Matrix Completion

Hon Ming Chun

The Problem

- Recovering a large matrix from a small subset of its entries (the famous Netflix problem).

Item

User		W	X	Y	Z
	A		4.5	2.0	
	B	4.0		3.5	
	C		5.0		2.0
	D		3.5	4.0	1.0

Rating Matrix

The Problem

- Recovering a large matrix from a small subset of its entries

  (the famous Netflix problem).

- The matrix should be of low-rank.

  (User preferences can often be described by a few factors, such as the movie genre and time of release.)

- The matrix is extremely sparse

  (The ratings matrix could be about 99% sparse. This make sense since user can only watch and rate a few movies out of the whole movie list)

Solving with singular value thresholding (SVT)

- Originally:
$$\min_{X} \quad \text{rank}(X) \qquad \text{subject to} \quad X_{ij} = M_{ij} \;\; \forall i, j \in E$$

- Convex relaxation:
$$\text{minimize} \quad \|\boldsymbol{X}\|_* \qquad \text{subject to} \quad X_{ij} = M_{ij}, \quad (i,j) \in \Omega,$$

  Ω is the set of location of the nonzero entries of the matrix M

- Orthogonal projection:
$$\text{minimize} \quad \|\boldsymbol{X}\|_* \qquad \text{subject to} \quad \mathcal{P}_\Omega(\boldsymbol{X}) = \mathcal{P}_\Omega(\boldsymbol{M}),$$

  P_Ω project onto the span of matrices vanishing outside of Ω so that the (i, j)th component of $P_\Omega(X)$ is equal to X_{ij} if (i, j) ∈ Ω and zero otherwise

Solving with SVT (cont)

- Orthogonal projection:

$$\begin{array}{ll} \text{minimize} & \|\boldsymbol{X}\|_* \\ \text{subject to} & \mathcal{P}_\Omega(\boldsymbol{X}) = \mathcal{P}_\Omega(\boldsymbol{M}), \end{array}$$

- Similar minimization:

$$\begin{array}{ll} \text{minimize} & \tau\|\boldsymbol{X}\|_* + \frac{1}{2}\|\boldsymbol{X}\|_F^2 \\ \text{subject to} & \mathcal{P}_\Omega(\boldsymbol{X}) = \mathcal{P}_\Omega(\boldsymbol{M}). \end{array}$$

(Final target)

τ is the parameter threshold 'T' in the SVT

This minimization converges as $\tau \to \infty$

SVT solve matrix completion using an iterative algorithm with a large τ

Inspired by

- SVT was inspired by the linearized Bregman iterations and iterative soft-thresholding algorithms which can be used in L1 minimization

- SVT can be regarded as an extension of classic iterative soft-thresholding since it finds a sparse vector of singular values and the bases instead of finds sparse solutions

- Classic iterative soft-thresholding algorithms fails to solve the target minimization and have different limits

SVT basic idea

- Iterate two matrices X and Y until the escaping condition is reached

  X and Y both are n x m which is the same size as M
- Set Y = M initially
- Each iteration we calculate the truncated SVD of Y (soft-thresholding) and use only the dominant singular values and singular vectors to calculate X
- Calculate the new Y using the new X and old Y
- Break the loop if error is small or maximum iteration is reached
- Set Xopt = X where Xopt is the solution

SVT basic idea

- $$\begin{cases} \boldsymbol{X}^k = \mathcal{D}_\tau(\boldsymbol{Y}^{k-1}), \\ \boldsymbol{Y}^k = \boldsymbol{Y}^{k-1} + \delta_k \mathcal{P}_\Omega(\boldsymbol{M} - \boldsymbol{X}^k) \end{cases}$$

where $\mathcal{D}_\tau$ is called the soft-thresholding operator

$$\mathcal{D}_\tau(\boldsymbol{X}) := \boldsymbol{U}\mathcal{D}_\tau(\boldsymbol{\Sigma})\boldsymbol{V}^*, \quad \mathcal{D}_\tau(\boldsymbol{\Sigma}) = \mathrm{diag}(\{\sigma_i - \tau)_+\})$$

$$\mathcal{D}_\tau(\boldsymbol{Y}) = \arg\min_{\boldsymbol{X}} \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{Y}\|_F^2 + \tau\|\boldsymbol{X}\|_*$$

- Essentially, $\mathcal{D}_\tau(\boldsymbol{Y}^{k-1})$ is the truncated SVD of Y^{k-1}
- But what is $\boldsymbol{Y}^{k-1} + \delta_k \mathcal{P}_\Omega(\boldsymbol{M} - \boldsymbol{X}^k)$?

Lagrange multiplier and SVT

- Lagrangian multipliers find local extrema of f(x) subjected to a constrant g(x)=0 using the lagrangian function $\mathcal{L}(x,\lambda) = f(x) + \lambda g(x)$

- Recall
$$\begin{array}{ll} \text{minimize} & \tau\|\boldsymbol{X}\|_* + \frac{1}{2}\|\boldsymbol{X}\|_F^2 \\ \text{subject to} & \mathcal{P}_\Omega(\boldsymbol{X}) = \mathcal{P}_\Omega(\boldsymbol{M}). \end{array}$$

$$\mathcal{L}(\boldsymbol{X},\boldsymbol{Y}) = f_\tau(\boldsymbol{X}) + \langle \boldsymbol{Y}, \mathcal{P}_\Omega(\boldsymbol{M}-\boldsymbol{X})\rangle$$

where $f_\tau(\boldsymbol{X}) = \tau\|\boldsymbol{X}\|_* + \frac{1}{2}\|\boldsymbol{X}\|_F^2$

Lagrange multiplier and SVT

- Use Uzawa's algorithm which minimize by finding saddle point and is a subgradient method

$$\begin{cases} \mathcal{L}(\boldsymbol{X}^k, \boldsymbol{Y}^{k-1}) = \min_{\boldsymbol{X}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}^{k-1}), \\ \boldsymbol{Y}^k = \boldsymbol{Y}^{k-1} + \delta_k \mathcal{P}_\Omega(\boldsymbol{M} - \boldsymbol{X}^k), \end{cases}$$

- Since $\partial_{\mathbf{Y}} g_0(\boldsymbol{Y}) = \partial_{\mathbf{Y}} \mathcal{L}(\tilde{\boldsymbol{X}}, \boldsymbol{Y}) = \mathcal{P}_\Omega(\boldsymbol{M} - \tilde{\boldsymbol{X}})$

  then $\boldsymbol{Y}^k = \boldsymbol{Y}^{k-1} + \delta_k \partial_{\mathbf{Y}} g_0(\boldsymbol{Y}^{k-1}) = \boldsymbol{Y}^{k-1} + \delta_k \mathcal{P}_\Omega(\boldsymbol{M} - \boldsymbol{X}^k)$

- Note $\arg\min\ f_\tau(\boldsymbol{X}) + \langle \boldsymbol{Y}, \mathcal{P}_\Omega(\boldsymbol{M} - \boldsymbol{X}) \rangle = \arg\min\ \tau \|\boldsymbol{X}\|_* + \frac{1}{2}\|\boldsymbol{X} - \mathcal{P}_\Omega \boldsymbol{Y}\|_F^2$

- Minimizer is $\mathcal{D}_\tau(\mathcal{P}_\Omega(\boldsymbol{Y}))$ and since $\boldsymbol{Y}^k = \mathcal{P}_\Omega(\boldsymbol{Y}^k)$

$$\begin{cases} \boldsymbol{X}^k = \mathcal{D}_\tau(\boldsymbol{Y}^{k-1}), \\ \boldsymbol{Y}^k = \boldsymbol{Y}^{k-1} + \delta_k \mathcal{P}_\Omega(\boldsymbol{M} - \boldsymbol{X}^k) \end{cases}$$

The algorithm

ALGORITHM 1. SINGULAR VALUE THRESHOLDING (SVT) ALGORITHM.

Input: sampled set Ω and sampled entries $\mathcal{P}_\Omega(\boldsymbol{M})$, step size δ, tolerance ϵ, parameter τ, increment ℓ, and maximum iteration count $k_{\max}$
Output: $\boldsymbol{X}^{\text{opt}}$
Description: Recover a low-rank matrix $\boldsymbol{M}$ from a subset of sampled entries

Set $\boldsymbol{Y}^0 = k_0 \delta \mathcal{P}_\Omega(\boldsymbol{M})$ (k_0 is defined in (5.3))
Set $r_0 = 0$
for $k = 1$ **to** $k_{\max}$
  Set $s_k = r_{k-1} + 1$
  repeat
    Compute $[\boldsymbol{U}^{k-1}, \boldsymbol{\Sigma}^{k-1}, \boldsymbol{V}^{k-1}]_{s_k}$
    Set $s_k = s_k + \ell$
  until $\sigma^{k-1}_{s_k-\ell} \leq \tau$
  Set $r_k = \max\{j : \sigma^{k-1}_j > \tau\}$
  Set $\boldsymbol{X}^k = \sum_{j=1}^{r_k} (\sigma^{k-1}_j - \tau)\boldsymbol{u}^{k-1}_j \boldsymbol{v}^{k-1}_j$
  if $\|\mathcal{P}_\Omega(\boldsymbol{X}^k - \boldsymbol{M})\|_F / \|\mathcal{P}_\Omega \boldsymbol{M}\|_F \leq \epsilon$ **then break**
  Set $Y^k_{ij} = \begin{cases} 0 & \text{if } (i,j) \notin \Omega, \\ Y^{k-1}_{ij} + \delta(M_{ij} - X^k_{ij}) & \text{if } (i,j) \in \Omega \end{cases}$
end *for* k
Set $\boldsymbol{X}^{\text{opt}} = \boldsymbol{X}^k$

Ω : location of nonzero entries
δ : step size
ϵ : loop escaping tolerance
τ : threshold parameter
ℓ : increment to s_k if $\sigma^{k-1}_{s_k-\ell} > \tau$
K_{max}: maximum iteration count
s_k : number singular values of Y^{k-1} to be computed at the kth iteration
R_k: rank(X^k)

The algorithm (remark)

- The solution is of low-rank since most singular value (factors) of the original M is “kill” by the parameter threshold τ
- Y^k is always sparse with the project operator P_Ω
- The SVD and the respective X^k can be computed quickly

Code

- One important part of the SVT algorithm is that it computes the truncated SVD instead of the full SVD to improve performance

- To implement this part of the SVT, we can use a library called sparsesvd

Documentation

The *sparsesvd* module offers a single function, *sparsesvd*, which accepts two parameters. One is a sparse matrix in the *scipy.sparse.csc_matrix* format, the other the number of requested factors (an integer):

- u1, s1, v1 = sparsesvd(ss.csc_matrix(Y), s)
- The first argument is the matrix to decomposed and second argument is the number of eigenvalue we want to calculate

Code

- The original paper use relative error

$$\text{relative error} = \|\boldsymbol{X}^{\text{opt}} - \boldsymbol{M}\|_F / \|\boldsymbol{M}\|_F$$

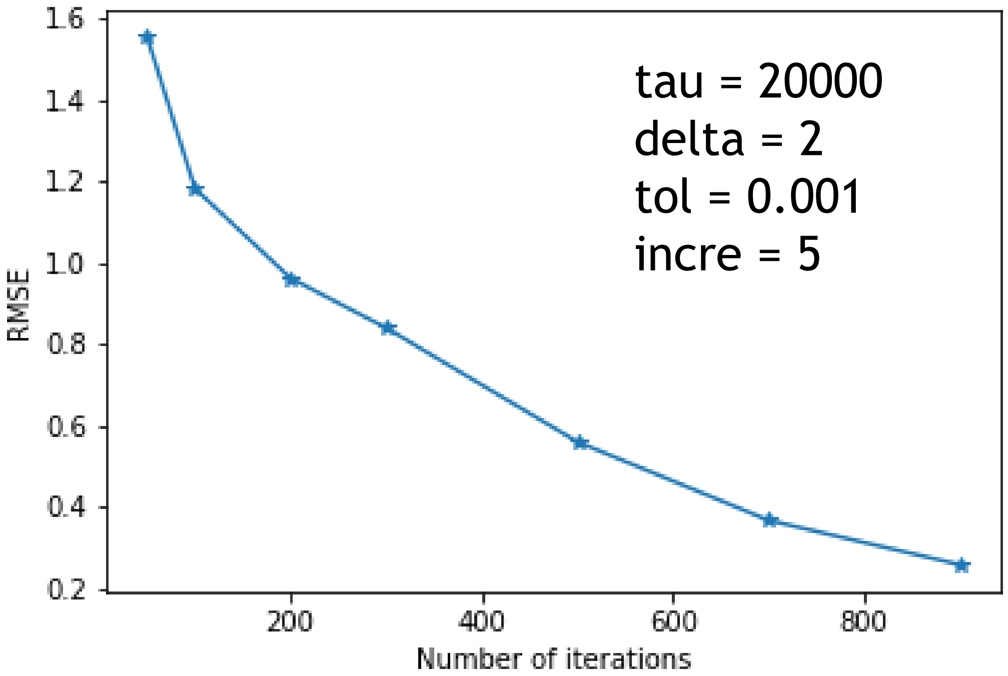
- Use rmse to measure because there is no such thing as the real unknown matrix in real movie data set
- The rmse is measured using the nonzero entries of the predicted matrix and the original matrix

Code

- The algorithm is demonstrated using the MovieLens 1M Dataset
- 1,000,209 ratings with 6040 users and 3883 movies
- It is about 95% sparse

```
array([[ 5., nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       ...,
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [ 3., nan, nan, ..., nan, nan, nan]])
```

Code

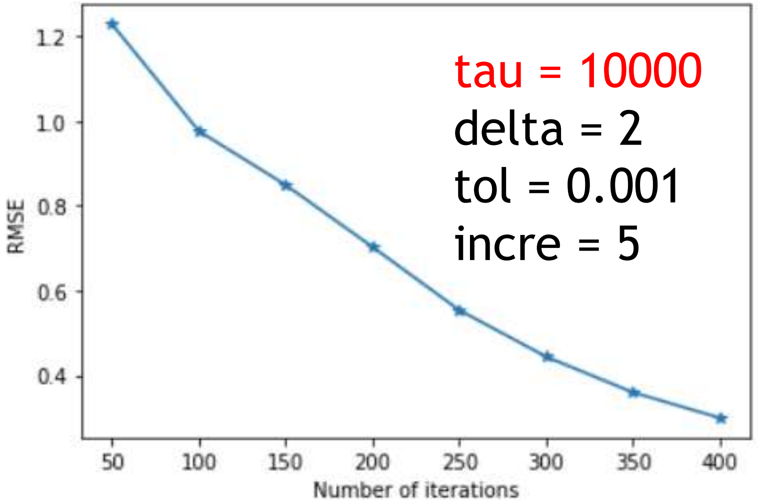


Time:4610s RMSE: 0.257 Rank=153

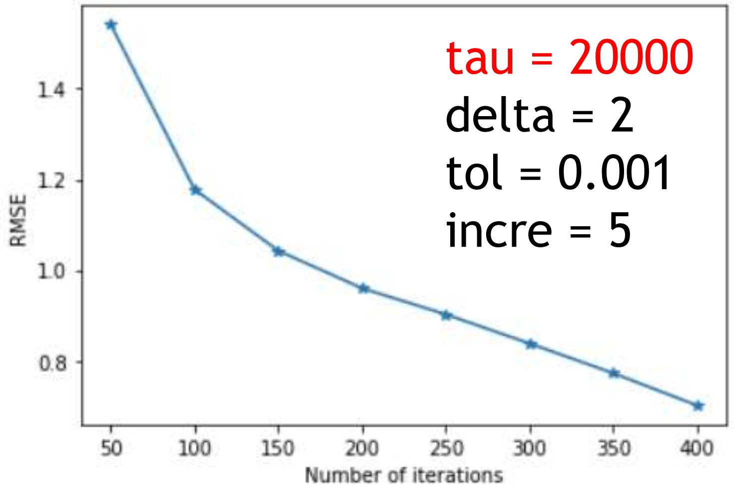
- The dataset matrix is not that low rank and the computation time is quite high
- Note that the computation become slower as the iteration number increases
- The following comparisons use 400 iterations instead of the full iterations

Code

step size δ, tolerance ϵ, parameter τ, increment ℓ



Time:2839s RMSE: 0.300 Rank=140

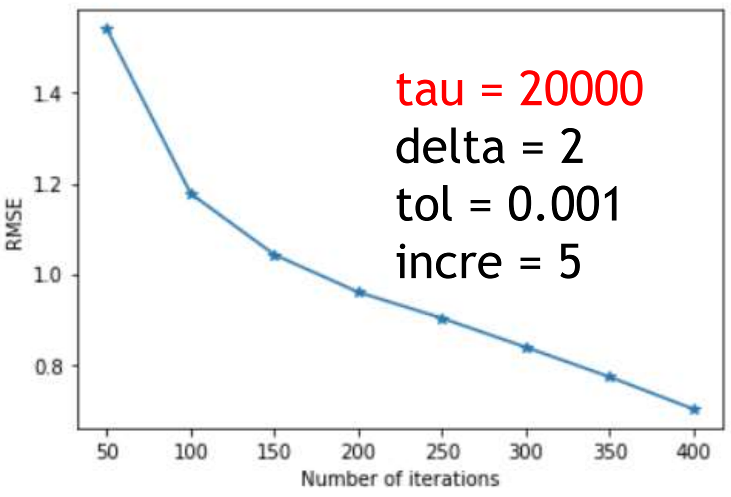


Time:735s RMSE: 0.701 Rank=35

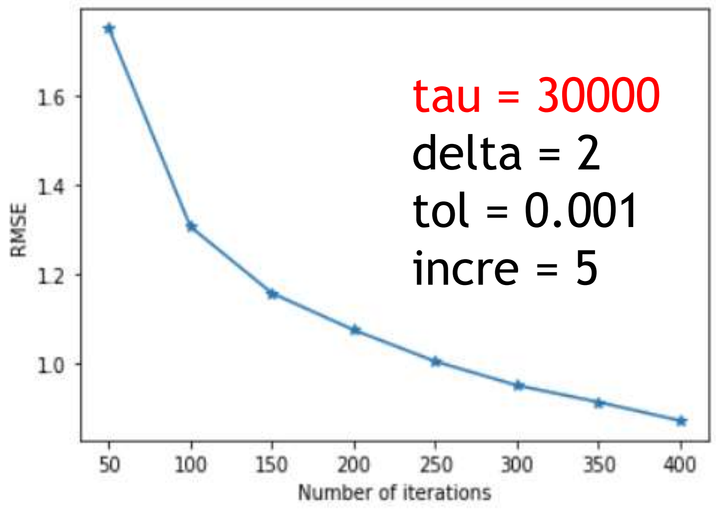
Low tau require more SVD computation each iteration, resulting high computation time

Code

step size δ, tolerance ϵ, parameter τ, increment ℓ



Time:735s RMSE: 0.701 Rank=35

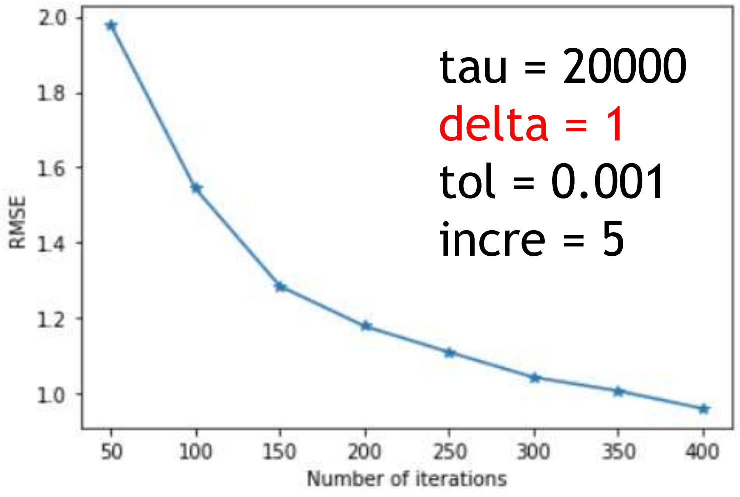


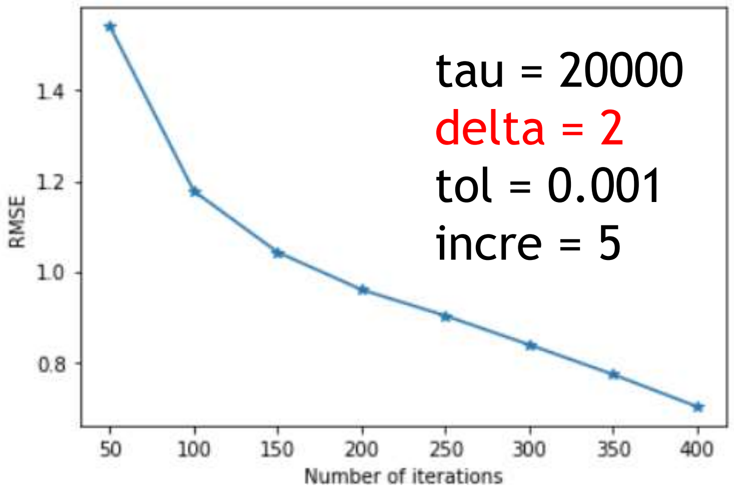
Time:98s RMSE: 0.872 Rank=10

High tau require less SVD computation but it may be too conservative and slow to converge

Code

step size δ, tolerance ϵ, parameter τ, increment ℓ





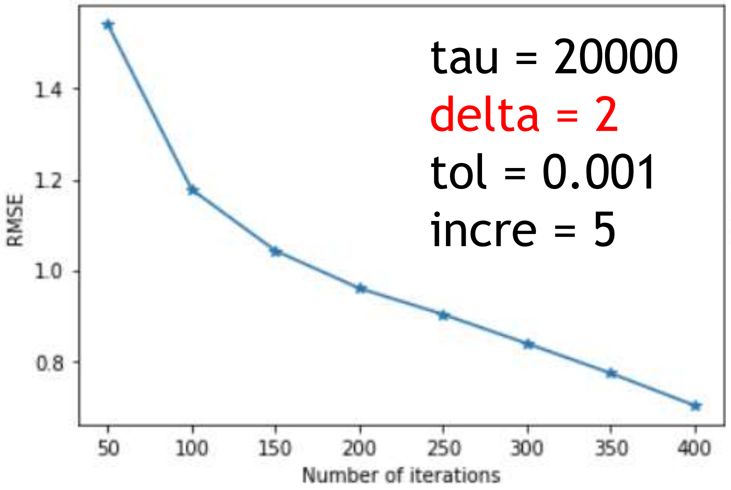
Time:94s RMSE: 0.96 Rank=6

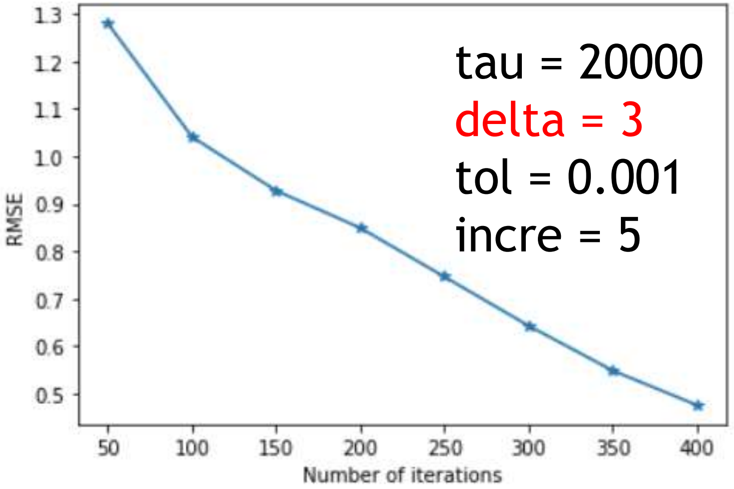
Time:735s RMSE: 0.701 Rank=35

Lower delta (0-2) guarantee to converge,
but the converge speed is slow
(takes more iterations to converge)

Code

step size δ, tolerance ϵ, parameter τ, increment ℓ





Time:735s RMSE: 0.701 Rank=35

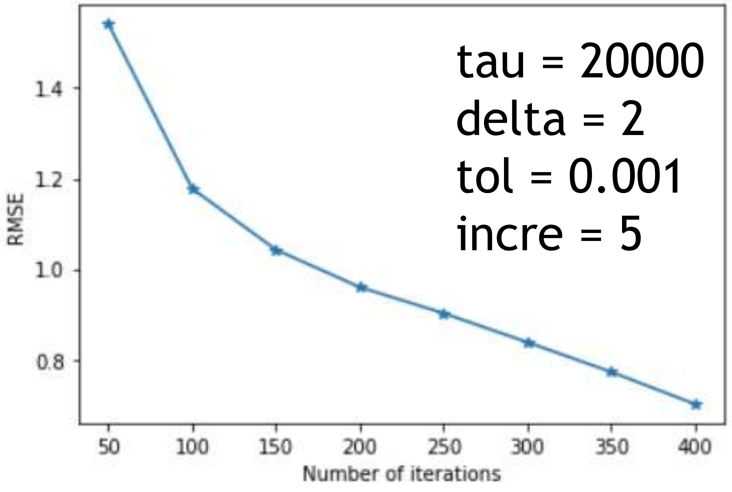
Time:2169s RMSE: 0.476 Rank=95

Higher delta may increase the converge speed
but it may not converge if delta is too high

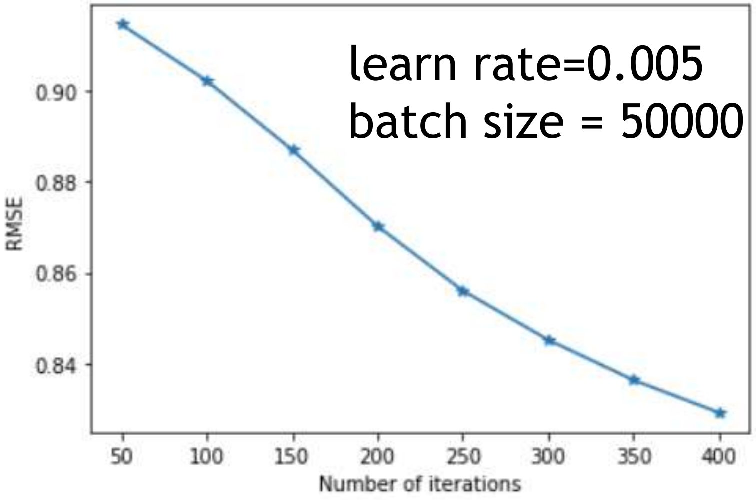
Comparison (stochastic gradient descent)

- The SVT can be seen as a projected gradient-descent algorithms
- compare SVT with a stochastic gradient descent matrix factorization approach
- It is basically the gradient descent approach but we only use a mini-batch for each step instead of using all the data

Comparison (stochastic gradient descent)



SVT
Time:735s RMSE: 0.701



SGD
Time:665s RMSE: 0.829

With similar amount of time, SVT can reach a much lower RMSE and perform better compared to the classic stochastic gradient descent approach

Pros and Cons of SVT

Pros:

- make good use of the characteristics of sparsity and low rank of the problem to increase performance
- relatively easy to understand and implement

Cons:

- the origin matrix need to be very low rank or else it may not converge or has poor performance
- It is an algorithm published in 2008, there are new algorithms with better performances created during this 14 years

Algorithms for matrix completion

- Convex relaxation (SVT)
- Gradient descent
- Alternating least squares minimization (AltMinComplete)

References

- https://cpb-us-w2.wpmucdn.com/blog.nus.edu.sg/dist/d/11132/files/2019/01/SVT-112fnaa.pdf
- https://pypi.org/project/sparsesvd/
- https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31
- https://dl.acm.org/doi/10.1145/2488608.2488693