Fast Monte Carlo Algorithms for Computing a Low-Rank Approximation to a Matrix

Vlad Ştefan Burcă

Advisor: Professor Mary Sandoval

Trinity College

Friday, May 2nd, 2014

Recommendation Systems

Widely used today - e-commerce websites, online radio or music streaming services, or even online dating and matchmaking.

Data stored in big matrices; consider data of an music streaming service / online radio.

2 big matrices: **USERS** $\in \mathbb{Z}^{N \times M}$ and **SONGS** $\in \{0,1\}^{M \times F}$.

USERS $\in \mathbb{Z}^{4\times 9}$ for 4 users and 9 songs.

Each user can rate a song on a range from **1** to **5**.

Users only rate a small subset of the entire song database $\Longrightarrow$ **sparse matrix**.

$$\textbf{USERS} = \begin{pmatrix} 3 & & & & 5 & 4 & 2 & & \\ & 1 & & 1 & & & 2 & 5 & \\ 5 & 5 & & 4 & & & & 3 & 2 \\ & & 2 & & 1 & 4 & & & 5 \end{pmatrix}$$

Predict the missing ratings $\Longrightarrow$ recommend the songs with predicted high rating.

HOW? Using the songs matrix, **SONGS** $\in \{0,1\}^{M \times F}$, for an initial *guess* and **Singular Value Decomposition**.

Why can't we just use SVD? Computational limits - the matrices are so big that they can not be stored integrally in the *fast memory* of a PC (Random Access Memory, or **RAM**).

RAM is a scarce resource compared to regular **disk storage**

So maybe we can use the disk storage somehow... $\Longrightarrow$
Fast Monte Carlo Matrix Approximation Algorithms

Consider $A \in \mathbb{R}^{m \times n}$ with $rank(A) = r \leq p = min\{m, n\}$.

Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ denote the positive singular values of A, in decreasing order, and $\sigma_{r+1} = \cdots = \sigma_n = 0$.

Let $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ be the orthogonal matrices whose columns are the right singular vectors v_i's and, respectively, the left singular vectors u_i's.

$$U = \begin{pmatrix} u_1 & \vdots & u_2 & \vdots & \cdots & \vdots & u_m \end{pmatrix} V = \begin{pmatrix} v_1 & \vdots & v_2 & \vdots & \cdots & \vdots & v_n \end{pmatrix}$$

Let $\Sigma \in \mathbb{R}^{m \times n}$ denote the the matrix of the form

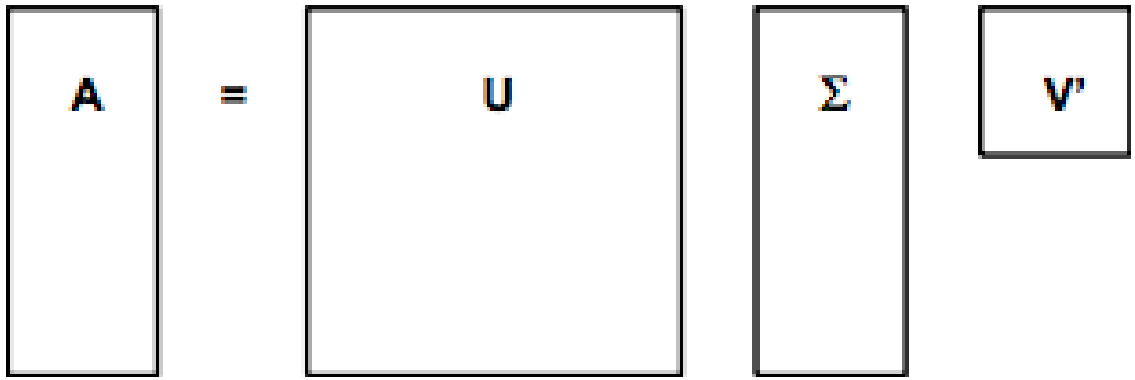
$$\Sigma = \begin{pmatrix} D & \vdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \vdots & 0 \end{pmatrix}, \text{where } D \in \mathbb{R}^{r \times r} \text{ is of the form}$$

$$D = \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{pmatrix}$$

Theorem (Singular Value Decomposition Theorem)

Any $m \times n$ matrix can be factored into a singular value decomposition (SVD),

$$A = U\Sigma V^{\mathsf{T}}$$



Theorem (Eckart-Young Theorem)

Let the SVD of A be given by the above equation. If $k < r = rank(A)$ and $A_k = \sum_{i=1}^{k} u_i \sigma_i v_i^T$, then

$$\min_{rank(B)=k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}$$

$$\min_{rank(B)=k} \|A - B\|_F^2 = \|A - A_k\|_F^2 = \sum_{i=k+1}^{r} \sigma_i^2$$

What does this mean? That the A_k constructed from the k largest singular triplets of A is the optimal $rank - k$ approximation to A with respect to both $\|\cdot\|_F$ and $\|\cdot\|_2$.

Why do we care about this? We want to be as close as possible to this *optimal* value when we use other approximating algorithms.

And we will be close enough (up to an additional error factor that is dependent on the way we are constructing the approximating matrix).

Due to multiplication of big sized matrices, in computing the SVD, the algorithm will perform a number of operations that is greater than $O(x)$.

That means that the running time of the algorithm increases faster than the size of the problem - in our case, the size of the problem is directly defined by the size of matrix A: m and n.

Input: the matrix $A \in \mathbb{R}^{m \times n}$, integers c, k and a probability distribution.

Goal: produce as output an approximation of the top k singular values and the corresponding left singular vectors of A.

Sample c columns of matrix A into $C \in \mathbb{R}^{m \times c}$ according to the probability distribution given at input.

Compute C's right singular vectors and the associated singular values through the regular SVD method.

Compute the left singular vectors of C and construct matrix $H_k \in \mathbb{R}^{m \times k}$ - the approximation of the left singular vectors associated to the top k singular values of A.

Algorithm 1 `LinearTimeSVD` Algorithm

Require: $A \in \mathbb{R}^{m \times n}$, $c, k \in \mathbb{Z}^+$ such that $1 \leq k \leq c \leq n$, $\{p_i\}_{i=1}^n$ such that $p_i \geq 0$ and $\sum_{i=1}^n p_i = 1$

Ensure: $H_k \in \mathbb{R}^{m \times k}$ and $\sigma_t(C)$, $t = 1, \ldots, k$.

1: **procedure** LINEARTIMESVD($A, k, c, \{p_i\}_{i=1}^n$)
2: **for** $t = 1$ to c **do**
3: Pick $i_t \in 1, \ldots, n$ with $\mathbf{Pr}[i_t = \alpha] = p_\alpha$, $\alpha = 1, \ldots, n$
4: Set $C^{(t)} = A^{(i_t)} / \sqrt{c p_{i_t}}$
5: **end for**

Algorithm 2 `LinearTimeSVD` Algorithm

6: Compute $C^\intercal C$ and its SVD. Thus $C^\intercal C = \sum_{t=1}^{c} \sigma_t^2(C) y^t {y^t}^T$
7: Compute $h^t = Cy^t/\sigma_t(C)$ for $t = 1, \ldots, k$
8: Construct H_k, where $H_k^{(t)} = h^{(t)}$

9: **return** H_k and $\sigma_t(C)$, $t = 1, \ldots, k$
10: **end procedure**

Sampling c columns. $O(c)$

Constructing $C^\intercal C$. $O(mc^2)$

Computing SVD of $C^\intercal C$. $O(c^3)$

Overall time complexity. $O(mc^2 + c^3)$

Linear in terms of the sizes of A. c is the sampling count, defined by the user at input.

Regular SVD approximation of A: A_k

Fast Monte Carlo approximation of A: $D_k = H_k H_k^\mathsf{T} A$

The error of D_k is dependent on the error of the best approximation, A_k, along with an additional error term of the form $\|AA^\mathsf{T} - CC^\mathsf{T}\|_F$.

Theorem (Frobenius error of D_k)

Suppose $A \in \mathbb{R}^{m \times n}$ and let H_k be constructed from the `LinearTimeSVD` algorithm. Then

$$\|A - H_k H_k^{\mathsf{T}} A\|_F^2 \leq \|A - A_k\|_F^2 + 2\sqrt{k}\|AA^{\mathsf{T}} - CC^{\mathsf{T}}\|_F$$

Theorem (Spectral error of D_k)

Suppose $A \in \mathbb{R}^{m \times n}$ and let H_k be constructed from the `LinearTimeSVD` algorithm. Then

$$\|A - H_k H_k^{\mathsf{T}} A\|_2^2 \leq \|A - A_k\|_2^2 + 2\|AA^{\mathsf{T}} - CC^{\mathsf{T}}\|_2$$

Theorem (Sampling advantage of the `LinearTimeSVD`)

Suppose $A \in \mathbb{R}^{m \times n}$*; let* H_k *be constructed from the* `LinearTimeSVD` algorithm by sampling c columns of A with probabilities $\{p_i\}_{i=1}^{n}$ such that $p_i \geq \beta \left|A^{(i)}\right|^2 / \|A\|_F^2$ for some $0 < \beta \leq 1$, and let $\eta = 1 + \sqrt{(8/\beta) log(1/\delta)}$. Let $\epsilon > 0$. If $c \geq 4k/\beta\epsilon^2$, then

$$\mathbf{E}\left[\|A - H_k H_k^{\mathsf{T}} A\|_F^2\right] \leq \|A - A_k\|_F^2 + \epsilon \|A\|_F^2,$$

and if $c \geq 4k\eta^2/\beta\epsilon^2$, then with probability at least $1 - \delta$,

$$\|A - H_k H_k^{\mathsf{T}} A\|_F^2 \leq \|A - A_k\|_F^2 + \epsilon \|A\|_F^2,$$

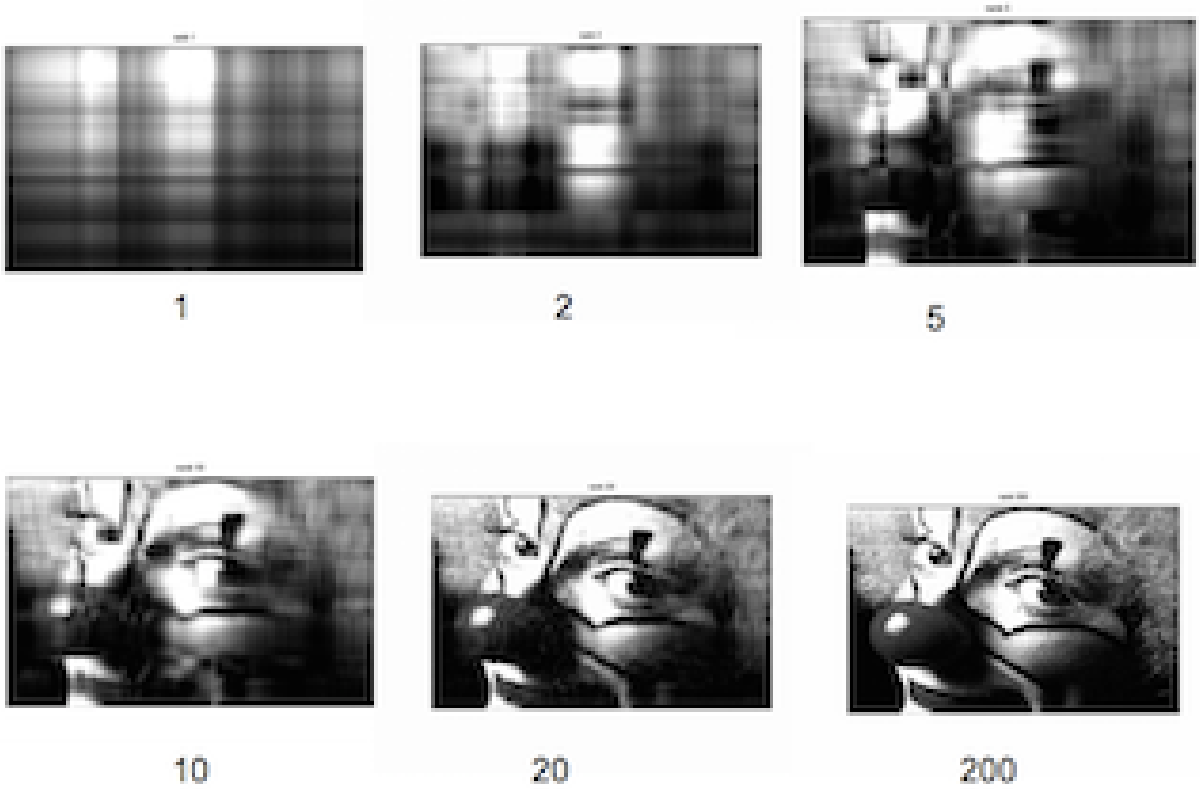
$$p_i \geq \beta \left| A^{(i)} \right|^2 / \|A\|_F^2$$

We now have an algorithm that does not have a running time super linear in terms of the sizes of A and that has a reasonable error bound.

But it works on full matrices - we have sparse matrices for recommendation systems.

Preprocessing has to be done that assigns some initial weights to the missing values, based on the similarities between songs.

With those weights filling in the empty ratings, we can apply the linear time SVD algorithm.


1
2
5
10
20
200

Acknowledgements

Thank you!

Thesis Advisor: Professor Mary Sandoval
Major Advisor: Professor Melanie Stein

Trinity College Mathematics Department