Recommender Systems

Online Update

Presented By-

Manish Mishra (271340)
Raghavendran Tata (271441)
Niraj Dev Pandey (271484)

Agenda

- Introduction
- Discussions on papers
- Comparison of the Papers
- Winning method

Introduction of the Topic

- **Online Update:** This topic deals with the updating the recommender system algorithms in inline/streaming way to increase the scalability , performance and accuracy of the systems
- **Motivation for Online Update:**
  - In today's big data environment, scalability of the algorithm is a challenge
  - User feedback is continuously being generated at unpredictable rates, which requires the algorithm to adapt and learn faster
  - Users' preferences are also not static, it change with time
- **Papers:**

1. Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems (Sarwar, Badrul, et al. Fifth International Conference on Computer and Information Science. 2002)
2. Vinagre, João, Alípio Mário Jorge, and João Gama. "Fast incremental matrix factorization for recommendation with positive-only feedback. "International Conference on User Modeling, Adaptation, and Personalization. Springer International Publishing, 2014
3. Matuszyk, Pawel, et al. "Forgetting methods for incremental matrix factorization in recommender systems." Proceedings of the 30th Annual ACM Symposium on Applied Computing. ACM, 2015.

- The presentation order follows the chronological order of the publication date of the papers

On line Update

Paper 1: Incremental singular value decomposition algorithms for highly scalable recommender systems

Presented by – Manish Mishra

Online Update

Structure

- Motivation and hypothesis
- State of art
  - Singular value decomposition (SVD)
  - Challenges of dimensionality reduction
- Incremental SVD algorithm
- Experimental evaluation
- Results
- Conclusion and future work

Introduction

Motivation:

- To investigate the use of dimensionality reduction for improving performance of Recommender Systems
- **Collaborative filtering (CF) -** based recommender systems are rapidly becoming a crucial tool
- Increasing amount of customer data poses two key challenges for CF based systems:
  - Quality of recommendations
  - Scalability
- **Singular Value Decomposition (SVD)** based recommendations algorithms can produce fast, high quality recommendations but has to undergo very expensive matrix factorization steps

Hypothesis:

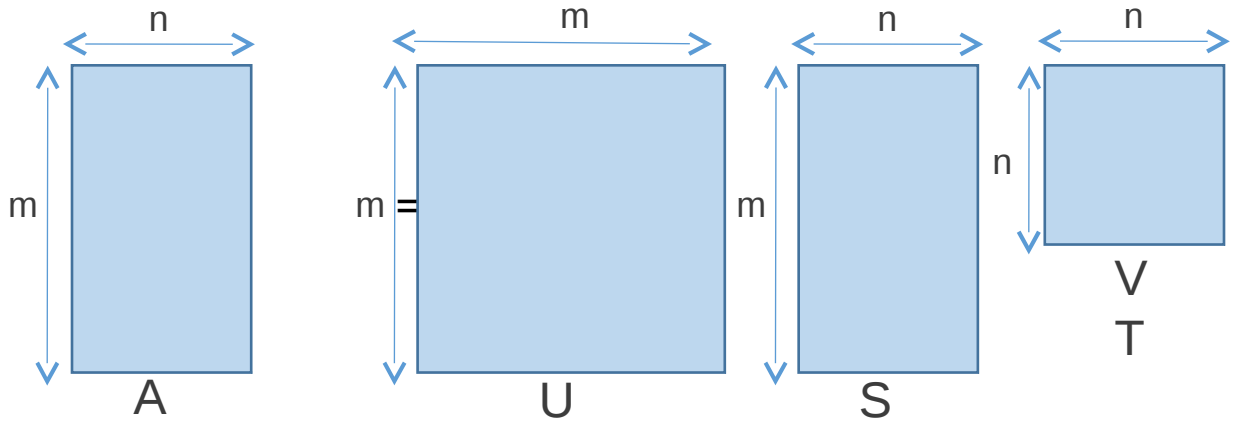
- The paper suggests an incremental model building technique for SVD-based CF that has potential to be highly scalable while producing good predictive accuracy
- Experimental results show that the overall algorithm works twice as fast while producing similar prediction accuracy

Online Update

State of the art: Singular Value Decomposition (SVD)

Matrix factorization technique for producing low-rank approximations

$$SVD(A_{m\times n}) = U_{m\times m} \times S_{m\times n} \times V^{T}_{n\times n}$$



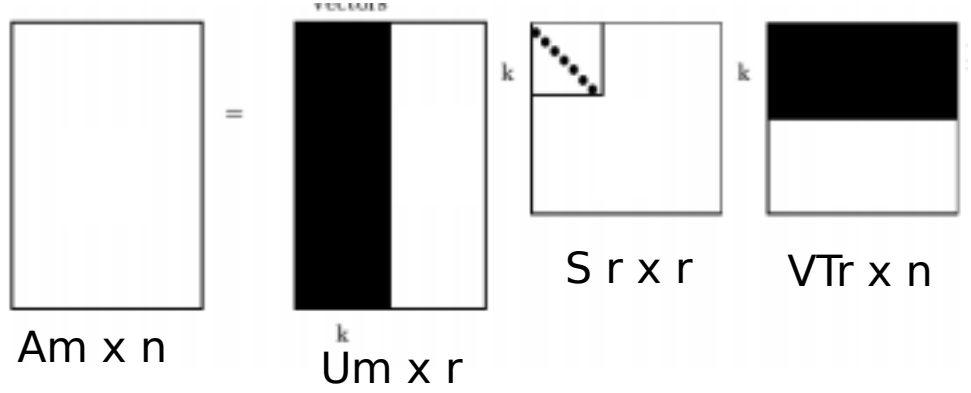
U and V are orthogonal matrices

S is a diagonal matrix with only r nonzero entries such that *si* > 0 and *s*1 ≥ *s*2 ≥ . . . ≥ *sr* where r is the rank of matrix A

The r columns of U corresponding to the nonzero singular values span the column space (eigenvectors of AAT) , and the r columns of V span the row space of the matrix A (eigenvectors of ATA)

Online Update

State of the art: Singular Value Decomposition (SVD) contd..



*Ref: [1]

- It is possible to retain only k << r singular values by discarding other entries (Sk diagonal matrix).
- The reconstructed matrix *Ak = Uk.Sk.VkT* is a *rank-k* matrix that is the closest approximation to the original matrix A
- Better than the original space itself due to the filtering out of the small singular values that introduce “noise” in the customer-product relationship.
- This produces a set of uncorrelated eigenvectors. Each customer and product is represented by its corresponding eigenvector.

Online Update

Manish Mishra

State of the art: Singular Value Decomposition (SVD) contd..

Prediction Generation Using SVD

$$P_{i,j} = \bar{r}_i + (U_k . \sqrt{S_k^T}(i)).(\sqrt{S_k} . V_k^T(j))$$

Where,

$P_{i,j}$ is the prediction for *ith* customer and *jth* product

$\bar{r}_i$ is the row average

Once the SVD decomposition is done, the prediction generation process involves only a dot product computation, which takes $O(1)$ time, since k is a constant

State of the art

Challenges of Dimensionality Reduction

- The entire recommender system algorithm works in two separate steps:
  - Offline or model-building step
    - User-user similarity computation and neighborhood formation i.e. SVD decomposition
    - Time consuming and infrequent
    - Run-time of O(m3) for matrix Am x n
  - On-line or the execution step
    - Actual prediction generation
    - O(1)

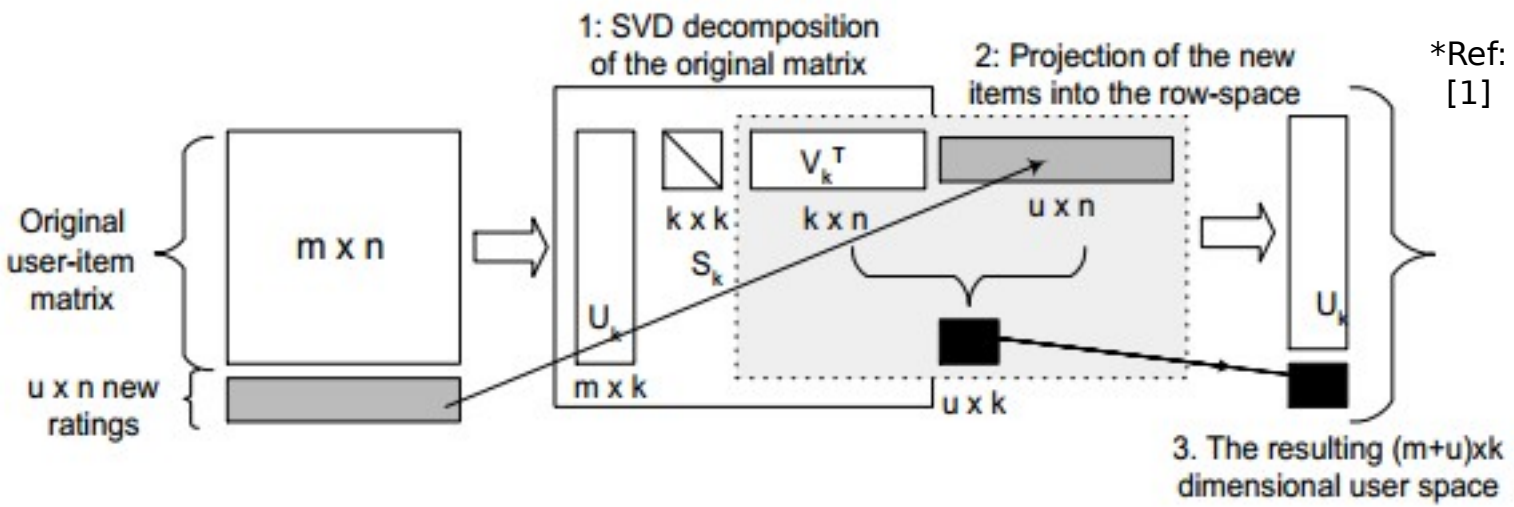
Online Update

Incremental SVD Algorithm

- The idea is borrowed from the Latent Semantic Indexing (LSI) world to handle dynamic databases
- LSI is a conceptual indexing technique which uses the SVD to estimate the underlying latent semantic structure of the word to document association.
- Projection of additional users provides good approximation to the complete model
- Authors build a suitably sized model first and then use projections to incrementally build on that
- Errors induced as the space is not orthogonal

Online Update

Incremental SVD Algorithm contd..



*Ref: [1]

Algorithm: ***Folding –in*** *(As per the paper)*

•Project the new user vector Nu (t x1) as

P= *Uk* × *Uk T* × *N*u

•Append k-dimensional vector UkT.Nu as new column of the k x n matrix Sk.VkT

Online Update

Algorithm: ***Folding –in*** *(Reference [2])*

- To fold in a new user vector u (1x n), a projection onto the current product vectors (Vk) is computed as $\hat{u}_{1\times k} = u V_k S_k^{-1}$
- Similarly to fold in a new product vector d (mx1), a projection onto the current user vectors (Uk) is computed as $\hat{d}_{1\times k} = d^T U_k S_k^{-1}$

Incremental SVD Algorithm contd..

Pseudo Code: Folding-in

Input: $A_{m\times n} = U_{m\times k}.S_{k\times k}.V^{T}_{k\times n}$ $d_{u\times n}$

Output: $A'_{(m+u)\times n} = U_{(m+u)\times k}.S_{k\times k}.V^{T}_{k\times n}$

1 **For** *i=1 to u* **do**

2

3 $\hat{d}_{i^{th}\times k} = d_{i^{th}\times n}.V_{n\times k}.S^{-1}_{k\times k}$ *// Append in U*

4 *end*

5 *return* $U_{(m+i)\times k}$ $\hat{d}_{i^{th}\times k}$

$$U_{(m+u)\times k}.S_{k\times k}.V^{T}_{k\times n}$$

Online Update

Experiment Details

Data Parameters	Description
Data source	MovieLens (www.movielens.umn.edu)
Ratings	100,000 ratings (Users with 20 or more ratings considered)
User-Movie matrix	943 users (rows) and 1682 movies (columns)
Test-Training ratio	X : 80%, 50% and 20%

- **Evaluation Metric**
  - Mean Absolute Error (MAE) = $\frac{\sum_{i=1}^{N} | p_i - q_i |}{N}$

  where,

  <*pi* – *qi*> is a ratings – prediction pair

Online Update

Manish Mishra

Experiment Procedure

- Two hyper parameters need to be optimized before the experiment

1. **The number of dimensions (k):** Optimized by performing prediction experiments over different dimensions each time. The results were plotted and k=14 was obtained as an optimal value

2. **The threshold model size (basis size):** Optimized by performing experiments with a fix basis size and computing the SVD model by projecting the rest of (total -basis) users using the folding-in technique. MAE was plotted for the experiments and the optimal basis size was chosen

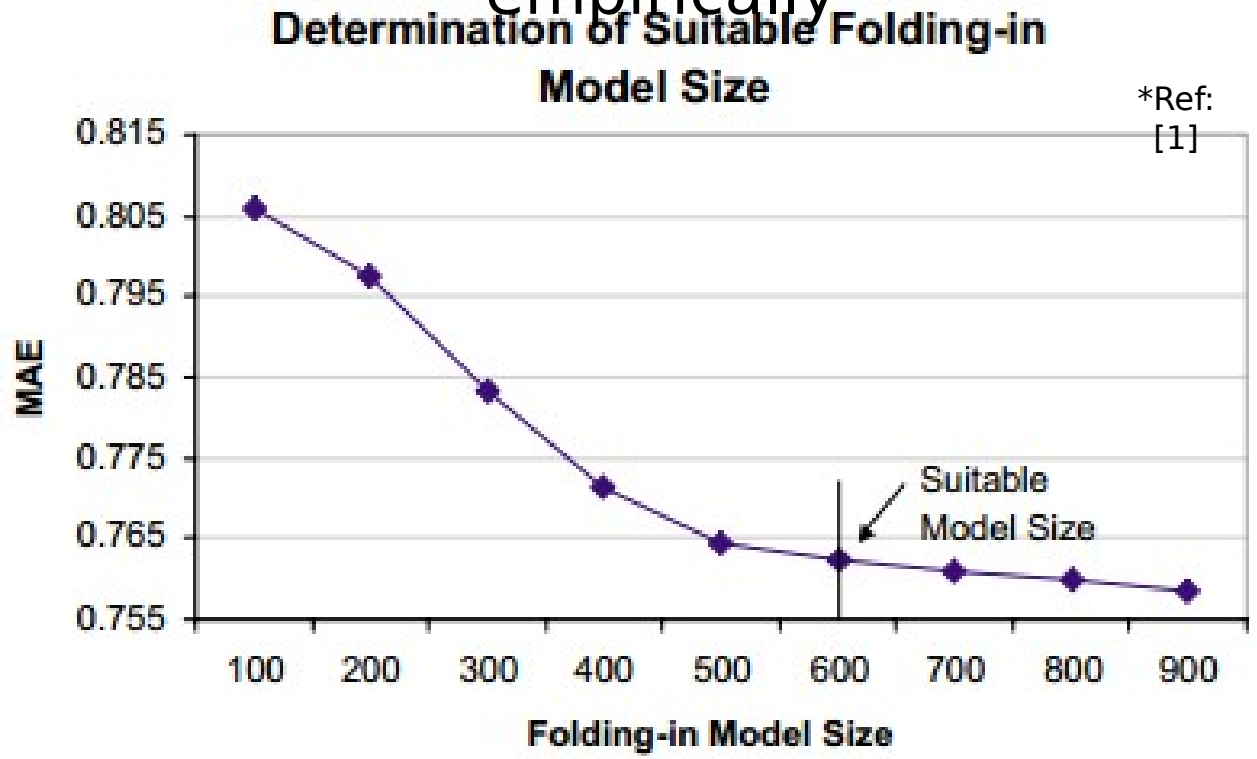
- These hyper parameters are used to build an initial SVD model (A=USVT) and then use the folding-in technique to incrementally compute the SVD model for additional users

- 10-fold cross validation by selecting random training and test data for all our experiments

Online Update

Model Size

Optimal reduced Rank k =14 was found empirically



*Ref: [1]

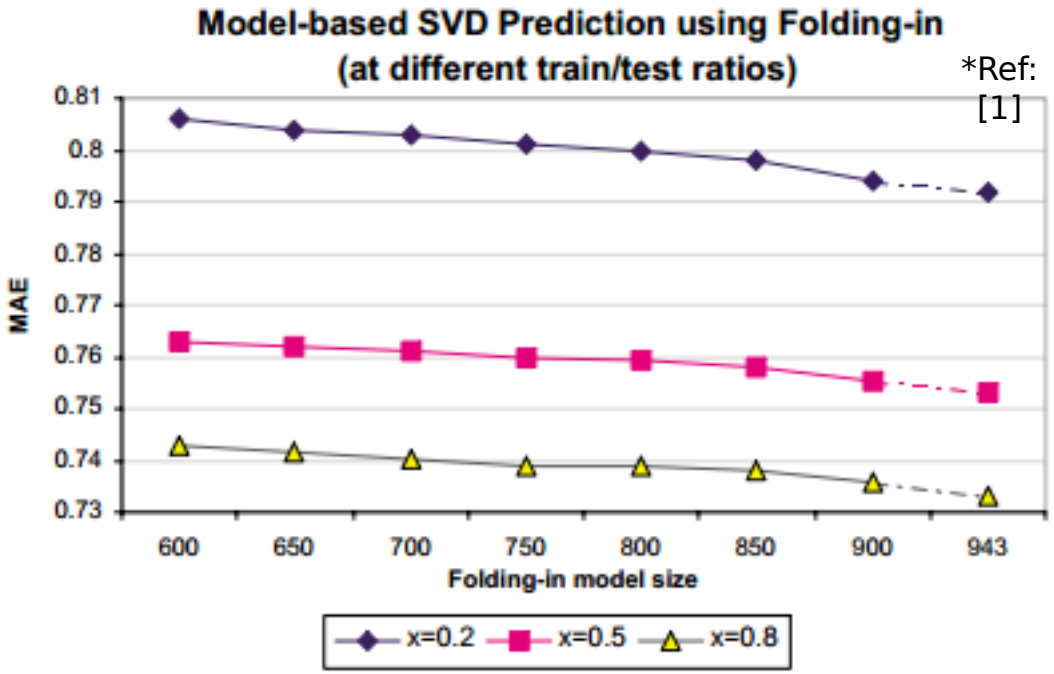
Select a basis size that is small enough to produce fast model building yet large enough to produce good prediction quality

(943 – Model size) is projected using folding-in

Online Update

Results

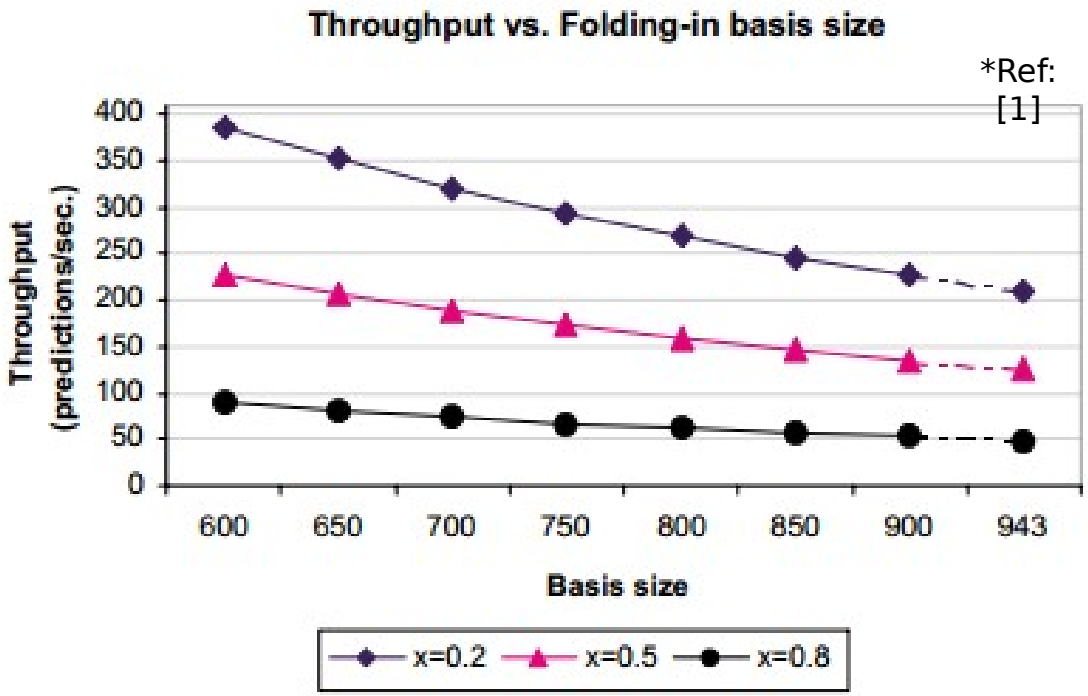
Quality



*Ref: [1]

MAE $x = 0.8 = 0.733$ (full model size) and 0.742 (model size of 600) (only 1.22% quality drop)
=> Even with a small basis size it is possible to obtain a good quality.

Performance



*Ref: [1]

Corresponding to x=0.8, at basis size 600 throughput rate s is 88.82 whereas at basis size 943(full model) throughput rate becomes 48.9. So there is 81.63% performance gain

Online Update

Manish Mishra

Conclusion

- The SVD-based recommendation generation technique leads to very fast online performance but computing the SVD is very expensive
- The Incremental SVD algorithms, based on folding-in, can help recommender systems achieve high scalability while providing good predictive accuracy
- The folding-in technique requires less time and storage space

Paper Evaluation

- SVD based recommender systems has following limitations
  - Can not be applied on sparse data
  - doesn't have regularization
- Future work led to better matrix factorization techniques to handle these limitations
- Importance of the papers lies in starting the discussion on "Online Update" for recommender systems

References

1. *Sarwar, Badrul, et al. "Incremental singular value decomposition algorithms for highly scalable recommender systems." Fifth International Conference on Computer and Information Science. 2002.*
2. *Berry, M. W., Dumais, S. T., and O'Brian, G. W. (1995). Using Linear Algebra for Intelligent Information Retrieval. SIAM Review, 37(4).*
3. *Deerwester, S., Dumais, S. T., Furnas, G. W.,Landauer, T. K., and Harshman, R. (1990). Indexing by Latent Semantic Analysis. Journal of the American Society for Information Science. 41(6).*
4. *G. W. O`brien, Information Management Tools for Updating an SVD-Encoded Indexing Scheme, Master's thesis, The University of Knoxville, Knoxville, TN, 1995*
5. *Wikipedia*

Online Update

Manish Mishra

Example:

Latent Semantic Indexing and Updating (Folding-in) Example

*Ref: [4]

Original Data

Document Id	Titles
c1	Human Machine Interface for Lab ABC Computer Applications
c2	A Survey of User Opinion of Computer Systems Response Time
c3	The EPS User Interface Management Systems
c4	Systems and Human Systems Engineering Testing of EPS-2
c5	Relation of User-Perceived Response Time to Error Measurement
m1	The Generation of Random, Binary, Unordered Tree
m2	Intersection Graph of Paths in a Tree
m3	Graph Minors IV: Tree-Width and Well-Quasi-Ordering
m4	Graph Minors - A Survey

- These titles are based on two topics : the “c" documents refer to human-computer interaction and the “m" documents refer to group theory

Term-Document Matrix of Original Data

*Ref: [4]

Terms	Documents								
	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	0	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

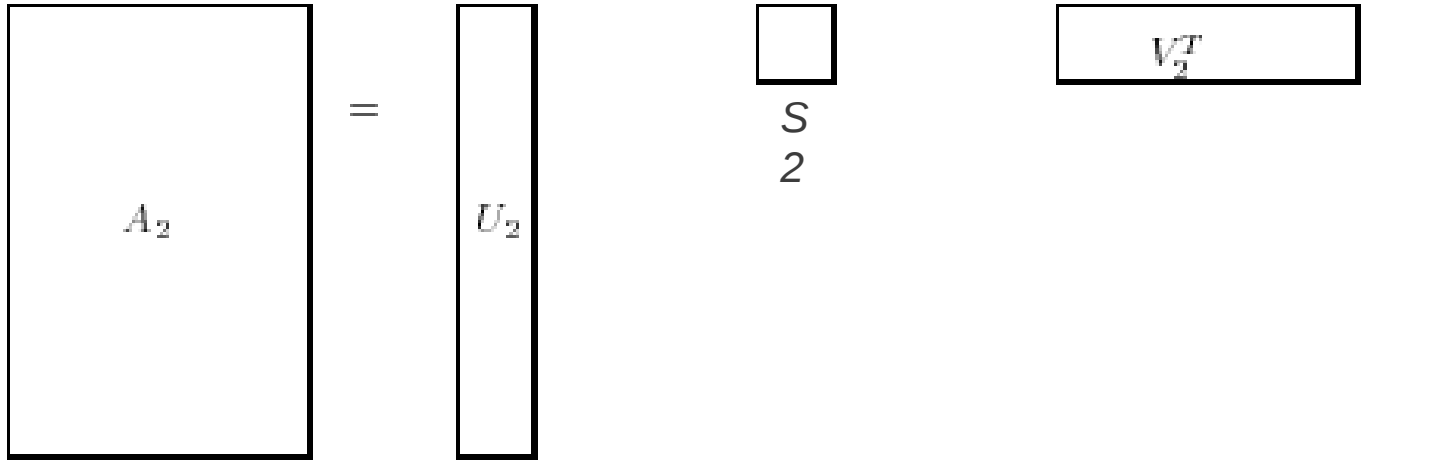
- The elements of this 129 term-document matrix are the frequencies in which a term occurs in a document or title

Online Update

Example:

SVD: Selecting k=2, the best rank-2

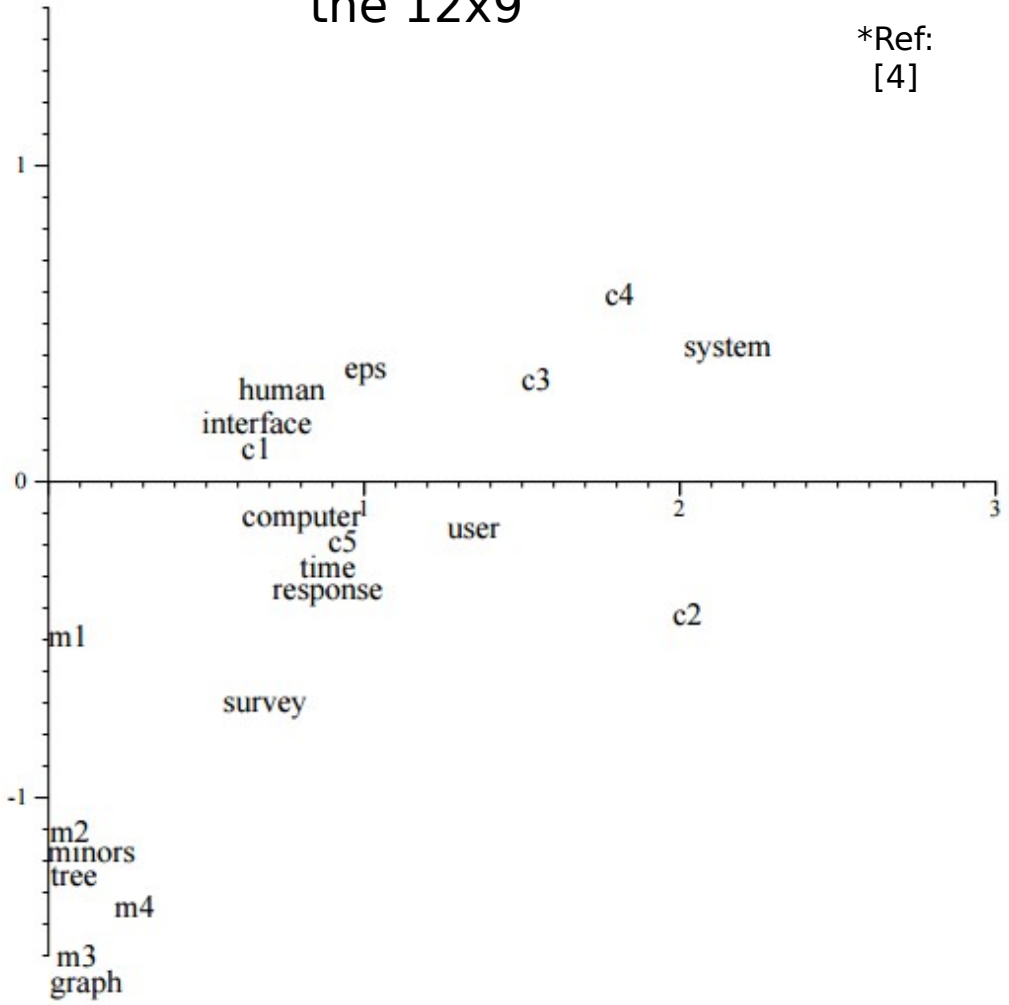
*Ref: [4]



Online Update

Example:

Two-dimensional plot of terms and documents for the 12x9



*Ref: [4]

Terms Representation:
X-axis:1st column of U2 scaled by s1
Y-axis:2nd column of U2 scaled by s2

Documents Representation:
X-axis: 1st column of V2 scaled by s1
Y-axis: 2nd column of V2 scaled by s2

Notice that documents and terms pertaining to human computer interaction are clustered around the x-axis and the graph theory-related terms and documents are clustered around the y-axis

Online Update

Example:

Folding-in

Suppose another document *d= "human computer"* needs to be added
Then,
Document vector *d12x1 = [1 0 1 0 0 0 0 0 0 0 0 0] T*
Projection of d12x1 will be

*Ref:
[4]

$$\begin{pmatrix} 0.1383 & 0.0275 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}^T \begin{pmatrix} 0.2405 & -0.0432 \\ 0.3008 & 0.1413 \\ 0.0361 & -0.6228 \\ 0.2214 & 0.1132 \\ 0.1976 & 0.0721 \\ 0.0318 & -0.4505 \\ 0.2650 & -0.1072 \\ 0.2059 & -0.2736 \\ 0.6445 & -0.1673 \\ 0.0127 & -0.4902 \\ 0.2650 & -0.1072 \\ 0.4036 & -0.0571 \end{pmatrix} \begin{pmatrix} 3.3409 & 0 \\ 0 & 2.5417 \end{pmatrix}^{-1}$$

This can be appended as column in *VT* to give *UmxkSkxkVTkx(n+1)*

Online Update

Agenda

- Introduction
  - Motivation
  - Hypothesis
- Batch Stochastic Gradient Descent (SGD)
- Evaluation Issues
- Proposed Algorithm -- Incremental Matrix Factorization for item prediction
- Example with Datasets
- Conclusion and Future Work
- References

Introduction

Motivation:

- The optimization process of batch SGD requires several iterations through the entire data set
- This procedure holds good for stationary data however, its not acceptable for streaming data
  - As number of observations increases, repeatedly visiting all the available data becomes too expensive

Hypothesis:

- The paper introduce a simple but fast Incremental Matrix Factorization algorithm for positive – only feedback
- Experimental results show that the overall algorithm has competitive accuracy, while being significantly faster

Introduction

- The purpose of recommender systems is to aid users in the usually overwhelming choice of items from a large item collection
- Collaborative Filtering (CF) is a popular technique to infer unknown user preferences from a set of known user preferences
- Collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating)

Batch Stochastic Gradient Descent (SGD)

Algorithm 1 BSGD: Batch SGD

Data: $D = < u, i, r >$
input : $feat$, $iters$, λ, η
output: A, B

init:
  for $u \in$ `Users`(D) **do**
    $A_u \leftarrow$ `Vector`(size : $feat$)
    $A_u \sim \mathcal{N}(0, 0.1)$
  for $i \in$ `Items`(D) **do**
    $B_i \leftarrow$ `Vector`(size : $feat$)
    $B_i \sim \mathcal{N}(0, 0.1)$

for $count \leftarrow 1$ **to** $iters$ **do**
  $D \leftarrow$ `Shuffle`(D)
  for $< u, i, r > \in D$ **do**
    $err_{ui} \leftarrow r - A_u.B_i^T$
    $A_u \leftarrow A_u + \eta(err_{ui}B_i - \lambda A_u)$
    $B_i \leftarrow B_i + \eta(err_{ui}A_u - \lambda B_i)$

- Edit Master text styles
  - Second level
    - Third level
      - Fourth level
        - Fifth level

* Ref [1]

- The advantage of BSGD is that complexity grows linearly with the number of known ratings in the training set, actually taking advantage of the high sparsity of R

Evaluation Issues

Classic evaluation methodologies for recommender systems begin by splitting the ratings dataset in two subsets – **training set** and **testing set** – randomly choosing data elements from the initial dataset. However, there are some issues:

- **Dataset ordering**
- **Time awareness**
- **Online updates**
- **Session grouping**
- **Recommendation bias**

Proposed Algorithm -- Incremental Matrix Factorization for item prediction

- The Algorithm 1 proposed consists of Batch procedure requiring several passes through the dataset to train a model
  - Easy – stationary environment
  - Much difficult and more expensive – moving / streaming data
- Algorithm 2 proposed is called Incremental SGD and has 2 differences when compared to Algorithm 1
  - At each observation < u, i >, the adjustments to factor matrices A and B are made in a single step
  - No data shuffling – or any other pre-processing is performed
- Since, we deal with positive only feedback we assume numerical values for True Values as **“1”** and error is measured as

$$err_{ui} = 1 - \hat{R}_{ui}$$

  - The Matrix R contains either true values – for positively rated items – or false values – for unrated items (we assume false values as missing values)

Algorithm

Algorithm 2 ISGD - Incremental SGD

Data: $D = \{< u, i >\}$, a finite set or a data stream
input : $feat$, λ, η
output: A, B

for $< u, i > \in D$ **do**
    if $u \notin$ `Rows`(A) **then**
        $A_u \leftarrow$ `Vector`(size : $feat$)
        $A_u \sim \mathcal{N}(0, 0.1)$
    if $i \notin$ `Rows`(B^T) **then**
        $B_i^T \leftarrow$ `Vector`(size : $feat$)
        $B_i^T \sim \mathcal{N}(0, 0.1)$
    $err_{ui} \leftarrow 1 - A_u.B_i^T$
    $A_u \leftarrow A_u + \eta(err_{ui} B_i - \lambda A_u)$
    $B_i \leftarrow B_i + \eta(err_{ui} A_u - \lambda B_i)$

* Ref [1]

Example with Datasets

- To support the proposed solution, the Author's have considered 4 different datasets with 4 Algorithm's and compared the values **"Update Time"** for set N ∈ {1, 5, 10}
- The Algorithms considered are:
  - Incremental Stochastic Gradient Descent (ISGD)
  - Bayesian Personalized Ranking MF (BPRMF)
  - Weighted Bayesian Personalized Ranking MF (IBPRMF)
  - User Based Nearest Neighbour's algorithm (UKNN)

Dataset	Events	Users	Items	Time frame	Sparsity
Music-listen	335.731	4.768	15.323	12 months	99,90%
Lastfm-600k	493.063	164	65.013	8 months	99,11%
Music-playlist	111.942	10.392	26.117	45 months	99,96%
MovieLens-1M	226.310	6.014	3.232	34 months	98,84%

Table 1. Dataset description

* Ref [1]

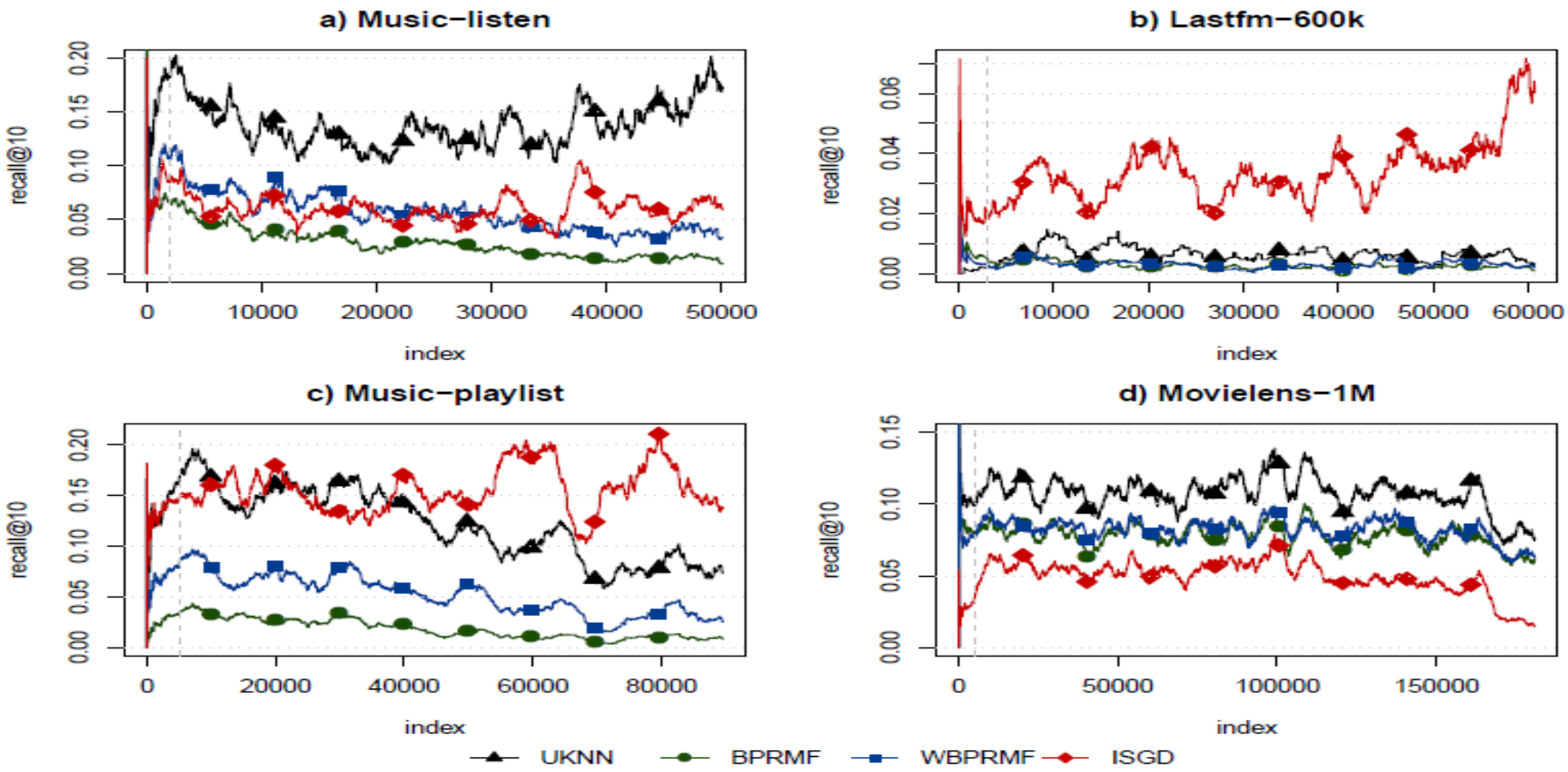
Example with Datasets

Dataset	Algorithm	Recall@1	Recall@5	Recall@10	Update time
Music-listen	BPRMF	0,003	0,016	0,028	0,846 ms
	WBPRMF	0,012	0,037	0,056	1,187 ms
	ISGD	0,017	0,044	0,061	**0,118 ms**
	UserKNN	**0,038**	**0,101**	**0,139**	328,917 ms
Lastfm-600k	BPRMF	<0,001	0,001	0,003	28,061 ms
	WBPRMF	<0,001	0,002	0,003	29,194 ms
	ISGD	**0,012**	**0,027**	**0,034**	**1,106** ms
	UserKNN	0,001	0,004	0,006	290,133 ms
Music-playlist	BPRMF	<0,001	0,009	0,020	1,889 ms
	WBPRMF	0,011	0,038	0,057	2,156 ms
	ISGD	**0,060**	**0,136**	**0,171**	**0,949 ms**
	UserKNN	0,033	0,095	0,132	190,250 ms
Movielens-1M	BPRMF	0,012	0,045	0,080	0,173 ms
	WBPRMF	0,013	0,050	0,084	0,229 ms
	ISGD	0,007	0,028	0,050	**0,016 ms**
	UserKNN	**0,018**	**0,066**	**0,110**	84,927 ms

Table 2. Overall results. Best performing algorithms are highlighted in bold for each dataset. Update times are the average value of the update time for all data points.

* Ref [1]

Example with Datasets



* Ref [1]

Conclusion and Future Work

- Conclusion
  - Proposed fast matrix factorization algorithm dealing with positive only user feedback
  - Proposed prequential evaluation framework for streaming data
  - By testing data sets with other incremental algorithms, ISGD is faster with competitive accuracy

- Future Work
- Better understanding of the effects of dataset properties such as
  - Sparseness
  - User-Item Ratios
  - Frequency Distributions

References

1. *Fast incremental matrix factorization for recommendation with positive-only feedback -- Joao Vinagre, Alipio Mario Jorge, and Joao Gama*
2. *Goldberg, D., Nichols, D.A., Oki, B.M., Terry, D.B.: Using collaborative filtering to weave an information tapestry. Commun. ACM 35(12) (1992) 61 – 70*
3. *Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. [25] 263 – 272*
4. *Pan, R., Zhou, Y., Cao, B., Liu, N.N., Lukose, R.M., Scholz, M., Yang, Q.: Oneclass collaborative filtering. [25] 502 – 511*
5. *Wikipedia*

Paper 03 – Selective Forgetting for Incremental Matrix Factorization

Presented by – Niraj Dev Pandey

Contents

Introduction

Motivation:

- The Recommender systems should reflect current state of preferences at any time point, but preferences are not static
- Preferences are subjected to concept drift / even drift i.e. it undergoes permanent changes as the taste of Users and perception of Items change over time
- It is important to select the actual data for training models and forget outdated ones

Hypothesis:

- The paper proposes two forgetting techniques for Incremental Matrix Factorization and incorporate them into a Stream recommender
- A new evaluation protocol for Recommender Systems in a Streaming Environment is introduced and it shows that the forgetting of outdated date increases the quality of recommendation substantially

Why to forget ?

- Users’ preferences are not static
- Extreme data sparsity
- Old data doesn’t reflect the current users’ preferences
- Training models upon old data decrease the quality of our prediction

Drift

- Time-changing data stream
- In order to guaranteed that results are always up-to-date, it is necessary to analyze the incoming data in an online manner
- Incorporate new and eliminate old

Drift software

- EDDM (Early drift detection method) , MOA , Rapid Miner

(https://en.wikipedia.org/wiki/Concept_drift)

Methods

Phase – 1 Initial Training

- Create latent user and items features using BRISMF algorithm
- It is a pre-phase for actual steam-based training
- Rating matrix ‘R’ should be decomposed into a product of two matrices $R = PQ$
- To Calculate the decomposition SGD is used

Phase – 2 Stream based Learning

- Result of Initial training would be input for this section
- This is prime mode
- Drift or Shift
- selective forgetting techniques are applied in this mode

Algorithm 1 Incremental Learning with Forgetting

Input: $r_{u,i}, R, P, Q, \eta, k, \lambda$

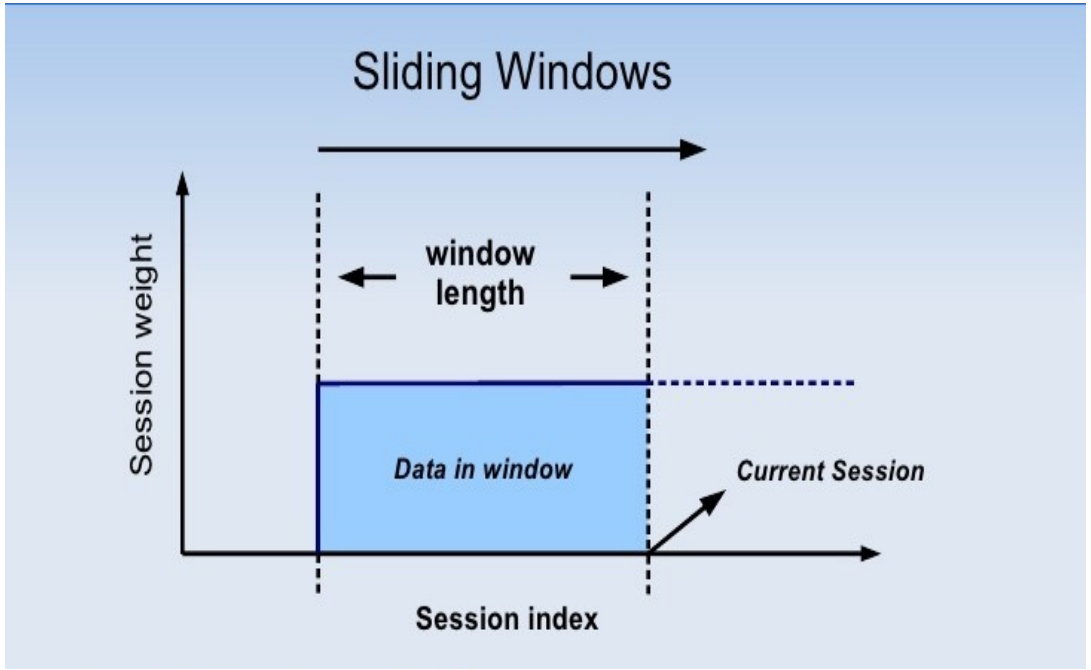
1: $\overrightarrow{p_u} \leftarrow \text{getLatentUserVector}(P, u)$
2: $\overrightarrow{q_i} \leftarrow \text{getLatentItemVector}(Q, i)$
3: $\widehat{r}_{u,i} = \overrightarrow{p_u} \cdot \overrightarrow{q_i}$ //predict a rating for $r_{u,i}$
4: $\text{evaluatePrequentially}(\widehat{r}_{u,i}, r_{u,i})$ //update evaluation measures
5: $\overrightarrow{r}_{u*} \leftarrow \text{getUserRatings}(R, u)$
6: $(\overrightarrow{r}_{u*}).\text{addRating}(r_{u,i})$
7: **applyForgetting**$(\overrightarrow{r}_{u*})$ **//old ratings removed**
8: $epoch = 0$
9: **while** $epoch < optimalNumberOfEpochs$ **do**
10: epoch++; //for all retained ratings
11: **for all** $r_{u,i}$ in $\overrightarrow{r}_{u*}$ **do**
12: $\overrightarrow{p_u} \leftarrow \text{getLatentUserVector}(P, u)$
13: $\overrightarrow{q_i} \leftarrow \text{getLatentItemVector}(Q, i)$
14: $predictionError = r_{u,i} - \overrightarrow{p_u} \cdot \overrightarrow{q_i}$
15: **for all** latent dimensions $k \neq 1$ in $\overrightarrow{p_u}$ **do**
16: $p_{u,k} \leftarrow p_{u,k} + \eta \cdot (predictionError \cdot q_{i,k} - \lambda \cdot p_{u,k})$
17: **end for**
18: **end for**
19: **end while**

Forgetting Techniques

Instance-based Forgetting

- If window grows above the predefined size, the oldest rating is removed as many

  times as needed to reduce it back to the size ‘w’



Instance Based Forgetting Algorithm

- New ratings are added into the list of user's ratings ‘ru’
- Window is represented by ‘w’

Algorithm 2 applyForgetting($r_{u,*}$) - Instance-based Forgetting

Input: $r_{u,*}$ a list of ratings by user u sorted w.r.t. time, w - window size

1: **while** $|r_{u,*}| > w$ **do**
2: removeFirstElement($r_{u,*}$)
3: **end while**

Time Based Forgetting Algorithm

- Define preferences with respect to time
- In volatile applications time span might be reasonable

Algorithm 3 applyForgetting($r_{u,*}$) - Time-based Forgetting

Input: $r_{u,*}$ a list of ratings by user u sorted w.r.t. time, a - age threshold

```
1: forgettingApplied ← true
2: while forgettingApplied == true do
3:     oldestElement ← getFirstElement(r_{u,*}) //the oldest rating
4:     if age(oldestElement) > a then
5:         removeFirstElement(r_{u,*})
6:         forgettingApplied ← true
7:     else
8:         forgettingApplied ← false
9:     end if
10: end while
```

Evaluation Measure – sliding RMSE

- Popular evaluation measure
- Based on deviation of predicted & real rating

RMSE

(where T is a test set)

- Calculating ‘sliding RMSE’ is the same as for RMSE
- Test set T is different

Experiments

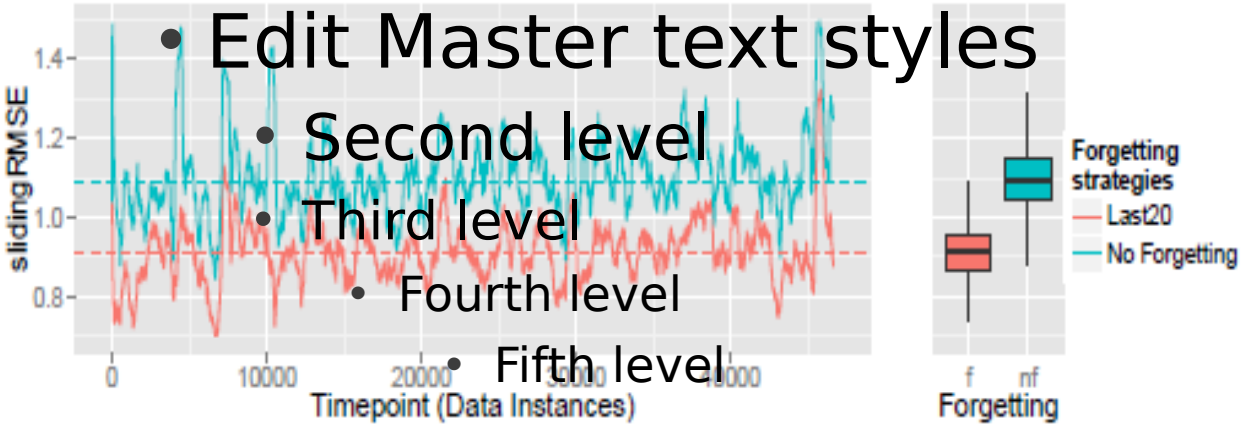
- Author's have dealt with 4 real datasets Movielens 1M, Movielens 100K, Netflix (a random sample of 1000 Users) Epinions (extended)
- Used modified version of ***BRISMF*** algorithm with and without forgetting
- Performed grid search to find the approximately optimal parameter setting

Dataset	ML1M	ML100k	Epinions	Netflix
avg. slidingRMSE - Forgetting	**0.9151**	**1.0077**	**0.6627**	**0.9138**
avg. slidingRMSE - NO Forgetting	1.1059	1.0364	0.8991	1.0162

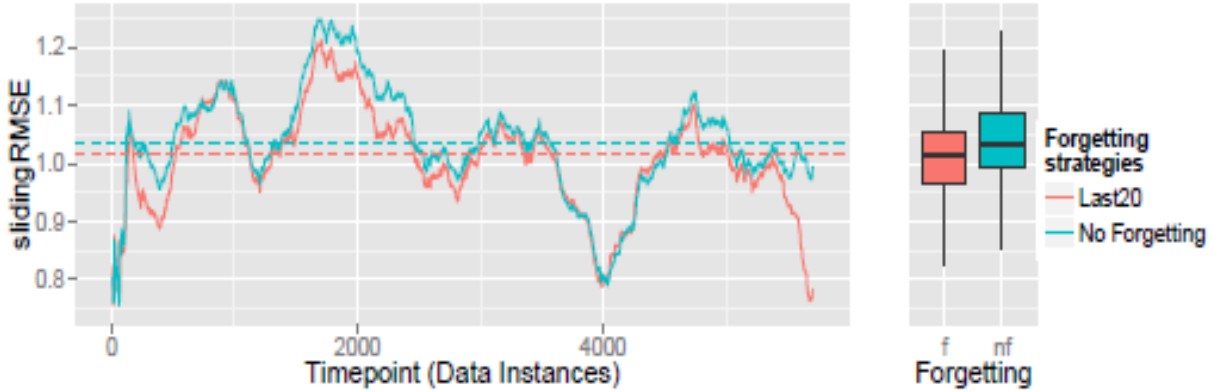
Table 1: Average values of sliding RMSE for each dataset (lower values are better). Our forgetting strategy outperforms the non forgetting strategy on all datasets

Experiments (1/2)

- Edit Master text styles
  - Second level
    - Third level
      - Fourth level
        - Fifth level

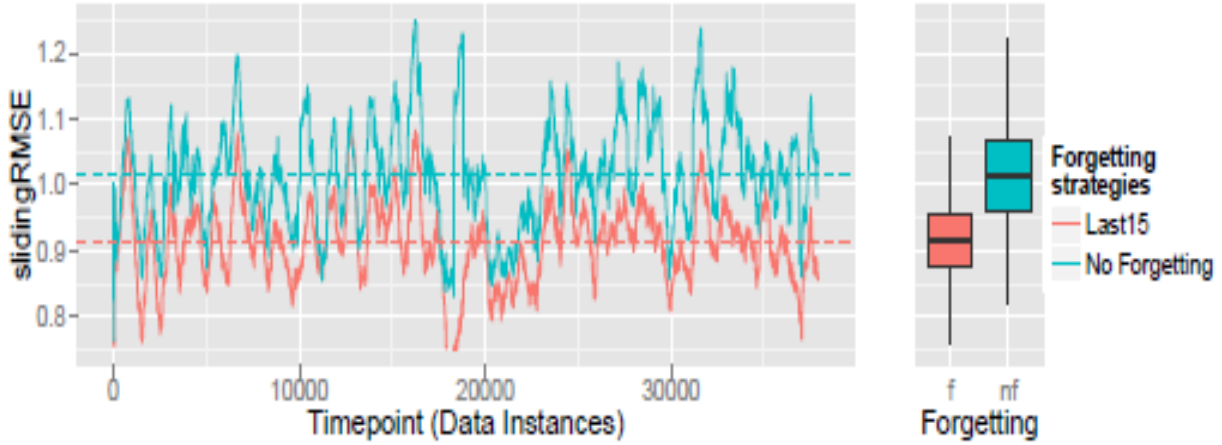


(a) Movielens 1M

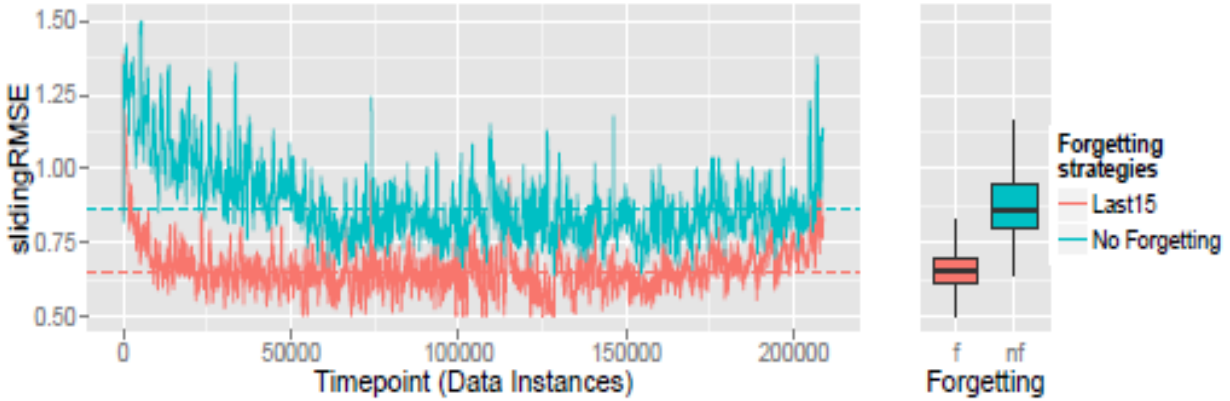


(b) Movielens 100k

Experiments (2/2)



(c) Netflix (random sample of 1000 users)



(d) Epinions extended

Conclusion

- We Investigated selective forgetting techniques for matrix factorization in order to improve the quality of recommendations
- We proposed two techniques, an instance-based and time-based forgetting
- Designed a new evaluation protocol for stream-based recommenders which takes the initial training and temporal aspects into account
- Incorporated them into a modified version of the BRISMF algorithm
- Our approach is based on a user-specific sliding window
- Introduced more appropriate evaluation measures *sliding RMSE*
- Beneficial to forget the outdated user's preferences despite of extreme data sparsity

References

1. *Matuszyk, Pawel, et al. "Forgetting methods for incremental matrix factorization in recommender systems." Proceedings of the 30th Annual ACM Symposium on Applied Computing. ACM, 2015.*
2. *https://www.wikipedia.org/*
3. *http://www.slideshare.net/jnvms/incremental-itembased-collaborative-filtering-4095306*
4. *file:///C:/Users/Dell/Downloads/tema_0931.pdf*
5. *C. Desrosiers and G. Karypis. A Comprehensive Survey of Neighborhood-based Recommendation Methods. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, Recommender Systems Handbook, pages 107{144. Springer US*
6. *J. Gama, R. Sebasti~ao, and P. P. Rodrigues. Issues in evaluation of stream learning algorithms. In KDD, 2009*
7. *Y. Koren. Collaborative filtering with temporal dynamics. In KDD, 2009*

Comparisons and Differences

Incremental SVD

- Incremental model building for SVD – Based CF systems
- Focus on Scalability of Recommender Systems
- ***Folding - In*** technique that requires less time and storage space

Incremental SGD

- Incremental matrix factorization (ISGD)
- Focus on positive only user feedback and prequential evaluation framework for streaming data

Selective Forgetting for Incremental Matrix Factorization

- Incremental Matrix Factorization using Forgetting techniques
- Focus on accuracy and using recent relevant data
- Modified version of BRISMF algorithm
- Introduced sliding window mechanism with limited space
- Forget extreme data sparsity

On line Update

Winning Method

- Although 3 papers deals with slightly different scenarios of the Online Update but the “Selective Forgetting for Incremental Matrix Factorization” seems to be more generic and hence should be winning method