} \mathbf{p}_j \mathbf{q}_i^\mathsf{T}$$

- Important difference between FISM and NSVD

FISM

- Stochastic Gradient Descent is used to learn *P* and *Q* for both variations
  - FISMrmse optimizes

$$\underset{\mathbf{P},\mathbf{Q}}{\text{minimize}} \ \frac{1}{2}\sum_{u,i\in R}\|r_{ui}-\hat{r}_{ui}\|_F^2+\frac{\beta}{2}(\|\mathbf{P}\|_F^2+\|\mathbf{Q}\|_F^2) + \frac{\lambda}{2}\|\mathbf{b_u}\|_2^2+\frac{\gamma}{2}\|\mathbf{b_i}\|_2^2$$

  - FISMauc optimizes

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

Evaluation Methodology

- 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%

Evaluation Methodology

- 5-fold Leave-One-Out-Cross-Validation
  - For each fold, select one item per user to exclude from the training data
  - Using the training data, learn the models
  - Predict a top-N list for each user
  - Measure performance based on whether (and where) the excluded item for each user appears in their list

$$HR = \frac{\#hits}{\#users}$$

$$ARHR = \frac{1}{\#users} \sum_{i=1}^{\#hits} \frac{1}{pos_i}$$

Baselines

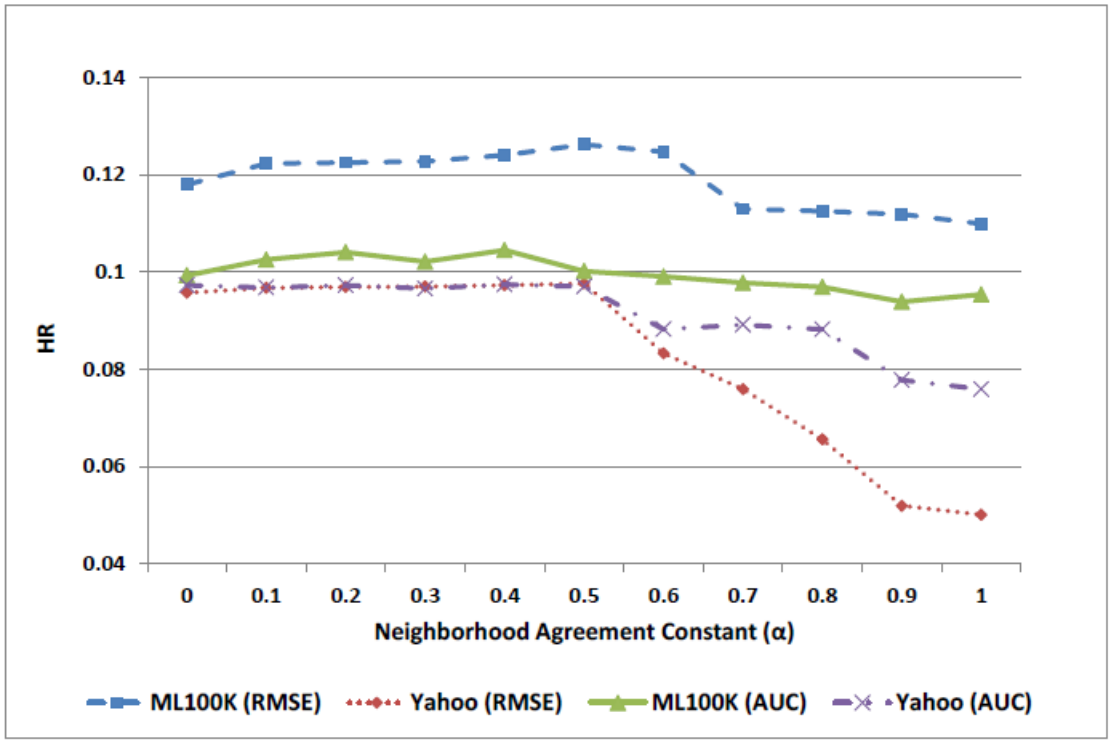
- ItemKNN (cos)
- ItemKNN (cprob)
- ItemKNN (log)
- PureSVD
- BPRkNN
- BPRMF
- SLIM

Effect of Bias

Scheme	ML100K				Yahoo			
	Beta	Lambda	Gamma	HR	Beta	Lambda	Gamma	HR
NoBias	8e-4	-	-	0.1281	2e-5	-	-	0.0974
UserBias	6e-4	0.1	-	0.1336	4e-5	0.1	-	0.1012
ItemBias	2e-4	-	0.01	0.1401	4e-5	-	1e-4	0.1007
User&ItemBias	6e-4	0.1	1e-4	0.1090	4e-5	0.1	1e-4	0.0977

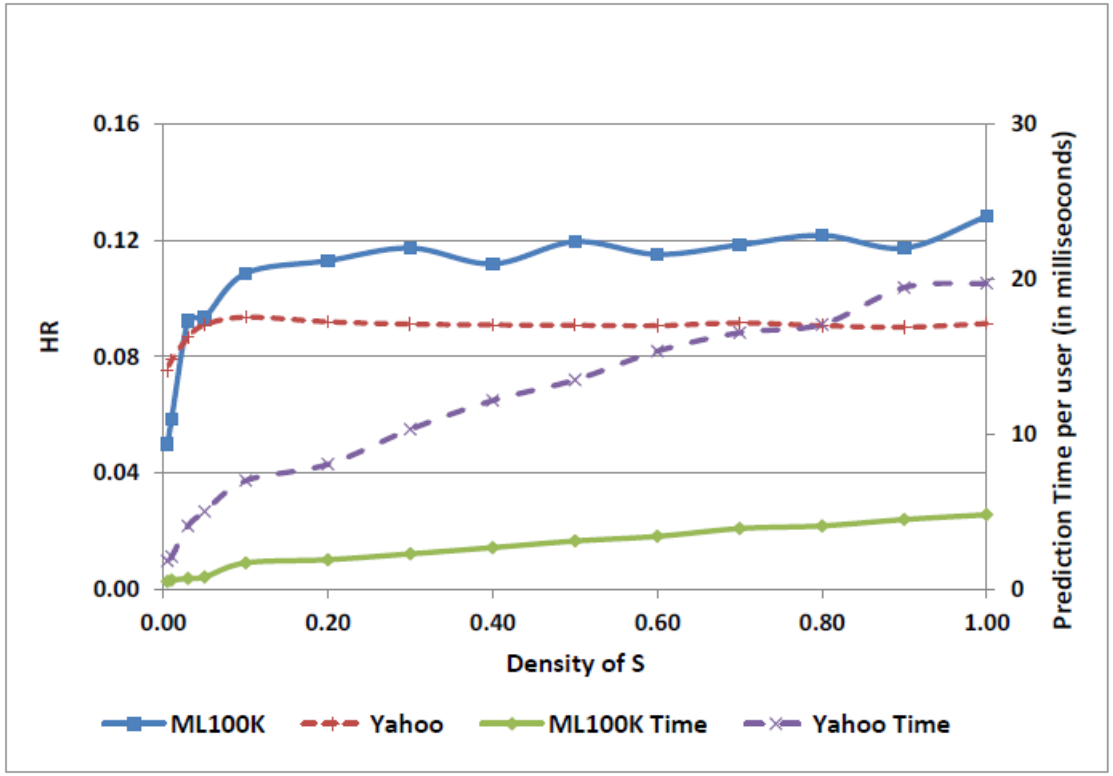
- Biases affect performance
- Item bias has more affect on performance

Effect of Neighborhood Agreement



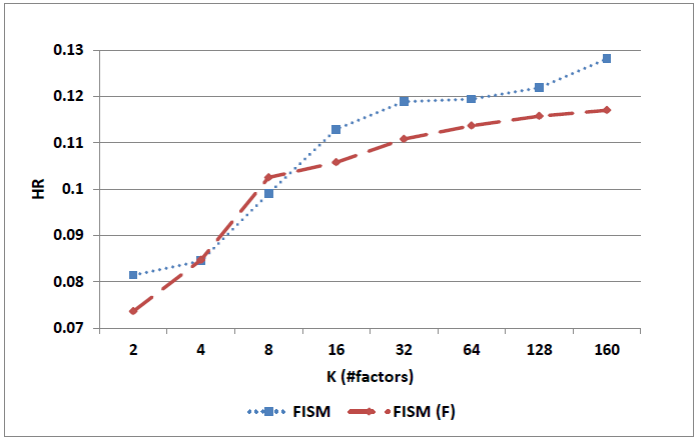
- Alpha between 0.4 and 0.5 seems to yield the best performance
- FISMauc more stable than FISMrmse because user bias is nullified

Performance on Sparse Data

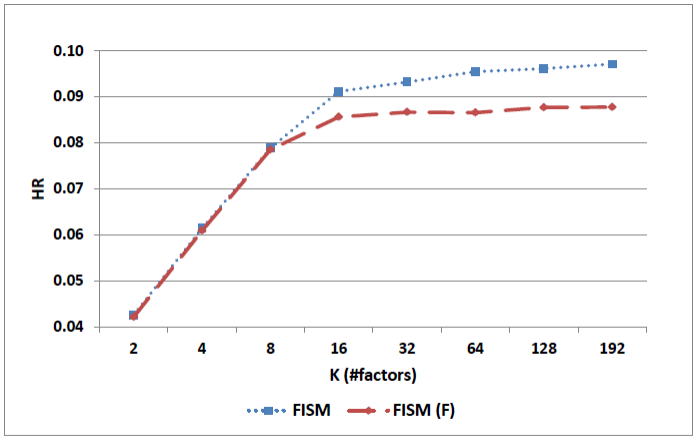


- Minimal reduction in performance up to density in the range of 0.1 to 0.15
- Average time to compute recommendations for each user is drastically reduced

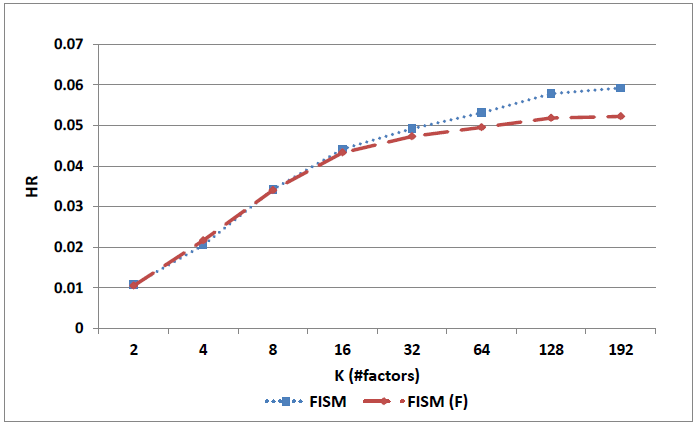
Effect of Estimation Approach



ML100K dataset.



Yahoo dataset.



Netflix dataset.

- Excluding the item under consideration yields better results as *k* increases

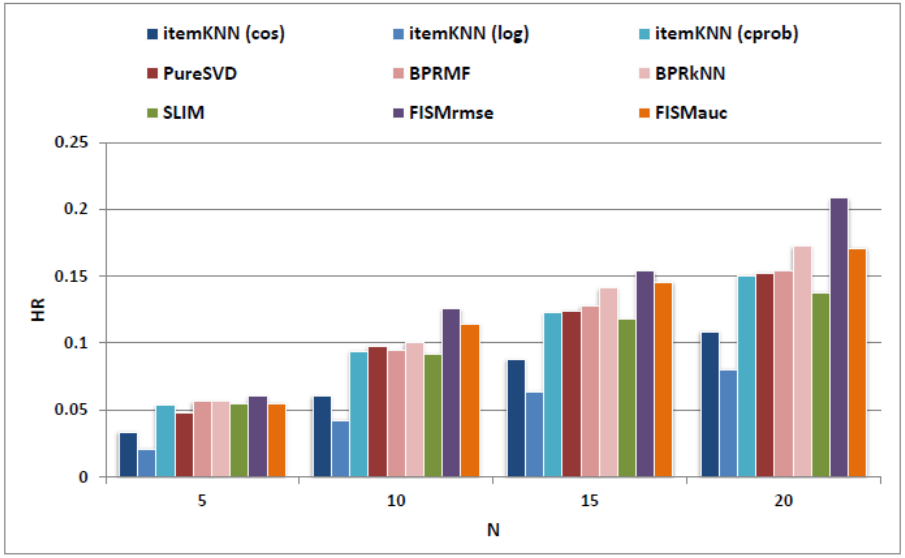
Comparison with Baselines

Method	ML100K-1					ML100K-2					ML100K-3				
	Params			HR	ARHR	Params			HR	ARHR	Params			HR	ARHR
ItemKNN (cos)	100	-	-	0.1604	0.0578	100	-	-	0.1214	0.0393	100	-	-	0.0602	0.0193
ItemKNN (log)	100	-	-	0.1047	0.0336	100	-	-	0.0809	0.0250	100	-	-	0.0424	0.0116
ItemKNN (cprob)	500	0.6	-	0.1711	0.0581	500	0.3	-	0.1308	0.0440	400	0.1	-	0.0938	0.0293
PureSVD	10	-	-	0.1700	0.0594	10	-	-	0.1362	0.0438	5	-	-	0.0438	0.0316
BPRkNN	1e-4	0.01	-	0.1621	0.0564	1e-5	0.01	-	0.1272	0.0447	1e-5	14	-	0.1006	0.0319
BPRMF	400	0.1	-	0.1610	0.0512	700	0.1	-	0.1224	0.0407	700	0.25	-	0.0943	0.0305
SLIM	0.1	20	-	0.1782	0.0620	0.01	18	-	0.1283	0.0448	1e-4	14	-	0.0919	0.0303
FISMrmse	96	2e-5	0.001	0.1908	0.0641	64	8e-4	0.01	0.1482	0.0462	96	8e-4	0.001	0.1260	0.0384
FISMauc	64	0.001	1e-4	0.1518	0.0504	144	2e-5	5e-5	0.1304	0.0424	144	8e-5	1e-5	0.1140	0.0340

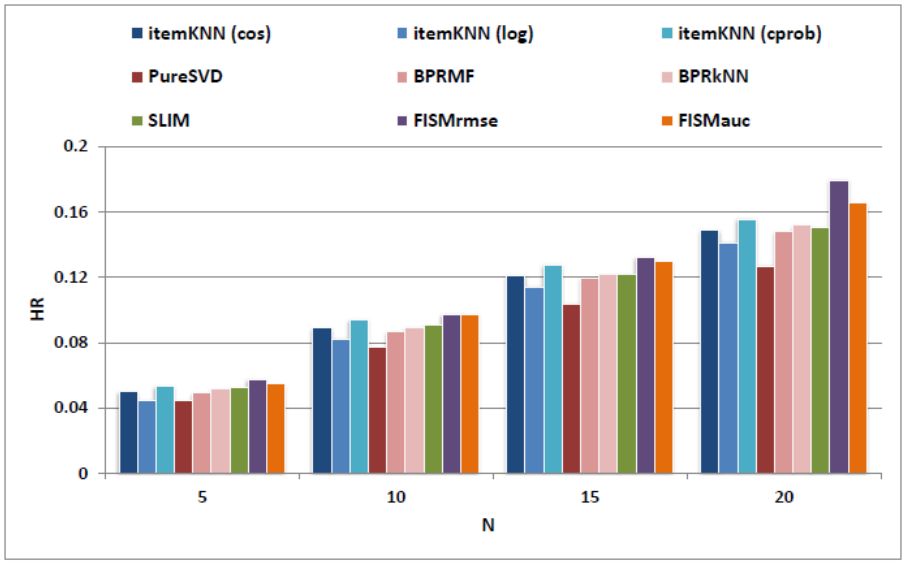
Method	Netflix-1					Netflix-2					Netflix-3				
	Params			HR	ARHR	Params			HR	ARHR	Params			HR	ARHR
ItemKNN (cos)	100	-	-	0.1516	0.0689	100	-	-	0.0849	0.0316	100	-	-	0.0374	0.0123
ItemKNN (log)	100	-	-	0.0630	0.0240	100	-	-	0.0838	0.0303	100	-	-	0.0188	0.0062
ItemKNN (cprob)	20	0.5	-	0.1555	0.0678	500	0.5	-	0.0879	0.0326	200	0.1	-	0.0461	0.0162
PureSVD	600	-	-	0.1783	0.0865	400	-	-	0.0807	0.0297	400	-	-	0.0382	0.0131
BPRkNN	1e-3	1e-4	-	0.1678	0.0781	1e-4	1	-	0.0889	0.0329	0.01	1e-3	-	0.0439	0.0148
BPRMF	800	0.1	-	0.1638	0.0719	700	0.1	-	0.0862	0.0318	5	0.01	-	0.0454	0.0153
SLIM	1e-3	8	-	0.2025	0.1008	0.1	8	-	0.0947	0.0374	1e-4	12	-	0.0422	0.0149
FISMrmse	192	2e-5	0.001	0.2118	0.1107	192	6e-5	0.001	0.1041	0.0386	128	6e-5	0.001	0.0578	0.0185
FISMauc	192	1e-5	1e-4	0.2095	0.1016	240	2e-5	1e-4	0.0979	0.0341	160	4e-4	5e-4	0.0548	0.0177

Method	Yahoo-1					Yahoo-2					Yahoo-3				
	Params			HR	ARHR	Params			HR	ARHR	Params			HR	ARHR
ItemKNN (cos)	100	-	-	0.1344	0.0502	100	-	-	0.0890	0.0295	100	-	-	0.0366	0.0116
ItemKNN (log)	100	-	-	0.1046	0.0358	100	-	-	0.0820	0.0261	100	-	-	0.0489	0.0153
ItemKNN (cprob)	500	0.6	-	0.1387	0.0510	200	0.4	-	0.0908	0.0313	20	0.1	-	0.0571	0.0187
PureSVD	50	-	-	0.1229	0.0459	20	-	-	0.0769	0.0257	20	-	-	0.0494	0.0154
BPRkNN	1e-3	1e-4	-	0.1432	0.0528	1e-3	1e-4	-	0.0894	0.0304	0.1	0.01	-	0.0549	0.0183
BPRMF	700	0.1	-	0.1337	0.0473	700	0.1	-	0.0869	0.0288	10	0.01	-	0.0530	0.0169
SLIM	0.1	12	-	0.1454	0.0542	1e-3	12	-	0.0904	0.0304	0.1	2	-	0.0491	0.0159
FISMrmse	192	1e-4	0.001	0.1522	0.0542	192	2e-5	5e-4	0.0971	0.0371	160	0.002	0.001	0.0740	0.0230
FISMauc	144	8e-5	1e-4	0.1426	0.0488	160	2e-5	5e-4	0.0974	0.0315	176	2e-4	0.001	0.0722	0.0228

Comparison with Baselines



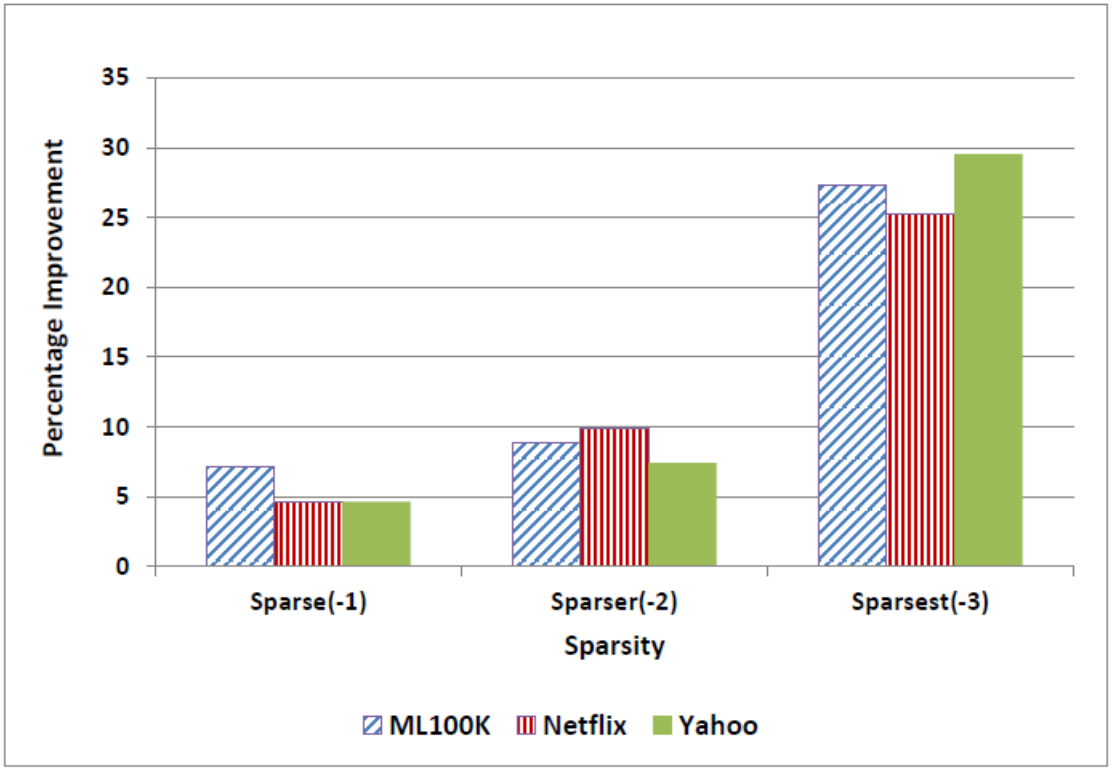
(a) ML100K dataset.



(b) Yahoo dataset.

- FISM outperforms all competitor methods across all datasets, even across different values of N
- Unexpectedly, FISMrmse outperforms FISMauc which is contrary to other reported results

Effect of Sparsity



As the data becomes more sparse, FISM increases its lead (in terms of HR) over competitor methods

Conclusion

- FISM learns item similarities as the product of two low rank matrices
- FISM generates high quality top-N recommendations even on sparse datasets
- Fast prediction can be achieved by intentionally introducing sparsity with a minimal reduction in quality