

SVD / MF RS I

STAT3009 Recommender Systems

by Ben Dai (CUHK-STAT)

on October 9, 2024

» Recall: RS

- * Training dataset: [userID, itemID, rating]
- * Testing dataset: [userID, itemID, ?]
- * Evaluation: Given a testing index set Ω^{te} (set of user-item pairs we want to predict),

$$RMSE = \left(\frac{1}{|\Omega^{\text{te}}|} \sum_{(u,i) \in \Omega^{\text{te}}} (\hat{r}_{ui} - r_{ui})^2 \right)^{1/2}.$$

- * Goal: Find predicted ratings $(\hat{r}_{ui})_{(u,i) \in \Omega^{\text{te}}}$ such that **minimizes RMSE**
- * Baseline methods: Global-average, user-average, item-average, user-item average

» Machine learning (ML): RS

Using ML methods to build RS:

- Step 1. Introduce a model with some **parameters** and **hyperparameters**
- Step 2. Estimate the **parameters** by minimizing (maximizing) the **Evaluation Loss** in **Training Set**
- Step 3. Using **Cross-Validation** to determine the optimal **hps**
- Step 4. Refit the best model, and make **prediction**

» Components in ML

- Data (feat, label) is a pair of **input features** and its **outcome**
- Model f_{θ} : a **parameterized** function to map features to label
- Loss $L(\cdot, \cdot)$: The measure of how good the **predicted** outcome compared with the **true** outcome
 - hp **hyperparameter** to control the complexity of the model to prevent **overfitting**
- Opt The **algorithm** for solving the problem

» Rethink baseline methods: Opt

Step 1. Introduce a method with some **params**

method	MATH	parameters
Global pred	$\hat{r}_{ui} = \mu_0$	μ_0
User pred	$\hat{r}_{ui} = a_u$	$\mathbf{a} = (a_1, \dots, a_n)^\top$
Item pred	$\hat{r}_{ui} = b_i$	$\mathbf{b} = (b_1, \dots, b_m)^\top$

Step 2. Estimate the **parameters** by minimizing **RMSE**

Global $\hat{f}_\theta(u, i) = \bar{r}$

User $\hat{f}_\theta(u, i) = \bar{r}_u$

Item $\hat{f}_\theta(u, i) = \bar{r}_i$

Step 3. CV to find the best model

Step 4. Refit the best model on the whole dataset, and make prediction

InClass demo: Recall Kaggle Quiz 1

» Discussion: Baseline Methods

“All models are wrong, but some are useful.” — George E. P. Box

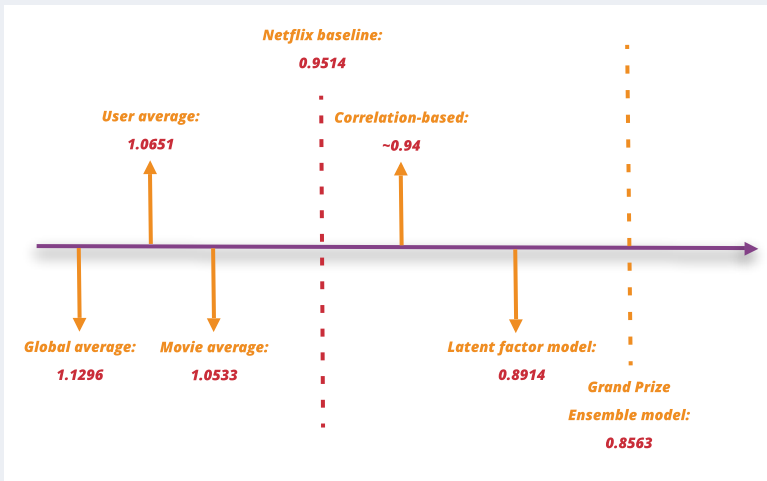
To evaluate each method, we need to understand the underlying **assumptions**.

- * **Global average** assumes that all users and items are homogeneous.
- * **User average** assumes that a user has uniform preference for all items.
- * **Item average** assumes that all users prefer “good” items.
- * **User-item average** assumes additive effects from users and items, with **no interaction**.

Example: I dislike action movies, regardless of their high ratings.

To improve upon these methods, we need to model the user-item **interaction**.

» Motivation: SVD Model



» Motivation: SVD Model

A new Python sklearn-type Estimator for RS...

Step 1. Introduce a method with **parameters** (latent factors):

- * Associate each user u with a K -length latent factor vector

$$\mathbf{p}_u = (p_{u1}, \dots, p_{uK})^\top.$$

- * Associate each item i with a K -length latent factor vector

$$\mathbf{q}_i = (q_{i1}, \dots, q_{iK})^\top.$$

- * Model the user-item **interaction** as the **inner product** of these vectors:

$$\hat{r}_{ui} = \mathbf{p}_u^\top \mathbf{q}_i \rightarrow r_{ui}.$$

- * The number of latent factors, K , is a pre-specified **hyperparameter**.

» Loss: SVD Model

Step 2. Estimate the **parameters** by minimizing **RMSE**

$$\min_{\mathbf{P} \in \mathbb{R}^{n \times K}, \mathbf{Q} \in \mathbb{R}^{m \times K}} \frac{1}{|\Omega|} \sum_{(u,i) \in \Omega} (r_{ui} - \mathbf{p}_u^T \mathbf{q}_i)^2 \quad (1)$$

Question: What are the **parameters** and **hyperparameters** of this model?

» SVD Model

Param $\mathbf{p}_u (u = 1, \dots, n)$ and $\mathbf{q}_i (i = 1, \dots, m)$ are the parameters we want to learn from data.

hp K is a pre-specified #Latent Factor, can **NOT** be solved from data.

* K increases \implies more **parameters** \implies lower **training loss**

How many params?

» Overfitting in ML: SVD Model

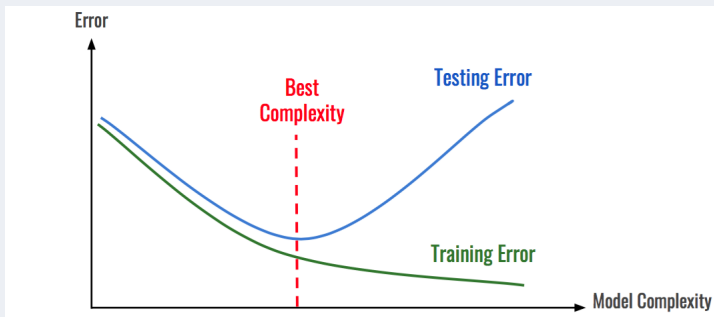


Source¹

- * **Overfitting**: fit the noise
- * Too many **parameters** (**model complexity**) leads to **overfitting**

¹<https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42>

» Overfitting in ML: SVD Model



Source²

* **Complexity** too large \Rightarrow Low Training loss but high Testing loss

²<https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42>

» Overfitting in ML: SVD Model

	Low Training Error	High Training Error
Low Testing Error	The model is learning!	Probably some error in your code. Or you've created a <i>psychic</i> AI.
High Testing Error	OVERFITTING	The model is not learning.

Source³

³<https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42>

» Tuning: SVD Model

- * **Q1:** How to **quantify** the Model **Complexity**:

- * **#Parameters:** $(n + m)K$

- * **Magnitude** of Parameters: $\sum_{u=1}^n \|\mathbf{p}_u\|_2^2, \sum_{i=1}^m \|\mathbf{q}_i\|_2^2$

- * **A1:** Control (#Parameters by K , **Magnitude** by l_2 -norm).

- * **Regularized** SVD Model:

$$\min_{\mathbf{P}, \mathbf{Q}} \underbrace{\frac{1}{|\Omega|} \sum_{(u,i) \in \Omega} (r_{ui} - \mathbf{p}_u^\top \mathbf{q}_i)^2}_{\text{Training loss}} + \lambda \underbrace{\left(\sum_{u=1}^n \|\mathbf{p}_u\|_2^2 + \sum_{i=1}^m \|\mathbf{q}_i\|_2^2 \right)}_{\text{Params magnitude}} \quad (2)$$

where K and $\lambda > 0$ are **tuning** parameters to **balance** the model complexity and training loss.

- * Why the later term can control the magnitude?

InClass demo: Implement `Estimator.__init__`

» Tuning: SVD Model

Step 3. Using **GridSearch** + **CV** to find the optimal (K, λ) .

- * (holdout or K-Fold CV)

Step 4. Refit the model with the optimal (K, λ) and make prediction.

» Summary: SVD Model

- Step 1. Introduce a method with some **params** + **hps**
- * Model the user-item **interaction** as **inner production**

$$\hat{r}_{ui} = \mathbf{p}_u^\top \mathbf{q}_i \rightarrow r_{ui}$$

- Step 2. Estimate the **parameters** by minimizing **RMSE**

$$\min_{\mathbf{P}, \mathbf{Q}} \frac{1}{|\Omega|} \sum_{(u,i) \in \Omega} (r_{ui} - \mathbf{p}_u^\top \mathbf{q}_i)^2 + \lambda \left(\sum_{u=1}^n \|\mathbf{p}_u\|_2^2 + \sum_{i=1}^m \|\mathbf{q}_i\|_2^2 \right) \quad (3)$$

- Step 3. Using **Cross-Validation** to determine the optimal **tuning parameters** (K, λ), denote as K^* and λ^*
- Step 4. **Refit** the model based on **full training data** with K^* and λ^* and make prediction.

» Big Picture: MF RS

Algorithm 1 Fitting+Tuning+Prediction MF

- 1: **Input:** **Training** set $(u, i, r_{ui})_{(u,i) \in \Omega}$
- 2: **Return:** Predicted ratings for **Testing** set: $(u, i) \in \Omega^{\text{te}}$
- 3: **for** $(K, \lambda) \in \text{Grid Set}$ **do**
- 4: (**Tuning**: compute CV score)
- 5: Estimate the model with (K, λ) by **solving** (3)
- 6: Compute **CV Score**
- 7: **end for**
- 8: Find the **best** hps (K^*, λ^*) with smallest **RMSE** on *valid* set
- 9: (**Refitting**) Estimate the **best tuned model** by **solving** (3)
- 10: (**Predict**) test ratings by the estimated **best** tuned model

Question: What's the Python workflow?

» Python Estimator

SVD(BaseEstimator)

- * `__init__`
- * `fit(X, y)`: Solving optimization problem in (3)
- * `predict(X)`

Then, GridSearch + CV can automatically implemented by GridSearchCV

Thus, the **key** is to implement the **fit** method to solve (3)?

InClass demo: Implement predict method.

» Optimization I: Matrix Factorization (Optional)

Recall the **regularized** Matrix Factorization (MF) problem:

$$\min_{\mathbf{P}, \mathbf{Q}} \frac{1}{|\Omega|} \sum_{(u,i) \in \Omega} (r_{ui} - \mathbf{p}_u^\top \mathbf{q}_i)^2 + \lambda \left(\sum_{u=1}^n \|\mathbf{p}_u\|_2^2 + \sum_{i=1}^m \|\mathbf{q}_i\|_2^2 \right)$$

We make the following key **observations**:

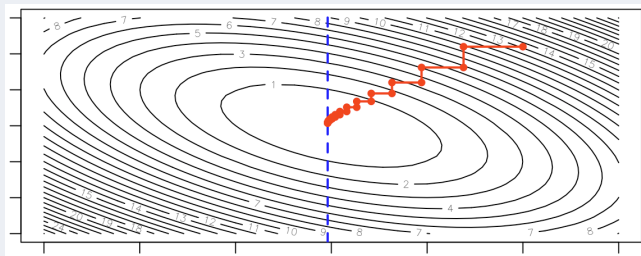
Obs 1 The optimization problem is **nonconvex** due to the **bilinear** term $\mathbf{p}_u^\top \mathbf{q}_i$.

Obs 2 However, when either \mathbf{P} or \mathbf{Q} is fixed, the problem becomes **convex** and can be solved as a standard Quadratic Program (QP), which is essentially a **ridge regression** problem.

These observations **motivate** us to consider using **coordinate descent** to solve this problem.

» Optimization I: Matrix Factorization (Optional)

CD Coordinate Descent



Idea At the $(l+1)$ th iteration, minimize the objective w.r.t. one coordinate, while keeping all others fixed:

$$\theta_j^{(l+1)} = \operatorname{argmin}_x \operatorname{Obj}(\theta_1^{(l+1)}, \dots, \theta_{j-1}^{(l+1)}, \mathbf{x}, \theta_{j+1}^{(l+1)}, \dots, \theta_{|\theta|}^{(l)})$$

- * Repeat until a **termination condition** is met.
- * This approach is useful when the **joint** optimization problem is difficult to solve, but the **sub-problems** (minimizing w.r.t. one coordinate) are easy to solve.

» Optimization I: Matrix Factorization (Optional)

BCD Blockwise Coordinate Descent

Idea At the $(l+1)$ th iteration, minimize the objective function with respect to a block of coordinates:

$\theta_j^{(l+1)} = \operatorname{argmin}_{\mathbf{x}} \operatorname{Obj}(\theta_1^{(l+1)}, \dots, \theta_{j-1}^{(l+1)}, \mathbf{x}, \theta_{j+1}^{(l+1)}, \dots, \theta_{|\theta|}^{(l)})$,
where each θ_j is a *vector*.

- * This approach is useful when the **joint** optimization problem is difficult to solve, but the **sub-problems** (minimizing with respect to a block of coordinates) are easy to solve.

Blockwise Coordinate Descent perfectly fits with our Matrix Factorization formulation...

» Optimization II: Matrix Factorization (Optional)

Let's take a closer look...

Update **Q** When $(\mathbf{p}_u)_{u=1}^n$ are fixed, (3) is a quadratic program (QP) with respect to $(\mathbf{q}_i)_{i=1, \dots, m}$

$$\begin{aligned} & \min_{\mathbf{Q}} \frac{1}{|\Omega|} \sum_{(u,i) \in \Omega} (r_{ui} - \mathbf{p}_u^\top \mathbf{q}_i)^2 + \lambda \left(\sum_{u=1}^n \|\mathbf{p}_u\|_2^2 + \sum_{i=1}^m \|\mathbf{q}_i\|_2^2 \right) \\ \iff & \min_{\mathbf{Q}} \frac{1}{|\Omega|} \sum_{(u,i) \in \Omega} (r_{ui} - \mathbf{p}_u^\top \mathbf{q}_i)^2 + \lambda \sum_{i=1}^m \|\mathbf{q}_i\|_2^2 \\ \iff & \min_{\mathbf{Q}} \sum_{i=1}^m \left(\frac{1}{|\Omega|} \sum_{u \in \mathcal{U}_i} (r_{ui} - \mathbf{p}_u^\top \mathbf{q}_i)^2 + \lambda \|\mathbf{q}_i\|_2^2 \right). \end{aligned} \quad (4)$$

- * Note that the objective function in (4) is *separable* with respect to \mathbf{q}_i for $i = 1, \dots, m$.

» Optimization III: Matrix Factorization (Optional)

Decompose Thus, solving the objective function in (4) is equivalent to separately solving m **small quadratic programs (QPs)**:

$$\begin{aligned} & \min_{\mathbf{Q}} \sum_{i=1}^m \left(\frac{1}{|\Omega|} \sum_{u \in \mathcal{U}_i} (r_{ui} - \mathbf{p}_u^\top \mathbf{q}_i)^2 + \lambda \|\mathbf{q}_i\|_2^2 \right) \\ \iff & \min_{\mathbf{q}_i} \frac{1}{|\Omega|} \sum_{u \in \mathcal{U}_i} (r_{ui} - \mathbf{q}_i^\top \mathbf{p}_u)^2 + \lambda \|\mathbf{q}_i\|_2^2, \text{ for } i = 1, \dots, m \end{aligned}$$

» Optimization III: Matrix Factorization (Optional)

Decompose Thus, solving the objective function in (4) is equivalent to separately solving m **small quadratic programs (QPs)**:

$$\min_{\mathbf{Q}} \sum_{i=1}^m \left(\frac{1}{|\Omega|} \sum_{u \in \mathcal{U}_i} (r_{ui} - \mathbf{p}_u^\top \mathbf{q}_i)^2 + \lambda \|\mathbf{q}_i\|_2^2 \right)$$

$$\iff \min_{\mathbf{q}_i} \frac{1}{|\Omega|} \sum_{u \in \mathcal{U}_i} (r_{ui} - \mathbf{q}_i^\top \mathbf{p}_u)^2 + \lambda \|\mathbf{q}_i\|_2^2, \text{ for } i = 1, \dots, m$$

Interestingly, each **sub-QP** is essentially a *Ridge Regression* problem:

$$\min_{\mathbf{q}_i} \frac{1}{|\Omega|} \sum_{u \in \mathcal{U}_i} (r_{ui} - \underbrace{\mathbf{q}_i^\top \mathbf{p}_u}_{\beta^\top \mathbf{x}_i \text{ in Linear Regression}})^2 + \lambda \underbrace{\|\mathbf{q}_i\|_2^2}_{\|\beta\|_2^2}.$$

InClass demo: Solve the sub-problem by `sklearn.linear_model.Ridge` for $i = 1$.

» Optimization SUM: MF

BCD perfectly fits our model (**alternative least squares** (ALS))

Steps solve \mathbf{Q} (fixed \mathbf{P}) \rightarrow solve \mathbf{P} (fixed \mathbf{Q}) \rightarrow ...

- * When \mathbf{P} is fixed, the objective function for \mathbf{Q} is a standard QP, and each \mathbf{q}_i can be solved **parallelly** with an *analytic solution*.
- * When \mathbf{Q} is fixed, the objective function for \mathbf{P} is a standard QP, and each \mathbf{p}_i can be solved **parallelly** with an *analytic solution*.

» ALS: MF

Algorithm 2 ALS for solving MF

- 1: **Input:** *Training* set $(u, i, r_{ui})_{(u,i) \in \Omega}$, *hps:* K, λ
- 2: **Return:** Est params: (\hat{P}, \hat{Q})
- 3: (**Initialization**) Initialize $P^{(0)}$
- 4: **for** $l = 0, \dots, \text{Max_Iter}$ **do**
- 5: (**Item-Update**)
- 6: **for** $i = 1, \dots, m$ **do**
- 7: $q_i^{(l+1)}$ updated by Ridge regression
- 8: **end for**

» ALS: MF

Algorithm 3 ALS for solving MF

```
1: Input: Training set  $(u, i, r_{ui})_{(u,i) \in \Omega}$ , hps:  $K, \lambda$ 
2: Return: Est params:  $(\hat{P}, \hat{Q})$ 
3: (Initialization) Initialize  $P^{(0)}$ 
4: for  $l = 0, \dots, \text{Max\_Iter}$  do
5:   (Item-Update)
6:   for  $i = 1, \dots, m$  do
7:      $q_i^{(l+1)}$  updated by Ridge regression
8:   end for
9:   (User-Update)
10:  for  $u = 1, \dots, n$  do
11:     $p_u^{(l+1)}$  updated by Ridge regression
12:  end for
13:  Break the loop if termination condition.
14: end for
15: Return  $(P^{(l+1)}, Q^{l+1})$ 
```

» ALS: Latent Factor Model

Termination condition:

- * Diff in params:

$$\frac{1}{n} \sum_{u=1}^n \|\mathbf{p}_u^{(l+1)} - \mathbf{p}_u^{(l)}\|_2^2 + \frac{1}{m} \sum_{i=1}^m \|\mathbf{q}_i^{(l+1)} - \mathbf{q}_i^{(l)}\|_2^2 \leq \varepsilon,$$

- * Diff in objective function:

$$\text{MSE}^{(l)} + \lambda \text{Reg}^{(l)} - (\text{MSE}^{(l+1)} + \lambda \text{Reg}^{(l+1)}) \leq \varepsilon.$$

InClass demo: Implementation of Algorithm 3.

» Theory of Algorithms

- * An iterative algorithm is said to **converge** when, as the iterations proceed, the output gets closer and closer to a **specific value**.
- * **Conditions** for convergence

Lemma (Monotone Convergence Lemma)

*If a sequence of real numbers is decreasing and bounded below, then it will **converge** to its **infimum**.*

» Theory of Algorithms

- * An iterative algorithm is said to **converge** when, as the iterations proceed, the output gets closer and closer to a **specific value**.
- * **Conditions** for convergence

Lemma (Monotone Convergence Lemma)

*If a sequence of real numbers is decreasing and bounded below, then it will **converge** to its **infimum**.*

- * Most algorithms use this lemma to show convergence
- C1 The objective function is **bounded below**
 - e.g. Most objective functions are bounded below by their definition: **Root Mean Squared Error (RMSE)** + **Regularization (Reg)**
 - C2 Each step should result in a **decreasing** objective function
 - e.g. Block Coordinate Descent (BCD) and Alternating Least Squares (ALS)

» Tips for Debugging Block Coordinate Descent (BCD)

Identifying a bug in the algorithm with **multiple blocks** in one iteration

- * Handling multiple blocks in a single iteration
- * Monitor the **objective function** after each block update
- * Identify the blocks for which the **objective function** is not decreasing
- * Pinpoint the location of the bug