Machine Learning: Recommender System

Prof. Mingkui Tan

SCUT Machine Intelligence Laboratory (SMIL)

Contents

Contents

Example: Recommender System

Example: Recommender System

Recommender System

- Recommender System applies statistics and knowledge discovery techniques to the problem of making product recommendations

- **Advantages of recommender systems:**

  - **Improve conversion rate:**

    Help customers find what they want to buy

  - **Cross-selling:**

    Suggest additional products

  - **Improve customer loyalty:**

    Create a value-added relationship

Use Cases of Recommendation Systems

- **Netflix:** 2/3 of the watched movies are recommended

- **Amazon:** 35% sales come from recommendations

- **Google News:** recommendations generate 38% more click-throughs

- **Choicestream:** 28% of the people would buy more music if they found what they liked

Collaborative Filtering

Make **automatic predictions (filtering)** about the interests of a user by collecting preferences or taste information **from many other users (collaboration)**

Collaborative Filtering Algorithms

- **Memory-based CF:** utilize the entire user-item database to generate a prediction
    - **User-based CF:** find similar users to predict ratings
    - **Item-based CF**: use similar items to predict ratings
- **Model-based CF:** build a model from the rating data (Matrix factorization, etc.) and use this model to predict missing ratings

Contents

Model-based CF Algorithms

■ Models are learned from the **underlying data** rather than heuristics models of user ratings (or purchases):

■ **Matrix Factorization**

■ Clustering (classification)

■ Association Rules

■ Deep learning based methods

■ Other models

Matrix Factorization is the **most widely used** algorithm

Matrix Factorization

- Given a rating matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$, with sparse ratings from m users to n items

- Assume rating matrix $\mathbf{R}$ can be factorized into the multiplication of two **low-rank feature matrices** $\mathbf{P} \in \mathbb{R}^{m \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times n}$
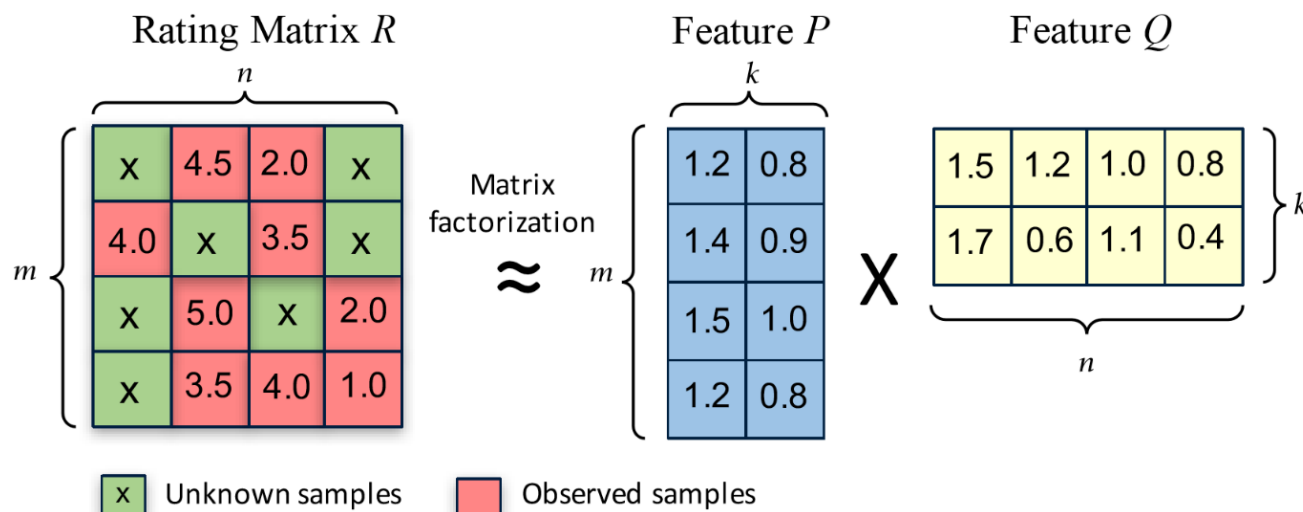
An example of matrix factorization ($m = 4, n = 4, k = 2$)

Determine Objective Function

- Determine an **objective function**

    - Squared error loss:
    $$\mathcal{L}(r_{ui}, \hat{r}_{ui}) = (r_{ui} - \hat{r}_{ui})^2$$

    - Binary hinge loss:
    $$\mathcal{L}(r_{ui}, \hat{r}_{ui}) = \max(0, 1 - r_{ui}\hat{r}_{ui})$$

- Notes: r_{ui} denotes the **actual rating** of user u for item i and $\hat{r}_{ui}$ denotes the **prediction**

Alternating Least Square (ALS) for MF

- ALS is to minimize the following objective function:

$$\mathcal{L} = \sum_{u,i} (r_{u,i} - \mathbf{p}_u^{\mathrm{T}} \mathbf{q}_i)^2 + \lambda \left(\sum_u \|\mathbf{p}_u\|^2 + \sum_i \|\mathbf{q}_i\|^2 \right)$$

- $\mathbf{P} = [\mathbf{p}_1, \mathbf{p}_2 \cdots, \mathbf{p}_m]^{\mathrm{T}} \in \mathbb{R}^{m \times k}$

- $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2 \cdots, \mathbf{q}_n] \in \mathbb{R}^{k \times n}$

- $r_{u,i}$ denotes the **actual rating** of user u for item i, $u = 1 \cdots m, i = 1 \cdots n$

- λ is **regularization parameter** to avoid overfitting

General Steps of ALS

Algorithm 1 General Steps of ALS

1: **Require** rating matrix **R**, feature matrices **P**, **Q** and regularization parameter λ

2: **Optimize P** while fixing **Q**

3: **Optimize Q** while fixing **P**

4: **Repeat** the above processes until **convergence**

■ Objective function:

$$\mathcal{L} = \sum_{u,i}(r_{u,i} - \mathbf{p}_u^\mathrm{T}\mathbf{q}_i)^2 + \lambda(\sum_u\|\mathbf{p}_u\|^2 + \sum_i\|\mathbf{q}_i\|^2)$$

■ Optimize **P** while fixing **Q**:

The first order optimality:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{p}_u} = 0$$

$$\Rightarrow \sum_i(\mathbf{q}_i\mathbf{q}_i^\mathrm{T} + \lambda\mathbf{I})\,\mathbf{p}_u = \mathbf{Q}\mathbf{R}_{u*}^\mathrm{T}$$

$$\Rightarrow \mathbf{p}_u = (\mathbf{Q}\mathbf{Q}^\mathrm{T} + \lambda\mathbf{I})^{-1}\mathbf{Q}\mathbf{R}_{u*}^\mathrm{T}$$

■ $\mathbf{R}_{\mathbf{u}*}$ denotes the u-th row of rating matrix **R**

■ Update all $\mathbf{p}_u$ with the above formula

Optimize Q while Fixing P

- Objective function:

$$\mathcal{L} = \sum_{u,i}(r_{u,i} - \mathbf{p}_u^{\mathrm{T}}\mathbf{q}_i)^2 + \lambda(\sum_u\|\mathbf{p}_u\|^2 + \sum_i\|\mathbf{q}_i\|^2)$$

- Optimize **Q** while fixing **P**:

  The first order optimality:

  $$\frac{\partial\mathcal{L}}{\partial\mathbf{q}_i} = 0$$

  $$\Rightarrow \sum_u(\mathbf{p}_u\mathbf{p}_u^{\mathrm{T}} + \lambda\mathbf{I})\,\mathbf{q}_i = \mathbf{P}^{\mathrm{T}}\mathbf{R}_{*\mathbf{i}}$$

  $$\Rightarrow \mathbf{q}_i = (\mathbf{P}^{\mathrm{T}}\mathbf{P} + \lambda\mathbf{I})^{-1}\mathbf{P}^{\mathrm{T}}\mathbf{R}_{*\mathbf{i}}$$

- $\mathbf{R}_{*\mathbf{i}}$ denotes the i-th column of rating matrix **R**

- Update all $\mathbf{q}_i$ with the above formula

ALS for MF

Algorithm 2 ALS Algorithm

1: **Require** rating matrix **R**, feature matrices **P**, **Q** and regularization parameter λ

2: **Optimize P** while fixing **Q**:

$$\mathbf{p}_u = (\mathbf{QQ}^{\mathrm{T}} + \lambda\mathbf{I})^{-1}\mathbf{QR}_{u*}^{\mathrm{T}}$$

3: **Optimize Q** while fixing **P**:

$$\mathbf{q}_i = (\mathbf{P}^{\mathrm{T}}\mathbf{P} + \lambda\mathbf{I})^{-1}\mathbf{P}^{\mathrm{T}}\mathbf{R}_{*\mathbf{i}}$$

4: **Repeat** the above processes until **convergence**

Why SGD?

- **Time complexity per iteration** of ALS:

$$O(|\Omega|k^2 + (m+n)k^3)$$

  - $|\Omega|$ denotes the number of observed samples

  - k denotes the rank

  - m and n denote the number of users and items

  - ALS is not scalable to large-scale datasets

- **Time complexity per iteration** of SGD:

$$O(|\Omega|k)$$

  - SGD is scalable to large-scale datasets

- SGD is to minimize the following objective function:

$$\mathcal{L} = \sum_{u,i \in \Omega} (r_{u,i} - \mathbf{p}_u^{\mathrm{T}} \mathbf{q}_i)^2 + \lambda_p \|\mathbf{p}_u\|^2 + \lambda_q \|\mathbf{q}_i\|^2$$

- $\mathbf{P} = [\mathbf{p}_1, \mathbf{p}_2 \cdots, \mathbf{p}_m]^{\mathrm{T}} \in \mathbb{R}^{m \times k}$

- $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2 \cdots, \mathbf{q}_n] \in \mathbb{R}^{k \times n}$

- $r_{u,i}$ denotes the **actual rating** of user u for item i

- Ω denotes the set of **observed samples** from rating matrix $\mathbf{R}$

- λ_p and λ_q are **regularization parameters** to avoid overfitting

General Steps of SGD

Algorithm 3 General Steps of SGD

1: **Require** feature matrices **P**, **Q**, observed set Ω, regularization parameters λ_p, λ_q and learning rate α

2: **Randomly** select an observed sample $r_{u,i}$ from observed set Ω

3: Calculate the **gradient** w.r.t to the objective function

4: Update feature matrices **P** and **Q** with learning rate α and gradient

5: **Repeat** the above processes until **convergence**

Gradient Computation

- Objective function:
$$\mathcal{L} = \sum_{u,i \in \Omega} (r_{u,i} - \mathbf{p}_u^{\mathrm{T}} \mathbf{q}_i)^2 + \lambda_p \|\mathbf{p}_u\|^2 + \lambda_q \|\mathbf{q}_i\|^2$$

- **Randomly** select an observed sample $r_{u,i}$

- Calculate the **prediction error**:
$$E_{u,i} = r_{u,i} - \mathbf{p}_u^{\mathrm{T}} \mathbf{q}_i$$

- Calculate the **gradient**:
$$\frac{\partial \mathcal{L}}{\partial \mathbf{p}_u} = 2(E_{u,i}(-\mathbf{q}_i) + \lambda_p \mathbf{p}_u)$$
$$\frac{\partial \mathcal{L}}{\partial \mathbf{q}_i} = 2(E_{u,i}(-\mathbf{p}_u) + \lambda_q \mathbf{q}_i)$$

Update Feature Matrices

- Objective function:

$$\mathcal{L} = \sum_{u,i \in \Omega} (r_{u,i} - \mathbf{p}_u^{\mathrm{T}} \mathbf{q}_i)^2 + \lambda_P \|\mathbf{p}_u\|^2 + \lambda_q \|\mathbf{q}_i\|^2$$

- **Randomly** select an observed sample $r_{u,i}$

- Calculate the **prediction error**:

$$E_{u,i} = r_{u,i} - \mathbf{p}_u^{\mathrm{T}} \mathbf{q}_i$$

- Calculate the **gradient**:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{p}_u} = 2(E_{u,i}(-\mathbf{q}_i) + \lambda_p \mathbf{p}_u)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{q}_i} = 2(E_{u,i}(-\mathbf{p}_{\mathrm{u}}) + \lambda_q \mathbf{q}_i)$$

- Update the feature matrices **P** and **Q** with **learning rate** α:

$$\mathbf{p}_u = \mathbf{p}_u + 2\alpha(E_{u,i} \mathbf{q}_i - \lambda_p \mathbf{p}_u)$$

$$\mathbf{q}_i = \mathbf{q}_i + 2\alpha(E_{u,i} \mathbf{p}_u - \lambda_q \mathbf{q}_i)$$

Algorithm 4 SGD Algorithm

1: **Require** feature matrices $\mathbf{P}$, $\mathbf{Q}$, observed set Ω, regularization parameters λ_p, λ_q and learning rate α

2: **Randomly** select an observed sample $r_{u,i}$ from observed set Ω

3: Calculate the **gradient** w.r.t to the objective function:

$$E_{u,i} = r_{u,i} - \mathbf{p}_u^{\mathrm{T}}\mathbf{q}_i$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{p}_u} = 2(E_{u,i}(-\mathbf{q}_i) + \lambda_p \mathbf{p}_u)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{q}_i} = 2(E_{u,i}(-\mathbf{p}_u) + \lambda_q \mathbf{q}_i)$$

4: **Update** feature matrices $\mathbf{P}$ and $\mathbf{Q}$ with learning rate α and gradient

$$\mathbf{p}_u = \mathbf{p}_u + 2\alpha\big(E_{u,i}\mathbf{q}_i - \lambda_p \mathbf{p}_u\big)$$

$$\mathbf{q}_i = \mathbf{q}_i + 2\alpha(E_{u,i}\mathbf{p}_u - \lambda_q \mathbf{q}_i)$$

5: **Repeat** the above processes until **convergence**

Matrix Factorization

Comparison between SGD and ALS

- ALS is easier to **parallelize** than SGD

- ALS **converges faster** than the SGD

- SGD has less **storage complexity** than ALS

  (ALS needs to store the rating matrix **R**)

- SGD has less **computational complexity** than ALS

  (ALS needs to compute the matrix-vector multiplication)

Accuracy Measures

- Mean Absolute Error (MAE) computes the deviation between predicted ratings and actual ratings

$$MAE = \frac{1}{|\Omega|} \sum_{u,i \in \Omega} \left| \hat{r}_{u,i} - r_{u,i} \right|$$

  - Ω denotes the **observed set**

  - $|\Omega|$ denotes the **number of observed set**

  - $r_{u,i}$ denotes the **actual rating** and $\hat{r}_{u,i}$ denotes the prediction

- **Root Mean Square Error (RMSE)** is similar to MAE, but places more emphasis on larger deviation

$$RMSE = \sqrt{\sum_{u,i \in \Omega} (\hat{r}_{u,i} - r_{u,i})^2 / |\Omega|}$$

Riemannian Pursuit for Big Matrix Recovery

Mingkui Tan [1]   TANMINGKUI@GMAIL.COM
Ivor W. Tsang [2]   IVOR.TSANG@GMAIL.COM
Li Wang [3]   LIW022@UCSD.EDU
Bart Vandereycken [4]   BARTV@PRINCETON.EDU
Sinno Jialin Pan [5]   JSPAN@I2R.A-STAR.EDU.SG

[1] School of Computer Science, The University of Adelaide, Ingkarni Wardli North Terrace Campus 5005, Australia
[2] Center for Quantum Computation & Intelligent Systems, University of Technology Sydney, Australia
[3] Department of Mathematics, University of California, San Diego, USA
[4] Department of Mathematics, Princeton University, Fine Hall, Washington Road, Princeton NJ 08544-1000, USA
[5] Institute for Infocomm Research, 1 Fusionopolis Way, #21-01 Connexis (South) 138632, Singapore

Abstract

Low rank matrix recovery is a fundamental task in many real-world applications. The performance of existing methods, however, deteriorates significantly when applied to ill-conditioned or large-scale matrices. In this paper, we therefore propose an efficient method, called Riemannian Pursuit (RP), that aims to address these two problems simultaneously. Our method consists of a sequence of fixed-rank optimization problems. Each subproblem, solved by a nonlinear Riemannian conjugate gradient method, aims to correct the solution in the most important subspace of increasing size. Theoretically, RP converges linearly under mild conditions and experimental results show that it substantially outperforms existing methods when applied to large-scale and ill-conditioned matrices.

Definition 1. *Given a linear operator $\mathcal{A} : \mathbb{R}^{m \times n} \to \mathbb{R}^l$, let $\mathbf{b} = \mathcal{A}(\hat{\mathbf{X}}) + \mathbf{e}$ be l linear measurements of an unknown rank-$\hat{r}$ matrix $\hat{\mathbf{X}} \in \mathbb{R}^{m \times n}$, where $\mathbf{e}$ denotes noise. Then the task of MR is to recover $\hat{\mathbf{X}}$ by solving*

$$\min_{\mathbf{X}} f(\mathbf{X}), \quad \text{s.t.} \quad \text{rank}(\mathbf{X}) \leq r, \qquad (1)$$

where $l \ll mn$, $r \geq \hat{r}$, and $f(\mathbf{X}) = \frac{1}{2}\|\mathbf{b} - \mathcal{A}(\mathbf{X})\|_2^2$.

The definition of $\mathcal{A}$ depends on the application context, such as matrix completion, quantum state tomography, matrix factorizations; see, e.g., (Recht et al., 2010; Candès & Plan, 2010a; Recht, 2011; Keshavan et al., 2010b; Laue, 2012). Although our derivation is valid for any $\mathcal{A}$ that allows for MR, in the numerical experiments, we will focus on *matrix completion* (MC) as a specific application. In this case, $\mathcal{A}(\mathbf{X})$ is defined as the element-wise restriction of $\mathbf{X}$ on Ξ, a subset of the complete set of entries of $\mathbf{X}$.

Problem (1) is known to be NP-hard. To address it, many researchers proposed to solve the nuclear-norm convex re-

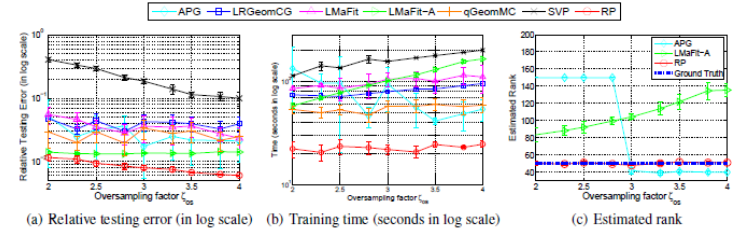*Figure 2.* Comparison on medium-sized problems of rank $\hat{r} = 50$. The results are obtained by averaging over 10 independent trials.

| (a) Relative testing error (in log scale) | (b) Training time (seconds in log scale) | (c) Estimated rank |

APG, LRGeomCG, qGeomMC, Lmafit, Lmafit-A, GECO, Jaggi's method and Laue's method.

In general, collaborative filtering data are very noisy. As a result, the singular values of the matrices tend to be long-tail. Therefore, we need to set larger stopping tolerances to alleviate over-fitting. For this experiment, we set $\lambda_F = 0.2$ and $\epsilon_{out} = 10^{-4}$. We set $\eta = 0.65$, and the detected ranks by RP are used as the rank estimations for the fixed-rank methods, namely LRGeomCG, qGeomMC and LMaFIT. Finally, we constrain the maximum rank for all methods to 100. For comparison, we report the testing RMSE of different methods over 10 random 80/20 train/test partitions as explained in (Laue, 2012).

Comparison results are shown in Table 3. From the table, we can observe that RP performs the best among all the methods in terms of RMSE and computational efficiency. It is worth mentioning that we use the rank detected by RP as the rank estimation for LRGeomCG and qGeomMC. Therefore, RP can be much faster than these two methods if the cost of the model selection is considered.

atures are similar to ours, thus the comparison is fair. In addition, we list the training time, the times of speedup, RMSE and the CPU details in Table 4 for reference.

Table 4. Performance comparison on Movie-10M dataset.

Methods	Time (in seconds)	SpeedUp	RMSE	CPU(GHz)
GECO	784,941	9,000x	0.821	2.5
Laue	2,663	30x	0.815	2.5
Jaggi	3,120	38x	0.862	2.4
RP	81	–	0.817	2.8

From Table 4, we observe that on the Movie-10M dataset, RP obtains comparable or better performance to the baseline methods in terms of RMSE, but can achieve great speedup with similar CPUs. Particularly, RP is orders of magnitude faster than all the other methods. With these comparisons, we can conclude that RP can achieve much faster training speed over the comparison methods.

6. Conclusion

We propose a Riemannian Pursuit (RP) method for tackling big MR problems. In contrast to nuclear-norm based methods, RP only needs to compute rank-ρ truncated SVD with ρ very small per iteration, as opposed to APG which may take hundreds of high-dimensional SVDs. By exploiting the Riemannian geometry of the fixed-rank manifold, RP uses a more efficient master solver. Moreover, RP increases the rank of the matrix by $\rho > 1$ per iteration, thus it exhibits good scalability for big MR problems with large ranks. Finally, RP automatically detects the rank with appropriate stopping conditions, and performs well on ill-conditioned problems. Extensive experimental results show that RP achieves superb scalability and maintain similar or better MR performance compared with state-of-the-art methods.

Table 3. Experimental results on real-world datasets.

Dataset	Movie-10M		Netflix	
	RMSE	Time (seconds)	RMSE	Time (seconds)
APG	1.096	1048±17	-	-
LRGeomCG	0.824	338±11	0.867	3128±35
QgeomMC	0.850	189±7	0.880	3965±74
Lmafit	0.837	307±1	0.875	3798±50
Lmafit-A	0.969	421±16	0.962	5286±165
RP	**0.817**	**81±1**	**0.859**	**1332±27**

* Result of APG on Netflix is absent due to out-of-memory issue. The standard variations of RMSE are not reported since they are not significant. The average ranks estimated by APG, Lmafit-A and RP on Movie are 100, 77 and 10, respectively. The average ranks estimated by Lmafit-A and RP on Netflix are 81 and 12, respectively.

Code available at: https://tanmingkui.github.io/files/sourcecode/Riemannian.rar

Riemannian Pursuit for Big Matrix Recovery

Mingkui Tan's Homepage

Home   Publications   Projects   CV   MPL   RP   CGM   Collaborators & Friends   Gallery

Blog   SBCW

Riemannian Pursuit For Big Matrix Recovery

Low rank matrix recovery is a fundamental task in many real-world applications. The performance of existing methods, however, deteriorates significantly when applied to ill-conditioned and large-scale matrices. In this paper, we therefore propose an efficient method, called Riemannian Pursuit (RP), that aims to address these two problems simultaneously. Our method consists of a sequence of fixed-rank optimization problems with increasing rank. Each subproblem, solved by a nonlinear Riemannian conjugate gradient method, aims to correct the solution in the most important subspace of a certain size. Theoretically, RP converges linearly under mild conditions and experimental results show that it substantially outperforms existing methods when applied to large-scale and ill-conditioned matrices.

The source codes are available HERE

Please cite the following paper if you find the codes are useful:

Mingkui Tan, Ivor W. Tsang, Li Wang, Bart Vandereycken, Sinno Jialin Pan. "Riemannian Pursuit for Big Matrix Recovery", Proceedings of the 31st International Conference on Machine Learning (ICML), 2014

Code available at: https://tanmingkui.github.io/files/sourcecode/Riemannian.rar

Thank You