Collaborative Filtering on a Budget

Alexandors Karatzoglou, Alex Smola, Markus Weimer

Outline

1. Main contribution
2. Algorithm details
3. Hashing
4. Modified algorithm with Hashing
5. Results

Main contribution

Use hashing so that required memory is *independent* of number of users and number of user/item features.

Notation

- Let Y be the rating matrix. Y_{ij} is the rating of the i^{th} user for the j^{th} item.
- Let F be the matrix of the predicted ratings.
- Then- $F_{ij} = \langle U_i, M_j \rangle$ and $U_i, M_j \epsilon \mathbb{R}^d$

Regularization

- Standard: $\Omega[U, M] = \frac{1}{2}\left[||U||^2_{Frob} + ||M||^2_{Frob}\right]$
- Proposed: $\Omega[U, M] = \frac{1}{2}\left[\sum_i n_i.||U_i||^2 + \sum_j m_j.||M_j||^2\right]$
- Minimize: $R[U, M] = L[UM^T, Y] + \lambda\Omega[U, M]$

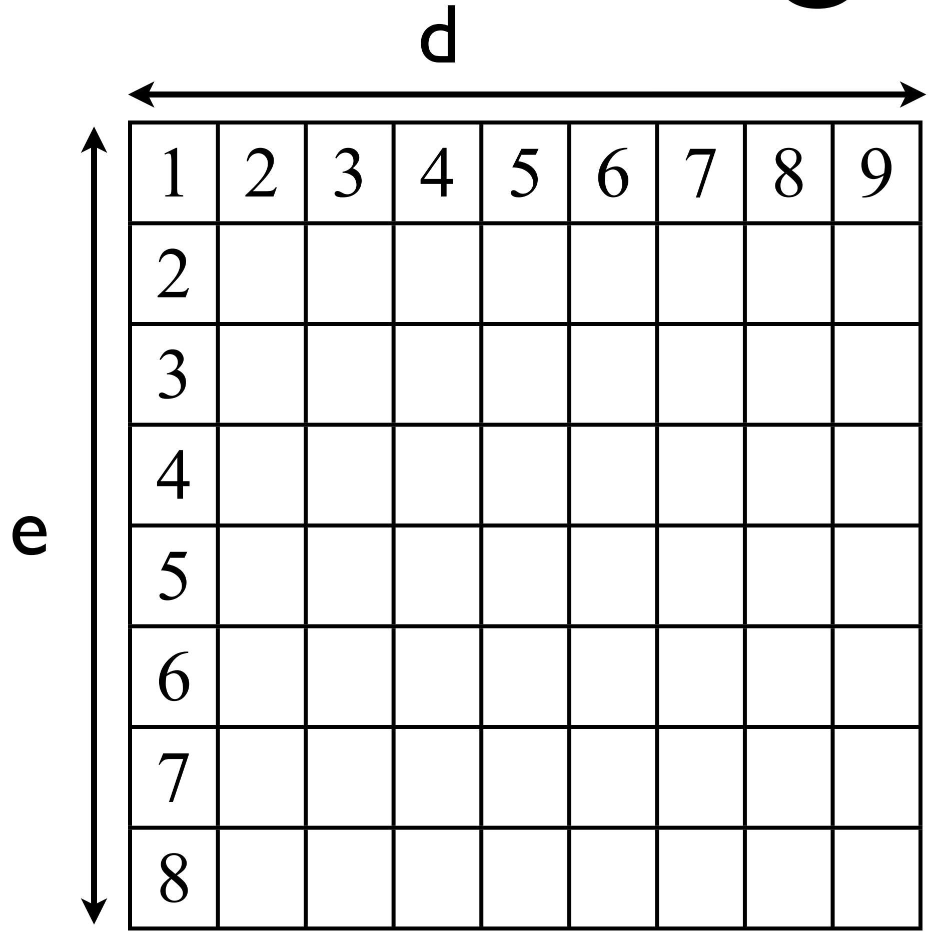
n_i and m_j are scaling factors that depend on the number of ratings

The algorithm

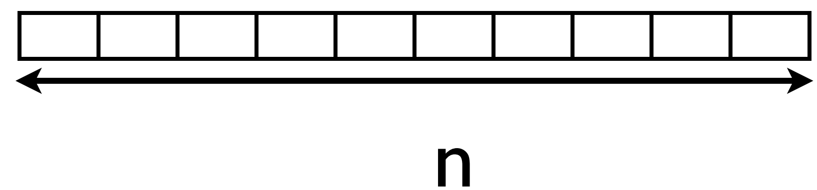
Algorithm 1 Matrix Factorization

Input Y, d
Initialize $U \in \mathbb{R}^{n \times d}$ and $M \in \mathbb{R}^{m \times d}$ with small random values.
Set $t = t_0$
while (i, j) in observations Y **do**
  $\eta \longleftarrow \frac{1}{\sqrt{t}}$ and $t \longleftarrow t + 1$
  $F_{ij} := \langle U_i, M_j \rangle$
  $U_i \longleftarrow (1 - \eta\lambda)U_i - \eta M_j \partial_{F_{ij}} l(F_{ij}, Y_{ij})$
  $M_j \longleftarrow (1 - \eta\lambda)M_j - \eta U_i \partial_{F_{ij}} l(F_{ij}, Y_{ij})$
end while
Output U, M

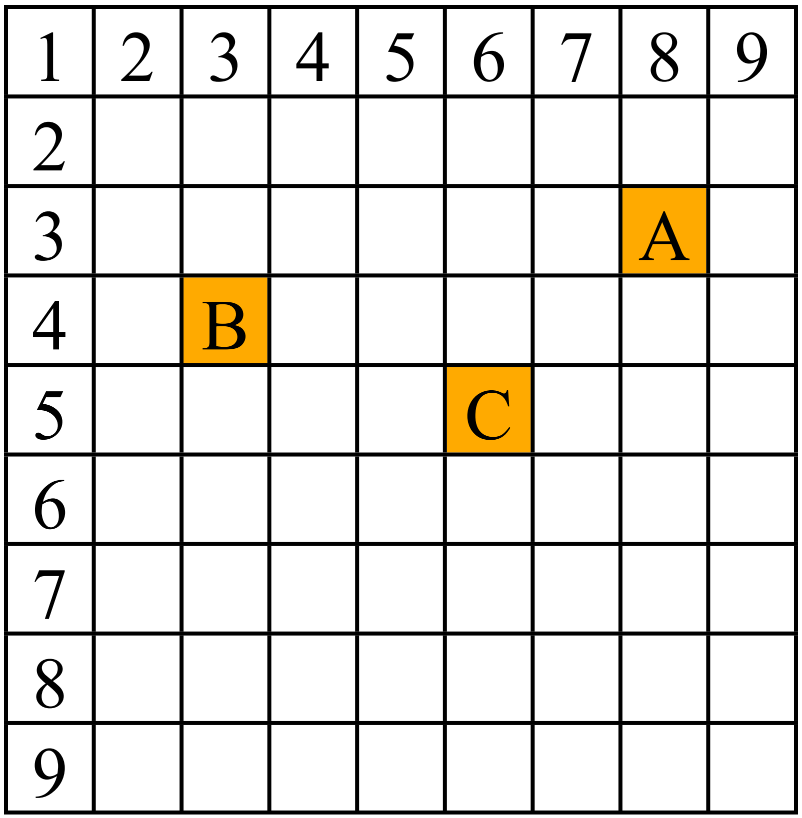
Hashing- The idea

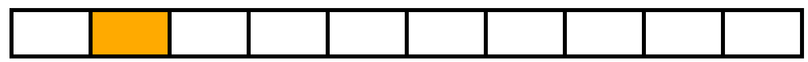


A hash function h, maps h (1 to e, 1 to d) to 1 to n
Say h(3,8) = h (4,3) = h (5,6) =2



Hashing- The idea





$$u_2 = \sum \left(A \cdot \sigma(3,8) + B \cdot \sigma(4,3) + C \cdot \sigma(5,6)\right)$$

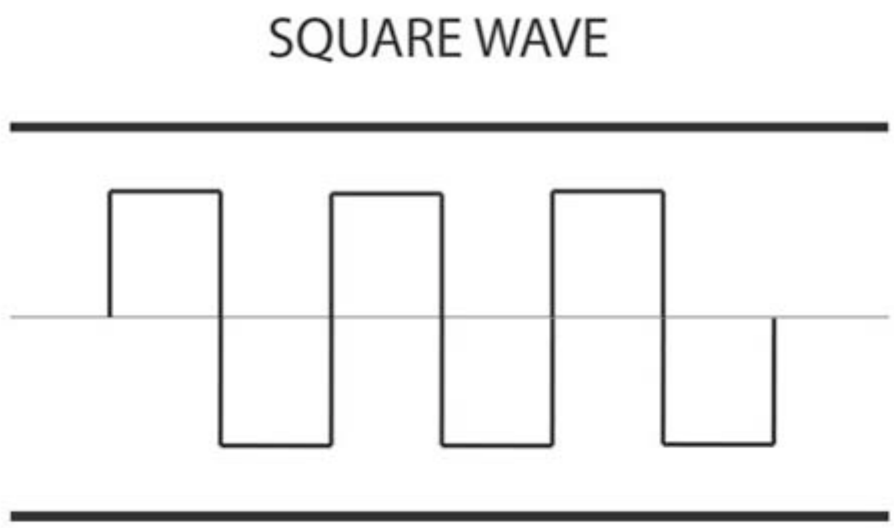
σ is the Rademacher function

Reconstructing: $A = u_2 \cdot \sigma(3,8)$

Works only when a small number of matrix values are significant

Rademacher function

- Also known as the square wave function! Looks like:



Mathematically...

Hashing:

$$u_i := \sum_{(j,k):h(j,k)=i} U_{jk}\sigma(j,k) \text{ and}$$
$$m_i := \sum_{(j,k):h'(j,k)=i} M_{jk}\sigma'(j,k)$$

Reconstruction:

$$\tilde{U}_{ij} := u_{h(i,j)}\sigma(i,j) \text{ and } \tilde{M}_{ij} := m_{h'(i,j)}\sigma'(i,j).$$

This allows us to reconstruct F_{ij} via

$$\tilde{F}_{ik} = \sum_j u_{h(i,j)} m_{h'(k,j)} \sigma(i,j)\sigma'(k,j).$$

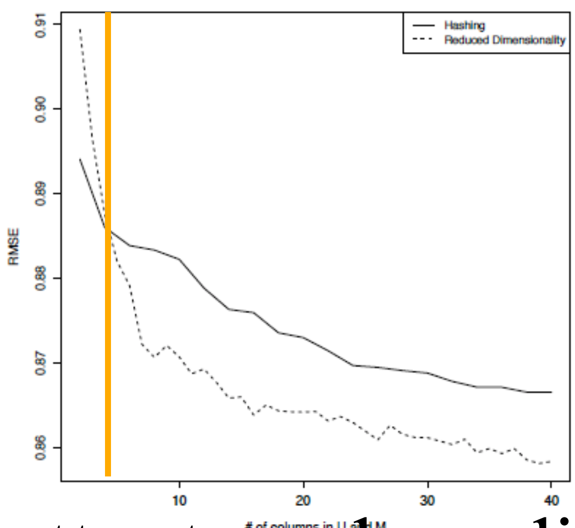
Modified algorithm with hashing

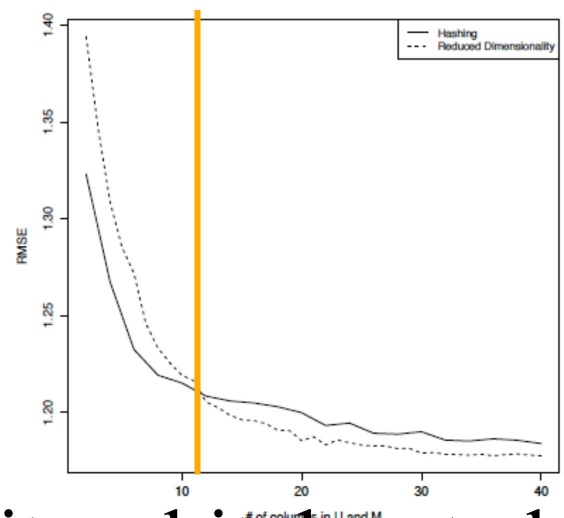
Algorithm 2 Compressed Matrix Factorization

Input Y,
Initialize $w \in \mathbb{R}^N$ with small random values.
Set $t = t_0$
while (i, k) in observations Y **do**
  $\eta \longleftarrow \frac{1}{\sqrt{t}}$ and $t \longleftarrow t + 1$
  $F_{ik} := \sum_j \sigma(i, j)\sigma'(k, j) w_{h(i,j)} w_{h'(k,j)}$
  $\gamma := \eta \partial_{F_{ik}} l(F_{ik}, Y_{ik})$
  $\mu := (1 - \eta\lambda)$
  for $j = 1$ **to** d **do**
    $w_{h(i,j)} \leftarrow \mu w_{h(i,j)} - \gamma \sigma(i, j)\sigma'(k, j) w_{h'(k,j)}$
    $w_{h'(k,j)} \leftarrow \mu w_{h'(k,j)} - \gamma \sigma(k, j)' \sigma(i, j) w_{h(i,j)}$
  end for
end while
Output w

Results

Squared	ϵ-insensitive	Huber
	MovieLens	
$\mathbf{0.857} \pm 0.006$	0.859 ± 0.004	$\mathbf{0.857} \pm 0.004$
	EachMovie	
1.177 ± 0.003	$\mathbf{1.161} \pm 0.007$	1.180 ± 0.005





Better to reduce dimensionality a bit than to hash. Use Hashing only when dimensionality needs to be reduced substantially.

Thank You!