9.20 Paper Presentation:

Matrix Factorization Methods and its Variants

Yibin Xiong

September 20, 2021

Table of Contents

Problem Setup: Recommendation System

Predict a user's preference of an item, then make recommendations

- Given transaction records, try to predict how likely a user u will buy the item i
- Given movie ratings on Netflix, try to predict what score a user u will give for a movie i
- Given retweet records (and more information) on Twitter, try to predict what tweet(s) i a user u is interested in

Problem Setup: Recommendation System

Predict a user's preference of an item, then make recommendations

- Given transaction records, try to predict how likely a user u will buy the item i
- Given movie ratings on Netflix, try to predict what score a user u will give for a movie i
- Given retweet records (and more information) on Twitter, try to predict what tweet(s) i a user u is interested in

Let r be a matrix with r_{ui} being the entry on the u-th row and i-th column. r_{ui} represents the rating that user u gives to item i.

r has many missing entries (i.e. r is sparse) whose values we need to predict.

Traditional Methods: Neighborhood Model

1. Neighborhood Model
   - Assume i) similar users have similar preferences
     ii) a user have similar preference for similar items
   - Interpolate the missing values based on some nearest neighbors

Traditional Methods: Neighborhood Model

1. Neighborhood Model
   - Assume i) similar users have similar preferences
     ii) a user have similar preference for similar items
   - Interpolate the missing values based on some nearest neighbors

Algorithm:

- Calculate the Pearson correlation coefficient ρ_{ij} between 2 items i and j based on users' rating histories
- Get the similarity measure $s_{i,j}$ by shrinking ρ_{ij}

$$s_{i,j} = \frac{n_{ij}}{n_{ij} + \lambda}\rho_{ij}$$

- Estimate the missing rating from user u to item i

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in S^k_{(i,u)}} s_{ij}(r_{uj} - b_{uj})}{\sum_{j \in S^k_{(i,u)}} s_{ij}}$$

Baseline Estimates

A baseline estimate of r_{ui}, denoted as $b_{u,i}$ is defined by

$$b_{ui} = \mu + b_u + b_i$$

where b_u, b_i are the *observed deviation* of the user and the item from the average level μ

Baseline Estimates

A baseline estimate of r_{ui}, denoted as $b_{u,i}$ is defined by

$$b_{ui} = \mu + b_u + b_i$$

where b_u, b_i are the *observed deviation* of the user and the item from the average level μ

To estimate b_u and b_i, we minimize the regularized sum square error

$$\min_{b^*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 (\sum_u b_u^2 + \sum_i b_i^2)$$

regularize to avoid overfitting caused by sparse data

Traditional Methods: LFM

2. Latent Factor Model

- Latent Feedback: as opposed to *explicit feedback* (i.e. ratings)
  E.g. previous history of ratings, which movies (out of all) the user has rated
- Latent Factor: some factors from latent feedback that influence the rating

Algorithm:

- Associate user u with a latent factor vector $p_u \in \mathbb{R}^d$ and item i with $q_i \in \mathbb{R}^d$
- Predict the rating by

$$\hat{r}_{ui} = b_{ui} + p_u^T q_i$$

This algorithm is also called SVD. Based on this, we define the SVD++ [3]

SVD

1. Most basic version
   - Given rating matrix r, we fill **0** or **average values** to the missing entries.
   - Then we do singular value decomposition (SVD) to r and get $r = P\Sigma Q^T$, where Σ is a diagonal matrix consisting of singular values σ.
   - We pick the top k singular values and the corresponding columns/rows in P and Q^T to reconstruct the rating matrix. The matrix $\hat{r}$ gives us the predicted ratings.
   - However, crude interpolation and inaccurate result!

2. A better version
   - Let $\tilde{r}$ be a **dense** matrix that consists of all rating records we have, without any missing values.
   - We do SVD on $\tilde{r}$ and get $\tilde{r} = \tilde{P}\tilde{\Sigma}\tilde{Q}^T$. Pick the top k columns/rows of $\tilde{P}$ and $\tilde{Q}^T$ that represents crucial factors related to the users and items.
   - When predicting a new user u_{new}, we construct a vector $p_{new} \in \mathbb{R}^k$ [1] and do $p_{new}\tilde{\Sigma}_k\tilde{Q}_k^T$ to get the predicted ratings.

[1] 2 ways: i) pseudo-inverse $p_{new} = r_{new}^T\tilde{Q}_k\tilde{\Sigma}_k^{-1}$
ii) least square regression

3. Funk SVD: Factor $\tilde{r}$ into 2 matrices $\tilde{r} = PQ^T$

Based on this idea, we just find the user vector and item vector(by min l2 regression error) that captures crucial latent factors in low dimensional spaces

$$\min_{p^*,q^*,b^*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$$

Idea: Combine the neighborhood model and latent factor model

- Improved neighborhood model

$$\hat{r}_{ui} = \mu + b_u + b_i + |R^k(i;u)|^{-1/2} \sum_{j \in R^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |N^k(i;u)|^{-1/2} \sum_{j \in N^k(i;u)} c_{ij}$$

$$\min_{b_*, w_*, c_*} \sum_{(u,i) \in \mathcal{K}} \left(r_{ui} - \mu - b_u - b_i - |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij} - |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} \right)^2 + \lambda_4 \left(b_u^2 + b_i^2 + \sum_{j \in \mathrm{R}^k(i;u)} w_{ij}^2 + \sum_{j \in \mathrm{N}^k(i;u)} c_{ij}^2 \right) \tag{11}$$

SVD++

Introduce the idea of latent factors q_i^T for items

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(|R(u)|^{-1/2} \sum_{j \in R(u)} (r_{uj} - b_{uj}) x_j + |N(u)|^{-1/2} \sum_{j \in N(u)} y_j \right)$$

Each item i is associated with 3 factor vectors q_i,x_i, and y_i

$$\begin{aligned} \min_{q_*, x_*, y_*, b_*} \sum_{(u,i) \in \mathcal{K}} \Bigg(& r_{ui} - \mu - b_u - b_i \\ & - q_i^T \left(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right) \Bigg)^2 \\ & + \lambda_5 \left(b_u^2 + b_i^2 + \|q_i\|^2 + \sum_{j \in \mathrm{R}(u)} \|x_j\|^2 + \sum_{j \in \mathrm{N}(u)} \|y_j\|^2 \right) \end{aligned} \quad (14)$$

Use gradient descent to update the parameters

CF with Short Term Preferences [4]

Long-term model(traditional):

$$\widehat{r}_{ui} = b_u + b_i + \left(p_u + \frac{\sum_{j \in N(u)} y_j}{\sqrt{|N(u)|}}\right)^T q_i \qquad (1)$$

Short-term model(new):

$$\widehat{r}_{ui}(t) = b_u + b_i + \left(p_u + \frac{\sum_{j \in N(u)} y_j}{\sqrt{|N(u)|}} + \frac{\sum_{j \in N(u,t)} \xi_j}{\sqrt{|N(u,t)|}}\right)^T q_i \qquad (2)$$

y_j, ξ_j are associated with item j

Collaborative Ranking: Recommend Top k Useful Tweets

- Start from collaborative filtering. We use a simple SVD algorithm to estimate the score/rating

$$\hat{y}_{ui} = \mu + b_u + b_i + p_u^T q_i$$

- Given user u and 2 items k and h, we model the pairwise ranking probability as

$$P(r(k) > r(h)|u) = sigmoid(\hat{y}_{uk} - \hat{y}_{uh}) = \frac{1}{1 + e^{-(\hat{y}_{uk} - \hat{y}_{uh})}}$$

Collaborative Ranking [1]

- We define the dataset $\mathcal{D} = \{< u, k, h > \mid k \in Re(u), h \notin Re(u)\}$
- Then we minimize the negative log likelihood over $\mathcal{D}$

$$\min \sum_{<u,k,h>\in\mathcal{D}} \ln(1 + e^{-(\hat{y}_{uk} - \hat{y}_{uh})}) + \text{L2 regularization}$$

Table of Contents

Motivation

- Insights from Matrix Factorization:

  ◦ Use auxiliary information (i.e. latent feedback) to overcome *data sparsity*

  ◦ Map auxiliary information into a lower-dimensional space of latent factors

Motivation

- Insights from Matrix Factorization:

  ◦ Use auxiliary information (i.e. latent feedback) to overcome *data sparsity*

  ◦ Map auxiliary information into a lower-dimensional space of latent factors

- Key task: learn the optimal mapping/encoding
- Previously: Find p_i, $q_j \in \mathbb{R}^k$ by minimizing L2 regression loss (with regularization) for each user i and item j
- New: Search for the best function f in a functional space $\mathcal{F}$ by gradient boosting [2]

Structure

Matrix Factorization:

$$\hat{Y}(i,j,x) = U^T(i,x)\ V(j,x), \text{ where } x \text{ is auxiliary information.}$$

Construct the user latent factor vector by a linear combination of features [2]

$$\forall \text{ latent dimension } k: \ U_k(i,x) = \sum_s \alpha_{k,s} f_{k,s}(i,x), \ f_{k,s} \in \mathcal{F}$$

∘ Usually, $\mathcal{F}$ is the class of decision trees or step functions.

[2] In implementation, the weights are a constant ϵ, also known as damping factor to avoid overfitting

Structure

Matrix Factorization:

$$\hat{Y}(i,j,x) = U^T(i,x)\ V(j,x), \text{ where } x \text{ is auxiliary information.}$$

Construct the user latent factor vector by a linear combination of features [2]

$$\forall \text{ latent dimension } k: \ U_k(i,x) = \sum_s \alpha_{k,s} f_{k,s}(i,x),\ f_{k,s} \in \mathcal{F}$$

∘ Usually, $\mathcal{F}$ is the class of decision trees or step functions.

Choose the optimal feature function f by minimizing

$$\mathcal{L} = \sum_{i,j} l(Y_{ij} - \hat{Y}_{ij}) + \sum_{k,s} \Omega(f_{k,s})$$

where Ω represents the *complexity* of a learner

[2] In implementation, the weights are a constant ϵ, also known as damping factor to avoid overfitting

Optimization Details

At a timestep, we want to add a feature function $f(i,x)$ to $U_k(i,x)$. Then

$$
\begin{aligned}
L(f) = & \sum_{i,j} l(\mathbf{Y}_{ij}, \hat{\mathbf{Y}}_{ij} + f(i,\mathbf{x})\mathbf{V}_{kj}) \\
= & \sum_{i,j} l(\mathbf{Y}_{ij}, \hat{\mathbf{Y}}_{ij}) + \sum_{i,j} (g_{ij}\mathbf{V}_{kj}) f(i,\mathbf{x}) \\
& + \frac{1}{2}\sum_{i,j}(h_{ij}\mathbf{V}_{kj}^2) f^2(i,\mathbf{x}) + o(f^2(i,\mathbf{x})) \\
= & \tilde{L}(f) + o(f^2(i,\mathbf{x}))
\end{aligned} \tag{5}
$$

We use $\tilde{L}(f)+\Omega(f)$ to find the best feature function f.

The regularization term is defined as

$$
\Omega(f) := \gamma|C| + \frac{1}{2}\lambda \sum_{c=1}^{|C|} \beta_c^2
$$

Optimization Details: Feature Function

Segmentation Function: divide data into different *clusters* based on auxiliary information, then learn a value for each cluster

- ◦ Complex, higher-order: CART (regression trees)
- ◦ Simple: step functions

Optimization Details: Feature Function

Segmentation Function: divide data into different *clusters* based on auxiliary information, then learn a value for each cluster

◦ Complex, higher-order: CART (regression trees)

◦ Simple: step functions

Let β_c denote the leaf value for cluster c.
We choose the optimal β_c by minimizing the regularized L2 loss.

$$\beta_c^* = -\frac{\sum_{i,j,\mathbf{x} \in I_c} g_{ij} \mathbf{V}_{kj}}{\sum_{i,j,\mathbf{x} \in I_c} h_{ij} \mathbf{V}_{kj}^2 + \lambda} \tag{9}$$

Time-based Step Functions

- Users' preference change over time $\Rightarrow$ model users' "local" (or short-term) preferences
- We divide the given period into a series of time bins and use **indicator functions** for each bin.

$$\hat{Y}(i,j,t) = U^T_{\mathrm{i,binid(t)}} V_j$$

- *User-specific*: Each user has his own feature function over time

Time-based Step Functions

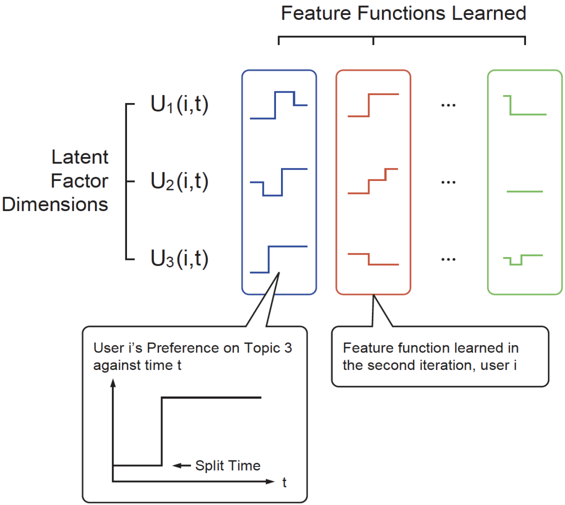

Figure 2. Illustration of Time-dependent Feature Function

Demographic-based Decision Trees

- Use users' demographic data (e.g. age, gender, occupation) to produce latent features
- Divide users into different cluster (leaves) based on demographic split rules
- Demographic means *user-independent*
- Efficiency compared with time-based step functions:

  ◦ Fewer times of feature function construction (users in the same cluster share a feature function f)
  ◦ Each individual f is more complex and involves more computational cost

Demographic-based Decision Trees

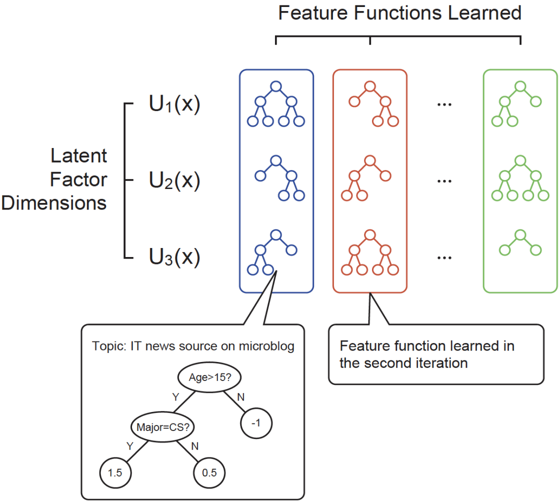

Figure 3. Illustration of Demographic Feature Function

Algorithm: Split Rule Search

Algorithm 2 Tree Split Finding with Missing Value

Require: I: instance set, n_p: number of properties

$gain \leftarrow 0$

$G_{all} \leftarrow \sum_{i,j \in I} g_{ij} \mathbf{V}_{kj}, H_{all} \leftarrow \sum_{i,j \in I} h_{ij} \mathbf{V}_{kj}^2$

for $p = 1$ **to** n_p **do**

  $I_p \leftarrow \{(i,j,x) \in I | x_p \neq missing\}$

  {enumerate default goto right}

  $G_{left} \leftarrow 0, H_{left} \leftarrow 0$

  for (i,j,x) in sorted(I_p, ascent order) **do**

    $G_{left} \leftarrow G_{left} + g_{ij} \mathbf{V}_{kj}$

    $H_{left} \leftarrow H_{left} + h_{ij} \mathbf{V}_{kj}^2$

    $G_{right} \leftarrow G_{all} - G_{left}, H_{right} \leftarrow H_{all} - H_{left}$

    $gain \leftarrow \max(gain, \frac{G_{left}^2}{H_{left}+\lambda} + \frac{G_{right}^2}{H_{right}+\lambda} - \frac{G_{all}^2}{H_{all}+\lambda})$

  end for

  {enumerate default goto left}

  $G_{right} \leftarrow 0, H_{right} \leftarrow 0$

  for (i,j,x) in sorted(I_p, descent order) **do**

    $G_{right} \leftarrow G_{right} + g_{ij} \mathbf{V}_{kj}$

    $H_{right} \leftarrow H_{right} + h_{ij} \mathbf{V}_{kj}^2$

    $G_{left} \leftarrow G_{all} - G_{right}, H_{left} \leftarrow H_{all} - H_{right}$

    $gain \leftarrow \max(gain, \frac{G_{left}^2}{H_{left}+\lambda} + \frac{G_{right}^2}{H_{right}+\lambda} - \frac{G_{all}^2}{H_{all}+\lambda})$

  end for

end for

output split and default direction with max gain

Table of Contents

Motivation

Problem:
random variables in CRF are dependent $\Rightarrow$ Hessian matrices are **dense**

Solution:

- Derive a Markov Chain mixing rate bound to quantify the dependencies
- Use gradient boosting to iteratively minimize an upper bound of the loss function

Conditional Random Fields (CRF)

Given an input $x \in \mathcal{X}$, a Conditional Random Field (CRF) defines the distribution over $y \in \mathcal{Y}$ as

$$P(y|x) = \frac{\exp\left(\Phi(y,x)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(\Phi(y',x)\right)}$$

The model Φ usually factorizes as a sum of unary and pairwise (edge) *potential function* $\phi_i : \mathcal{X} \to \mathbb{R}$

$$\phi(y,x) = \sum_{i=1}^{m} \phi_i(x)\mu_i(y), \ \phi_i \in \mathcal{F}, \mu_i \in \mathcal{N} \cup \mathcal{E}$$

$$\text{subject to } \phi_i = \phi_j \text{ for } (i,j) \in \mathcal{C}$$

Conditional Random Fields (CRF)

- $\mathcal{N} = \{\mathbb{1}(y_t = k)\}$, $\mathcal{E} = \{\mathbb{1}(y_s = k_1, y_t = k_2)\}$ are sets of indicator functions for each node and edge state
- Each μ_i corresponds to an event: $y_t = k$ for a node and $y_s = k_1, y_t = k_2$ for an edge

Conditional Random Fields (CRF)

- $\mathcal{N} = \{\mathbb{1}(y_t = k)\}$, $\mathcal{E} = \{\mathbb{1}(y_s = k_1, y_t = k_2)\}$ are sets of indicator functions for each node and edge state
- Each μ_i corresponds to an event: $y_t = k$ for a node and $y_s = k_1, y_t = k_2$ for an edge
- $\mathcal{F} = \mathcal{F}_N \cup \mathcal{F}_\mathcal{E}$ is the family of potential functions.

Conditional Random Fields (CRF)

- $\mathcal{N} = \{\mathbb{1}(y_t = k)\}$, $\mathcal{E} = \{\mathbb{1}(y_s = k_1, y_t = k_2)\}$ are sets of indicator functions for each node and edge state
- Each μ_i corresponds to an event: $y_t = k$ for a node and $y_s = k_1, y_t = k_2$ for an edge
- $\mathcal{F} = \mathcal{F}_N \cup \mathcal{F}_{\mathcal{E}}$ is the family of potential functions.
- $\mathcal{C}$ is the set of constraints, which defines the *parameter sharing* of potential functions

References

Kailong Chen et al. "Collaborative personalized tweet recommendation". In: *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. 2012, pp. 661–670.

Tianqi Chen et al. "General functional matrix factorization using gradient boosting". In: *International Conference on Machine Learning*. PMLR. 2013, pp. 436–444.

Yehuda Koren. "Factorization meets the neighborhood: a multifaceted collaborative filtering model". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 426–434.

Diyi Yang et al. "Collaborative filtering with short term preferences mining". In: *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. 2012, pp. 1043–1044.