ALS-WR for preferences with confidence

Notation. element of vector vs. i-th vector in a vector set:

$\mathbf{u}^{(k)}$ k-th vector in implied set of vectors

$\mathbf{u}_i$ i-th element of vector $\mathbf{u}$

Similarly for matrices.

User subscript is denoted by u and item's subscript is denoted by i.

Problem 1. (User-Item preferences): Given:

- user-item preference $p_{u,i} \in \{0, 1\}$, total matrix $\mathbf{P} \in \mathbb{R}^{m \times n}$ (m is number of users, n is number of items)
- user-item confidence matrix $\mathbf{C}$ consists of some weights $c_{u,i}$. This can be sparse where confidence is 0

We are solving matrix completion for $\mathbf{P} \approx \mathbf{U}^\top \mathbf{V}$ by finding $\mathbf{U} \in \mathbb{R}^{m \times k}$ and $\mathbf{V} \in \mathbb{R}^{n \times k}$.■

Solution. Denote row-vector $\mathbf{u}^{(u)} = \mathbf{U}^\top_{u,*}$, $\mathbf{v}^{(i)} = \mathbf{V}^\top_{i,*}$, $\mathbf{p}^{(u)} = \mathbf{P}^\top_{u,*}$, $\mathbf{p}'^{(i)} = \mathbf{P}_{*,i}$

Preference predictor

$$\hat{p}^{(u,i)} = \mathbf{u}^{(u)\top} \mathbf{v}^{(i)}, \tag{1}$$

or simply the whole prediction matrix

$$\hat{\mathbf{P}} = \mathbf{U}\mathbf{V}^\top \tag{2}$$

Minimizing cost with L2 reguralization

$$\sum_{u,i} \left[c_{u,i} \left(p_{ui} - \mathbf{u}^{(u)\top} \mathbf{v}^{(i)} \right)^2 \right] + \\ + \lambda \left(\sum_u n_u \left\| \mathbf{u}^{(u)} \right\|^2 + \sum_i n'_i \left\| \mathbf{v}^{(i)} \right\|^2 \right) \tag{3}$$

via alternating updates:

$\forall u \in 1..m \Rightarrow$

$$\mathbf{u}_u = \left(\mathbf{V}^\top \mathbf{D}^{(u)} \mathbf{V} + n\lambda \mathbf{I} \right)^{-1} \mathbf{V}^\top \mathbf{D}^{(u)} \mathbf{p}^{(u)} \tag{4}$$

where $\mathbf{D}^{(u)}$ is $n \times n$ diagonal matrix such that $\mathbf{D}^{(u)}_{i,i} = c_{u,i}$ and n_u is the number of measurements for user u such that $c_{ui} \neq 0$;

and $\forall i \in 1..n \Rightarrow$

$$\mathbf{v}_i = \left(\mathbf{U}^\top \mathbf{D}^{(i)} \mathbf{U} + n_i \lambda \mathbf{I} \right)^{-1} \mathbf{U}^\top \mathbf{D}^{(i)} \mathbf{p}^{(i)} \tag{5}$$

where $\mathbf{D}^{(i)}$ is $m \times m$ diagonal matrix such that $\mathbf{D}^{(i)}_{u,u} = c_{u,i}$ and n_i is the number of measurements for item i such that $c_{ui} \neq 0$.

The algorithm seeds $\mathbf{U}$ and $\mathbf{V}$ with small random numbers out of say $U(0, 0.01) - 0.005$ and then sequentially applies updates (4) and (5) to them until convergence. ■

Another thing to notice is that perhaps $n_u \ll n$ and $n'_i \ll m$ for many u and i and and therefore the equations can be efficiently blockified to throw away rows for which there's no observation, i.e. $c_{u,i} = 0$. Hence, dimensionality of these equations will typically be significantly reduced (as defined in [Zhou-1])

Minimum confidence in unobserved preference. Variation presented in [Koren-1] mostly differs in that it never assumes $c_{u,i} = 0$ (in fact, it assumes $c_{u,i} = 1$ and $p_{u,i} = 0$ whenever there's no observation at all to reflect the fact that there's a standardized tiny belief that no observation may mean lack of interest of the user in an item). This approach can be made computationally equivalent to the sparse observations (see details in the [Koren-1]).

In our situation it is especially important because we only observe clicks ($p_{click} = 1$, $c_{click} > c_0$), add-to-cart ($p_{cart} = 1$, $c_{cart} > c_0$) and conversions ($p_{conv} = 1$, $c_{conv} > c_{click}$) but we never directly observe ignored recommendation, i.e. we only observe $p = 1$ values but never $p = 0$. Assuming $c_0 = 0$ will likely lead to undersired effects in the linear system.

To generalize, we will assume minimum confidence for any directly unobserved preference p_0 being

some constant c_0 and increment any actual observed confidence with it:

$$c^*_{u,i} = c_0 + c_{u,i}.$$

Substituting $\mathbf{D}^{*(u)} = \mathbf{D}^{(u)} - c_0\mathbf{I}$, we modify (4) and (5) as

$$\mathbf{u}_u = \left(c_0\mathbf{V}^\top\mathbf{V} + \mathbf{V}^\top\mathbf{D}^{*(u)}\mathbf{V} + n_u\lambda\mathbf{I}\right)^{-1} \times$$
$$\times\, \mathbf{V}^\top\left(c_0\mathbf{I} + \mathbf{D}^{*(u)}\right)\mathbf{p}^{(u)} \qquad (6)$$
$$\forall u \in 1..m; \text{ and}$$

$$\mathbf{v}_i = \left(c_0\mathbf{U}^\top\mathbf{U} + \mathbf{U}^\top\mathbf{D}^{*(i)}\mathbf{U} + n_i\lambda\mathbf{I}\right)^{-1} \times$$
$$\times\, \mathbf{U}^\top\left(c_0\mathbf{I} + \mathbf{D}^{*(i)}\right)\mathbf{p}^{(i)} \qquad (7)$$
$$\forall i \in 1..n.$$

Here, $c_0\mathbf{V}^\top\mathbf{V}$ is precomputed once for all u per iteration; $\mathbf{V}^\top\mathbf{D}^{*(u)}\mathbf{V}$ is still sparse as before; and $\mathbf{p}^{(u)}$ keeps the part $\mathbf{V}^\top\left(c_0\mathbf{I} + \mathbf{D}^{*(u)}\right)\mathbf{p}^{(u)}$ still sparse (as we assume $\mathbf{P}_{u,i} = 0$ for any interaction we don't have data for).

Sparse data fold-in Also important thing to notice that sometimes the matrix under $(\cdot)^{-1}$ in (4) or (5) becomes singular and solution for that user or item is subsequently not possible. In particular, it is always the case whenever $n_u < k$ or $n'_i < k$.

When this happens, we don't attempt to compute shortened vector for the user or item by using, for example, n_u top principal columns of $\mathbf{U}$.

In that situation we still could use 6 to update the reduced-rank user vector by assuming $\mathbf{V}^{(u)}_{\text{princ}}$ instead of $\mathbf{V}$ where $\mathbf{V}^{(u)}_{\text{princ}} \in \mathbb{R}^{n\times\tilde{n}_u}$ is a vertical block of $\mathbf{V}$ containing top n_u most principal components of decomposition.

However, due to nature of the algorithm, we don't investigate emerging principal components during training and only reveal them at the end of the algorithm by reording correspondent columns in accorance to order of the set of values $\{s_i = \|\mathbf{U}_{*,i}\| \cdot \|\mathbf{V}_{*,i}\|\}$ $(\forall i = 1..k\ )$. Therefore, we cannot figure $\mathbf{V}$ in the course of algorithm. But at the end of the training cycle and figuring out principal vectors,

$$\mathbf{V}^{(u)}_{\text{princ}} = \mathbf{V}\left(:, 1 : n_u\right).$$

After that, any new observation for a new user u will look like

$$\tilde{\mathbf{u}}_u = \left[c_0\left(\mathbf{V}^{(u)}_{\text{princ}}\right)^\top\mathbf{V}^{(u)}_{\text{princ}} + \right.$$
$$\left. + \left(\mathbf{V}^{(u)}_{\text{princ}}\right)^\top\tilde{\mathbf{D}}^{(u)}\mathbf{V}^{(u)}_{\text{princ}} + \tilde{n}_u\lambda\mathbf{I}\right]^{-1} \times$$
$$\times \left(\mathbf{V}^{(u)}_{\text{princ}}\right)^\top\left(c_0\mathbf{I} + \tilde{\mathbf{D}}^{(u)}\right)\tilde{\mathbf{p}}_u \qquad (8)$$

where $\tilde{x}$ denotes anything related to a new observation of a new user. We denote matrix of results of all fold-in users as

$$\tilde{\mathbf{U}} = \begin{pmatrix} \tilde{\mathbf{u}}_1^\top \\ \tilde{\mathbf{u}}_2^\top \\ \vdots \\ \tilde{\mathbf{u}}_{m_1}^\top \end{pmatrix} \in \mathbb{R}^{\tilde{m}\times n}. \qquad (9)$$

This process can be applied symmetrically, if necessary, to fold-in items as well and obtain $\tilde{\mathbf{V}} \in \mathbb{R}^{m\times\tilde{n}}$.

Spark mapping notes.

Precompute steps.

- $\mathbf{U}$ and $\mathbf{V}$ initialized with random small numbers and presented as Spark-based DRM.
- Next, every row in $\mathbf{U}$ ($\mathbf{V}$) is connected with corresponded sparse vectors $\mathbf{c}^{(u)} = \mathbf{C}_{u,*}$ ($\mathbf{c}^{(i)} = \mathbf{C}_{*,i}$) and $\mathbf{p}^{(u)} = \mathbf{P}_{u,*}$ ($\mathbf{p}^{(i)} = \mathbf{P}_{*,i}$). So what we have in the end are two spark RDDs of Bagel vertices containing (id=$u|i$, $\mathbf{U}_{u,*}|\mathbf{V}_{*,i}$, $\mathbf{c}^{(u)}|\mathbf{c}^{(i)}$, $\mathbf{p}^{(u)}|\mathbf{p}^{(i)}$).

Iteration steps.

- $\mathbf{V}^\top\mathbf{V}$ precomputed in distributed way for each iteration and broadcast. Since this matrix gas tiny $k \times k$ geometry, collecting and broadcasting it should not be a problem.
- Form Bagel messages from current $\mathbf{V}$ side : $\mathbf{V}_i$ is sent to two sets of user vertices: (1) users for which $c_{u,i} \neq 0$; (2) users for which $p_{u,i} \neq 0$.
- During Bagel's compute(), group (1) is used to compute $\mathbf{V}^\top\mathbf{D}^{*(u)}\mathbf{V}$; group (2) is used to compute $\mathbf{V}^\top\mathbf{p}^{(u)}$. Update $\mathbf{U}_{u,*}$ per (6).
  - Use Cholesky to solve $\mathbf{AX} = \mathbf{B}$ by applying $\mathbf{A} = \mathbf{LL}^\top \Rightarrow \mathbf{X} = \left(\mathbf{L}^\top\right)^{-1}\mathbf{L}^{-1}\mathbf{B}$ which is `chol(A).solveRight(I) %*% chol(A).solveRight(B).`

Iteration updating $\mathbf{V}$ should be symmetrical.

References

[Koren-1] Y. Koren, et.al. Collaborative Filtering for Implicit Feedback Datasets

[Zhou-1] Y. Zhou, et. al. Large-scale Parallel Collaborative Filtering for the Netflix Prize