# Recommendations for Streaming Data

Karthik Subbian [†] Charu Aggarwal [*] Kshiteesh Hegde [†]

[†]University of Minnesota - Minneapolis, MN

[*]IBM Watson Research Center - Yorktown Heights, NY

CIKM 2016

## Introduction

- Most current recommender systems are designed in the context of offline setting.
- It is desirable to provide real-time recommendations in large-scale scenarios.
- Some applications: social networks, movie/book suggestions, dating.

## Challenges

- Changing user preferences:
  - New items keep appearing.
  - The underlying user patterns keep changing.
  - This causes the recommendations to vary with time.
- In-core memory for memory-resident operations is quite limited.
- Classical methods like neighborhood-based and latent factor models have shortcomings.
  - They require a computationally expensive offline phase.
  - Factorizing large matrices is cumbersome when the said matrices are rapidly changing with time.

## The Setup

- The ratings are received in the format: `<userID, itemID, rating>`.
- If rating is drawn from $\{-1, +1\}$, then let users who have given a rating of $+1$ to item $i$ at time $t$ be represented by $P(i, t)$ and $-1$ by $N(i, t)$.
- Since exact similarity computation is intensive, we compute it probabilistically by imposing a sort order on the users with the help of hash functions.
  - Use $d$ mutually independent hash functions.
  - Each hash function takes in an identifier of a user and outputs a random number uniformly distributed in $(0, 1)$.

## Probabilistic Similarity

- Now, for a given sort order, what is the probability that the first user with a positive rating for item $i$ is the same as the first user with a positive rating for item $j$?
- This is the probability that both $i$ and $j$ take on the value $+1$ when at least one of them takes on the value of $+1$ given by:

$$\frac{P(i,t) \cap P(j,t)}{P(i,t) \cup P(j,t)}$$

- The above expression represents the similarity between items $i$ and $j$ with respect to positive ratings.

## Probabilistic Similarity

- The $d$ mutually independent hash functions are applied to the user indices that have rated item $j$ positively.
- For each hash function, the least hash value (*min-hash value*) among these positive users and the corresponding user index (*min-hash index*) are maintained.
- $d$ of such pairs are maintained for the $n$ items seen which are easily updatable. This drastically reduces memory requirements.
- Similarly, the process is repeated for negatively rated items. The system can be extended for scenarios with multiple ratings as well.

## Probabilistic Similarity

- How good is the quality of the similarity measure computed in this manner?
- Let $R^+(i, j, t)$ be an approximation to the Jaccard coefficient computed by the min-hash approach and $S^+(i, j, t)$ be the actual value. Then, we prove the following:
- Lower Tail Bound: For any $\epsilon \in (0, 1)$, $R^+(i, j, t)$ lies outside $S^+(i, j, t)$ by a factor of $(1 - \epsilon)$ with the probability:

$$P(R^+(i,j,t) < (1-\epsilon) \cdot S^+(i,j,t)) \leq exp(-d \cdot S^+(i,j,t) \cdot \epsilon^2/2)$$

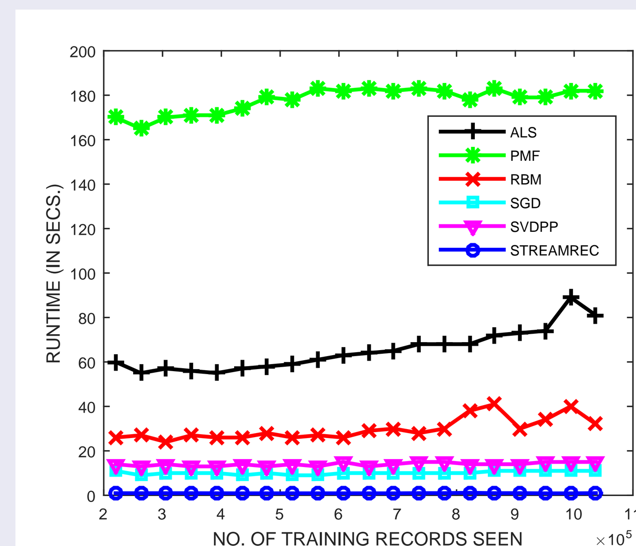
- Upper Tail Bound: For any $\epsilon \in (0, 2 \cdot e - 1)$, $R^+(i, j, t)$ lies outside by a factor of $(1 + \epsilon)$ with the probability:

$$P(R^+(i,j,t) > (1+\epsilon) \cdot S^+(i,j,t)) \leq exp(-d \cdot S^+(i,j,t) \cdot \epsilon^2/4)$$

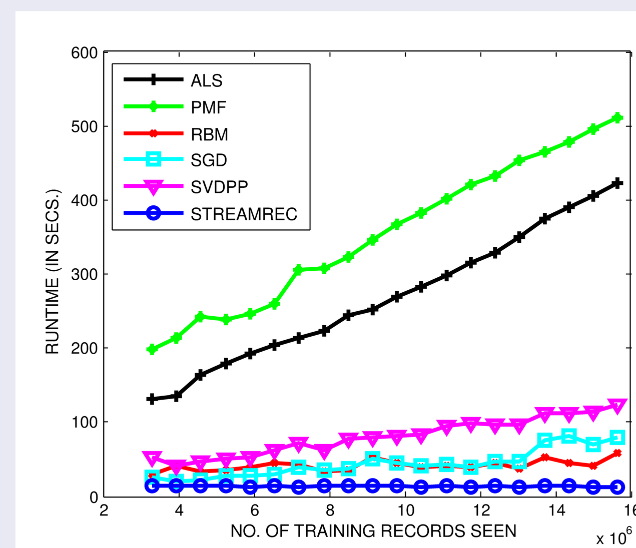
## Summary

- We address the problem of providing recommendations in a streaming setting.
- We provide an efficient algorithm to perform streaming recommendations using a probabilistic model.
- The proposed algorithm stores the rating matrix compactly and hence memory requirements are low.
- Our thorough experiments show that the proposed method performs as good or better than the state-of-the-art.

## Results



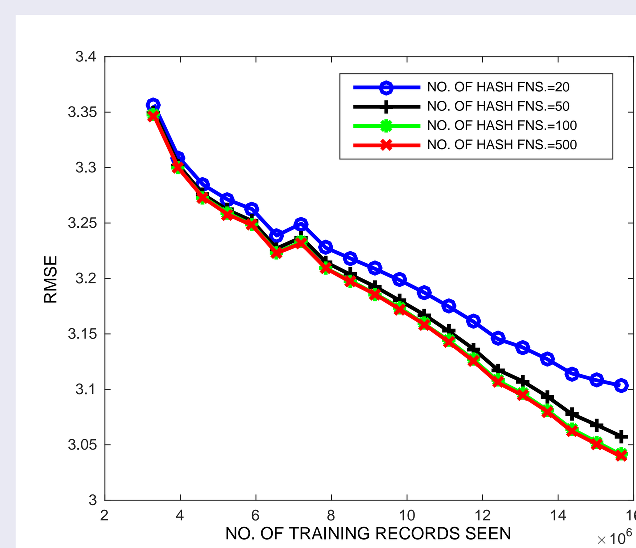

(a) Books Runtime



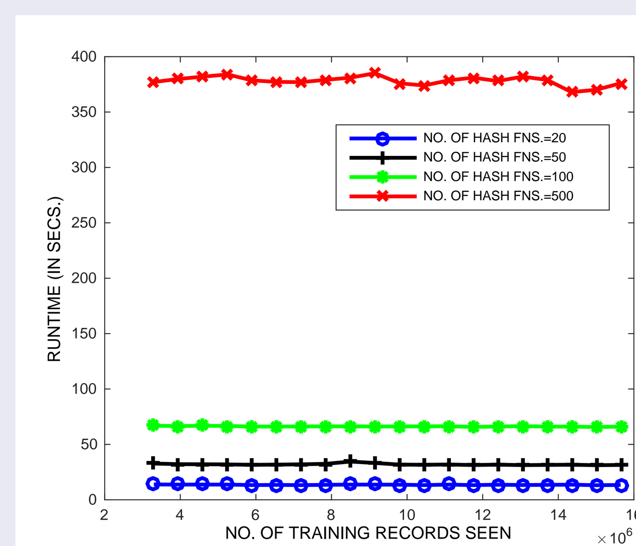

(b) Dating Runtime

Figure: Runtime

## Results



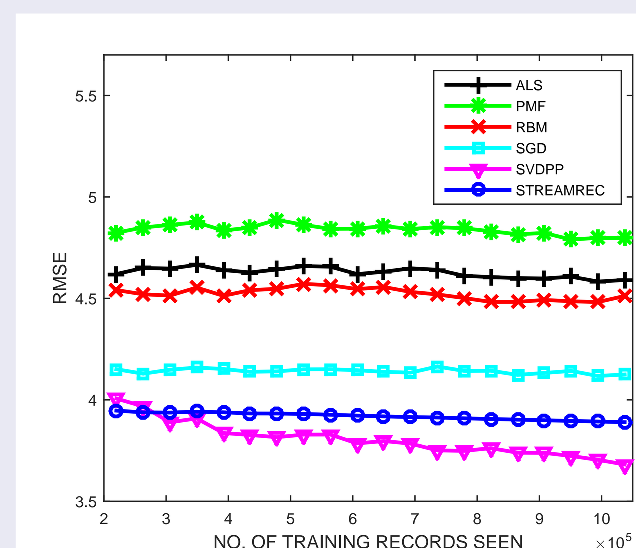

(a) Sensitivity Dating RMSE



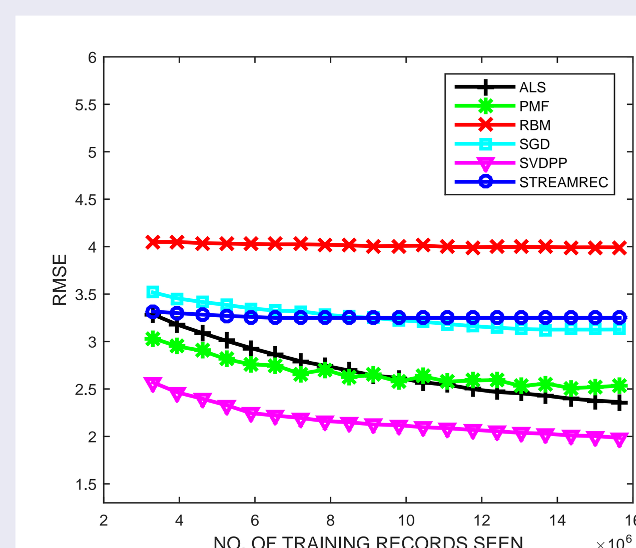

(b) Sensitivity Dating Runtime

Figure: Sensitivity

## Results




(a) Books RMSE




(b) Dating RMSE

Figure: Efficiency