SVD Models I

STAT3009 Recommender Systems

by Ben Dai (CUHK)
on Department of Statistics and Data Science

» What You Should Know by Now

1. **RS data and Python & Numpy Programming**
   * Basic RS data structures
   * Array operations and vectorization
2. **Machine Learning Basics**
   * Linear regression, regularization (Ridge)
   * Cross-validation techniques, Grid search
3. **Baseline Methods** (global mean, user/item bias)
4. **scikit-learn Usage**
   * Implement a RS method using sklearn `BaseEstimator` framework
   * Automatically perform hyperparameter tuning with `GridSearchCV`
5. **Kaggle Platform Operations**

» 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),

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

* 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	$\hat{r}_{ui} = a_u$	$\boldsymbol{a} = (a_1, \cdots, a_n)^\intercal$
Item pred	$\hat{r}_{ui} = b_i$	$\boldsymbol{b} = (b_1, \cdots, b_m)^\intercal$

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

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

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

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

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

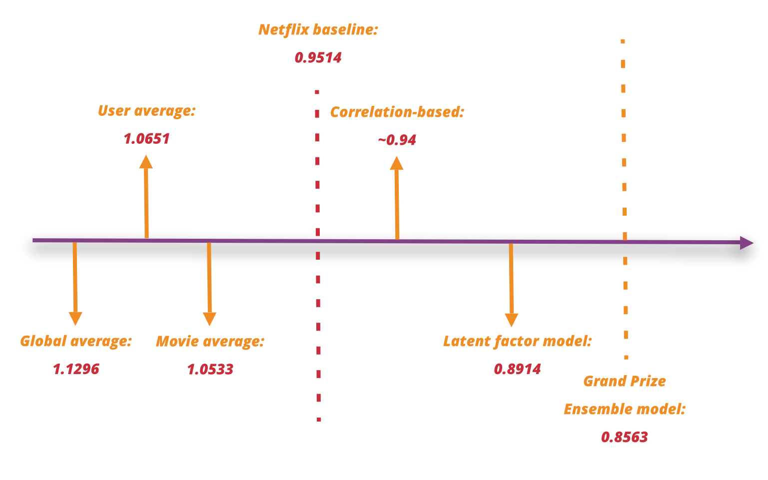
To evaluate each method, we need to understand the underlying **assumptions**.

* 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: Eric is a generous person, and this is indeed an excellent film, but he simply do not like it.

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
$$\boldsymbol{p}_u = (p_{u1}, \cdots, p_{uK})^\intercal.$$
- Associate each item i with a K-length latent factor vector
$$\boldsymbol{q}_i = (q_{i1}, \cdots, q_{iK})^\intercal.$$
- Model the user-item **interaction** as the **inner product** of these vectors:
$$\hat{r}_{ui} = \boldsymbol{p}_u^\intercal \boldsymbol{q}_i \to r_{ui}.$$

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

Intuition: Each user/item is represented by a k-dimensional vector capturing latent preferences/attributes.

Example with $k = 2$ latent factors:

User vectors $\mathbf{p}_u \in \mathbb{R}^2$:

$$\mathbf{p}_{\text{Alice}} = \begin{bmatrix} 0.9 \\ 0.2 \end{bmatrix}$$

$$\mathbf{p}_{\text{Bob}} = \begin{bmatrix} 0.3 \\ 0.8 \end{bmatrix}$$

Item vectors $\mathbf{q}_i \in \mathbb{R}^2$:

$$\mathbf{q}_{\text{Avengers}} = \begin{bmatrix} 1.0 \\ 0.1 \end{bmatrix}$$

$$\mathbf{q}_{\text{Notebook}} = \begin{bmatrix} 0.2 \\ 0.9 \end{bmatrix}$$

Predicted Ratings:

Alice watches Avengers:

$$r_{\text{Alice,Avengers}} = 0.9 \times 1.0 + 0.2 \times 0.1 = \boxed{0.92}$$

Bob watches Avengers:

$$r_{\text{Bob,Avengers}} = 0.3 \times 1.0 + 0.8 \times 0.1 = \boxed{0.38}$$

Alice watches Notebook:

$$r_{\text{Alice,Notebook}} = 0.9 \times 0.2 + 0.2 \times 0.9 = \boxed{0.36}$$

...

» Geometric Interpretation of $r_{ui} = \mathbf{p}_u^T \mathbf{q}_i$

Dot Product as Similarity:

$$\mathbf{p}_u^T \mathbf{q}_i = \|\mathbf{p}_u\| \|\mathbf{q}_i\| \cos(\theta)$$

* $\theta \approx 0$: vectors aligned $\Rightarrow$ high rating
* $\theta \approx 90$: orthogonal $\Rightarrow$ neutral rating
* Large $\|\mathbf{p}_u\|$: user with strong preferences
* Large $\|\mathbf{q}_i\|$: item with distinct features

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

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

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

» SVD Model

Param $\boldsymbol{p}_u (u = 1, \cdots, n)$ and $\boldsymbol{q}_i (i = 1, \cdots, m)$ 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 $\Longrightarrow$ more **parameters** $\Longrightarrow$ lower **training loss**

How many params?

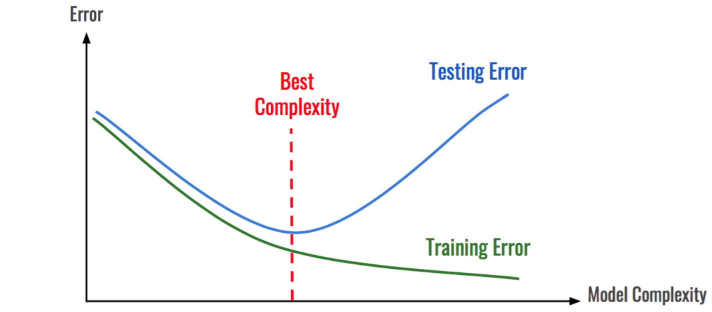
» Overfitting in ML: SVD Model

Source[1]

- **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



Source[2]

* **Complexity** too large $\implies$ 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 *psychic* AI.
High *Testing* Error	OVERFITTING	The model is not learning.

Source[3]

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: $\sum_{u=1}^{n} \|\boldsymbol{p}_u\|_2^2, \sum_{i=1}^{m} \|\boldsymbol{q}_i\|_2^2$
- **A1:** Control (#Parameters by K, **Magnitude** by l_2-norm).
- **Regularized** SVD Model:

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

  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__` and a method `obj` to compute the objective function in (2).

» 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**

$$\hat{r}_{ui} = \boldsymbol{p}_u^\intercal \boldsymbol{q}_i \to r_{ui}$$

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

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

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

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

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

We make the following key observations:

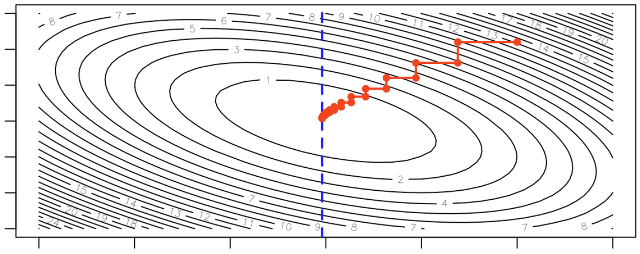
Obs 1 The optimization problem is **nonconvex** due to the **bilinear** term $\boldsymbol{p}_u^\intercal \boldsymbol{q}_i$.

Obs 2 However, when either $\boldsymbol{P}$ or $\boldsymbol{Q}$ 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:

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

- 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.

Image Source: `https://egallic.fr/Enseignement/ML/ECB/book/gradient-descent.html`

» 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:
$\theta_j^{(l+1)} = \text{argmin}_{\boldsymbol{x}} \text{ Obj}\big(\theta_1^{(l+1)}, \cdots, \theta_{j-1}^{(l+1)}, \boldsymbol{x}, \theta_{j+1}^{(l+1)}, \cdots, \theta_{|\theta|}^{(l)}\big)$,
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...

» Example: BCD for Matrix Factorization ($k = 1$) - Part 1

Setup: 2 users, 2 items, $k = 1$ latent factor, $\lambda = 0$
Observed ratings: $r_{12} = 5$, $r_{21} = 4$
Objective: $\min_{p_1, p_2, q_1, q_2} (5 - p_1 q_2)^2 + (4 - p_2 q_1)^2$

Iteration 0 (Initialize):

$$p_1 = 1, \quad p_2 = 1, \quad q_1 = 1, \quad q_2 = 1$$

Current obj: $(5 - 1)^2 + (4 - 1)^2 = 16 + 9 = \boxed{25}$

BCD Strategy: Alternate between two blocks:

* Block 1: Update all p's (user factors) while fixing all q's (item factors)
* Block 2: Update all q's (item factors) while fixing all p's (user factors)

» Example: BCD for Matrix Factorization ($k = 1$) - Part 2

Iteration 1, Step 1: Fix $q_1 = 1, q_2 = 1$, update $\mathbf{p}$

* Update p_1: $\min_{p_1}(5 - p_1 \cdot 1)^2 = (5 - p_1)^2$
  Derivative: $\frac{d}{dp_1}(5 - p_1)^2 = -2(5 - p_1) = 0 \Rightarrow \boxed{p_1 = 5}$
* Update p_2: $\min_{p_2}(4 - p_2 \cdot 1)^2 = (4 - p_2)^2$
  Derivative: $\frac{d}{dp_2}(4 - p_2)^2 = -2(4 - p_2) = 0 \Rightarrow \boxed{p_2 = 4}$

Iteration 1, Step 2: Fix $p_1 = 5, p_2 = 4$, update $\mathbf{q}$

* Update q_1: $\min_{q_1}(4 - 4 \cdot q_1)^2 = 16(1 - q_1)^2$
  Derivative: $32(1 - q_1)(-1) = 0 \Rightarrow \boxed{q_1 = 1}$
* Update q_2: $\min_{q_2}(5 - 5 \cdot q_2)^2 = 25(1 - q_2)^2$
  Derivative: $50(1 - q_2)(-1) = 0 \Rightarrow \boxed{q_2 = 1}$

After Iteration 1: $p_1 = 5, p_2 = 4, q_1 = 1, q_2 = 1$
New predictions: $\hat{r}_{12} = 5 \times 1 = 5$ ✓, $\hat{r}_{21} = 4 \times 1 = 4$ ✓
New obj: $(5 - 5)^2 + (4 - 4)^2 = \boxed{0}$ Obj is decreasing!

» Optimization II: Matrix Factorization (Optional)

Let's take a closer look...

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

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

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

» Optimization III: Matrix Factorization (Optional)

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

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

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

Decompose

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

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

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

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

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

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

```
Input: Training set (u, i, r_ui)_{(u,i)∈Ω}, hps: K, λ
Return: Est params: (P̂, Q̂)
(Initialization) Initialize P^(0)
for l = 0, ···, Max_Iter do
    (Item-Update)
    for i = 1, ···, m do
        q_i^(l+1) updated by Ridge regression
    end for
```

Input: *Training* set $(u,i,r_{ui})_{(u,i)\in\Omega}$, *hps:* K,λ

Return: Est params: $(\widehat{\boldsymbol{P}},\widehat{\boldsymbol{Q}})$

(**Initialization**) Initialize $\boldsymbol{P}^{(0)}$

for $l = 0,\cdots,Max\_Iter$ **do**

  (**Item-Update**)

  for $i = 1,\cdots,m$ **do**

    $\boldsymbol{q}_i^{(l+1)}$ updated by Ridge regression

  end for

» ALS: MF

Algorithm 3 ALS for solving MF

Input: *Training* set $(u, i, r_{ui})_{(u,i)\in\Omega}$, *hps:* K, λ
Return: Est params: $(\widehat{\boldsymbol{P}}, \widehat{\boldsymbol{Q}})$
(**Initialization**) Initialize $\boldsymbol{P}^{(0)}$
for $l = 0, \cdots, Max\_Iter$ **do**
    (**Item-Update**)
    for $i = 1, \cdots, m$ **do**
        $\boldsymbol{q}_i^{(l+1)}$ updated by Ridge regression
    end for
    (**User-Update**)
    for $u = 1, \cdots, n$ **do**
        $\boldsymbol{p}_u^{(l+1)}$ updated by Ridge regression
    end for
    Break the loop if *termination condition*.
end for
Return $(\boldsymbol{P}^{(l+1)}, \boldsymbol{Q}^{l+1})$

» ALS: Latent Factor Model

Termination condition:

* Diff in params:

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

* Diff in objective function:

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

InClass demo: Implementation of Algorithm 3.

- 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