Alternating least squares for personalized ranking

Article · September 2012

DOI: 10.1145/2365952.2365972

2 authors:

Gábor Takács
Széchenyi István University, Gyor
22 PUBLICATIONS **721** CITATIONS

SEE PROFILE

Domonkos Tikk
Gravity Research & Development Zrt
197 PUBLICATIONS **3,022** CITATIONS

SEE PROFILE

Alternating Least Squares for Personalized Ranking

Gábor Takács [*]
Széchenyi István University, Dept. of
Mathematics and Computer Science
Egyetem tér 1.
Győr, Hungary
gtakacs@sze.hu

Domonkos Tikk
Gravity Research and Development Ltd.
Expo tér 5–7.
Budapest, Hungary
domonkos.tikk@gravityrd.com

ABSTRACT

Two flavors of the recommendation problem are the explicit and the implicit feedback settings. In the explicit feedback case, users rate items and the user–item preference relationship can be modelled on the basis of the ratings. In the harder but more common implicit feedback case, the system has to infer user preferences from indirect information: presence or absence of events, such as a user viewed an item. One approach for handling implicit feedback is to minimize a ranking objective function instead of the conventional prediction mean squared error. The naive minimization of a ranking objective function is typically expensive. This difficulty is usually overcome by a trade-off: sacrificing the accuracy to some extent for computational efficiency by sampling the objective function. In this paper, we present a computationally effective approach for the direct minimization of a ranking objective function, without sampling. We demonstrate by experiments on the Y!Music and Netflix data sets that the proposed method outperforms other implicit feedback recommenders in many cases in terms of the ErrorRate, ARP and Recall evaluation metrics.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*parameter learning*

Keywords

alternating least squares, ranking, collaborative filtering

1. INTRODUCTION

The goal of recommender systems is to provide personalized recommendations on items to users [1]. The basis of the recommendation is user event history related to items, and metadata about users and items. Two flavors of the recommendation problem are the explicit and the implicit feedback settings.

In the *explicit feedback* case, users rate items, therefore the preference relationship between a set of user–item pairs is directly known. A famous example of the explicit feedback setting is the movie recommendation problem of the Netflix Prize competition [2]. A typical characteristic of explicit feedback problems is that user–item pairs where no rating is present are not necessarily needed for modeling users' taste.

In many application domains, however, ratings are not available. The recommender system has to infer user preferences from *implicit feedback* [4], such as the presence or absence of purchase, click, or search events. A significant part of the information is contained in the absence of events, therefore the system has to consider each user–item pair. This makes the implicit feedback problem substantially harder than the explicit feedback one. This paper deals with the implicit feedback variant of the recommendation problem.

Many of the known methods for giving recommendation can be classified as prediction based or ranking based. *Prediction based* methods (e.g. [4]) try to predict the presence or absence of interaction between users and items. *Ranking based* methods (e.g. [5]) try to model users' choice between item pairs. In both cases, model building (training) is typically executed as the minimization of an objective function.

In the implicit feedback case, the naive minimization of the objective function is expensive. This difficulty is usually overcome by a trade-off: sacrificing the accuracy to some extent for computational efficiency by sampling the objective function.

In this paper, we propose RankALS, a computationally effective approach for the direct minimization of a ranking objective function without sampling, that is able to cope with the implicit feedback case. We show an efficient implementation of this method; for that we apply a similar but more complex derivation as used in ImplicitALS [4]. We also prove that the time complexity of RankALS is identical with that of ImplicitALS. We perform experiments on two "implicitized" rating data sets, Y!Music and Netflix. In a comparison we show that RankALS outperforms ImplicitALS in most cases in terms of three evaluation metrics, and it is usually better than ranking based methods employing the sampling strategy.

The rest of the paper is organized as follows. First, the notation is introduced. In Section 2, prediction and ranking based objective functions are reviewed. Next, we survey related work and highlight ImplicitALS, the algorithm that inspired RankALS. Section 4 introduces RankALS; first, its

[*] Also affiliated with Gravity Research and Development Ltd., Expo tér 5–7, Budapest, Hungary.

RecSys'12, September 9–13, 2012, Dublin, Ireland.

main idea and its advantageous computational properties are shown, then pseudocode with explanation is presented. The last part discusses experiments and performance comparisons with other methods on two publicly available large-scale data sets.

1.1 Notation

U and I will denote the number of users and items. $\mathcal{U} = \{1, \ldots, U\}$ and $\mathcal{I} = \{1, \ldots, I\}$ will denote the set of items and users respectively. We use $u \in \mathcal{U}$ as index for users, and $i, j \in \mathcal{I}$ as indices for items.

The implicit rating of user u on item i is r_{ui}, and its prediction is $\hat{r}_{ui}$. Implicit ratings r_{ui} are arranged in the matrix $R \in \mathbb{R}^{U \times I}$. $\mathcal{T}$ denotes the set of (u, i) indexes of R where a positive feedback is provided from user u on item i. We assume that the implicit rating value is 0 for negative feedback (i.e. $r_{ui} = 0$, if $(u, i) \notin \mathcal{T}$) and positive for positive feedback. We assume that users give positive feedback for items at most once.

$T = |\mathcal{T}|$ denotes the number of positive feedbacks. $\mathcal{I}_u = \{i : (u, i) \in \mathcal{T}\}$ denotes the set of items for which a positive feedback is provided by user u. $\mathcal{U}_i = \{u : (u, i) \in \mathcal{T}\}$ denotes the set of user who provided positive feedback for item i. $z_u = |\mathcal{I}_u|$ denotes the number of positive feedbacks of user u and $z \in \mathbb{R}^U$ denotes the vector of $\mathcal{I}_u$ values.

We also employ the usual matrix algebraic notations $\operatorname{diag}(x)$, for the diagonal matrix containing the elements of vector x in its main diagonal, and $\operatorname{tr}(X)$ for the trace of square matrix X. In matrix algebraic expressions, 1 means a vector of all ones of appropriate size.

$X_{[\mathcal{S}]}$ denotes the submatrix of matrix X selected by the row index set $\mathcal{S}$, and $x_{[\mathcal{S}]}$ denotes the subvector of vector x selected by the index set $\mathcal{S}$. $X_{[\mathcal{S}_1, \mathcal{S}_2]}$ denotes the submatrix of X selected by the row index set $\mathcal{S}_1$ and the column index set $\mathcal{S}_2$. Submatrix selection has precedence over the transpose operation, i.e. $X^T_{[\ldots]}$ is evaluated as $(X_{[\ldots]})^T$.

2. OBJECTIVE FUNCTIONS

Two major approaches for the recommendation problem are prediction based and ranking based recommendation. The debate on which approaches is better and when one is preferred to the other is still active. In the next subsections, we define objective functions for the two approaches. We will omit regularization terms in order to keep the discussion shorter.

2.1 Prediction based objective function

The objective function associated with a simple prediction based approach can be defined as

$$f_P(\Theta) = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} c_{ui} (\hat{r}_{ui} - r_{ui})^2,$$

where Θ contains the parameters of the prediction model. [1] The $c_{ui} > 0$ values are parameters of the objective function that have to be fixed in advance. The goal of model building is to make the sum of weighted squared differences between the predicted and true ratings small. c_{ui} means the extent to which we penalize the error on user u and item i.

The standard choice for c_{ui} in the explicit feedback case is $c_{ui} = 1$, if $(u, i) \in \mathcal{T}$ and 0 otherwise, which means that the objective function contains only T terms. In the implicit feedback case, this simplification is not reasonable, because a significant part of the information is contained in the absence of positive feedback.

A typical choice for c_{ui} in the implicit feedback case is $c_{ui} = c^+$, if $(u, i) \in \mathcal{T}$ and c^- otherwise. The values c^+ and c^- can be interpreted as confidence levels associated with positive and negative feedbacks respectively. We will refer to this specific variant of the prediction based objective function as f_I (I stands for implicit). The function f_I contains $U \cdot I$ terms, therefore its naive minimization is expensive.

Usually, $c^+ \gg c^-$, which means that the presence of positive feedback has to be predicted more accurately than the absence of positive feedback. To see the rationale behind this asymmetry, consider a webshop scenario. If user u has bought item i that we can consider this as a positive feedback with high confidence. If i is not present in the purchase history of u, then we can treat this as a negative feedback from user u on item i, but only with low confidence.

Note that the interpretation of implicit feedback data may not necessarily reflect user satisfaction. Even the presence of positive feedback may be misleading: (1) a purchased item could be disappointing for the user but the purchase information is generated before the user could evaluate the product; (2) the user may purchase the item for someone else, therefore her selection only reflects her knowledge and assumption on the preference of another person. The interpretation of the absence of positive feedback is even more speculative: we cannot directly interpret missing navigational or purchase information as an intentional negative feedback; in the implicit setting direct negative feedback is not available.

2.2 Ranking based objective function

The objective function associated with a simple ranking based approach can be defined as

$$f_R(\Theta) = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}} c_{ui} \sum_{j \in \mathcal{I}} s_j \left[(\hat{r}_{ui} - \hat{r}_{uj}) - (r_{ui} - r_{uj}) \right]^2,$$

where the c_{ui} and s_j values are parameters of the objective function. This function was introduced by [5] in the recommender system literature (although other ranking based objectives appeared earlier). Here we only consider the implicit feedback case and assume that $c_{ui} = 0$ if $r_{ui} = 0$, and 1 otherwise. The role of c_{ui} is to select user–item pairs corresponding to positive feedbacks from all possible pairs. The meaning of s_j is the importance weight of the j-th item in the objective function. Note that the number of terms in f_R is $T \cdot I$, therefore the naive minimization of f_R is expensive.

Note that there is an asymmetric relation between the objective functions f_I and f_R. A good solution with respect to f_I will also be a good solution with respect to f_R, but the opposite is not necessarily true. One can also observe that user-only dependent terms in $\hat{r}_{ui}$ do not change the value of f_R, in other words, f_R is invariant to user biases.

[1] Note that $\hat{r}_{ui}$ depends on Θ, but for brevity, we will not write $\hat{r}_{ui}(\Theta)$ explicitly.

3. RELATED WORK

The literature of explicit feedback based recommendation algorithms is rich. We do not highlight any approach here, since this paper focuses on implicit feedback, just direct the interested reader to the survey work [10].

The area of implicit feedback based recommendation is much less explored. One of the earliest solutions for handling implicit feedback is Hu et al.'s ImplicitALS [4] that consists of a matrix factorization model, a prediction based objective function, and an alternating least squares (ALS) optimizer, tuned for the problem (see a more detailed discussion in Section 3.1).

A faster, approximative version of ImplicitALS was proposed by Pilászy et al. [8]. The speedup is achieved by replacing the exact least squares solver by a coordinate descent method. The authors reported on a marginal loss of accuracy compared to the significant decrease in training time in their experiments.

Pan et al. [7] proposed two prediction based frameworks for handling implicit feedback. The first one is similar to ImplicitALS, but it contains a naive ALS optimizer instead of a tuned one. The second one is based on negative example sampling.

Recently, the results of Track 2 at KDD Cup 2011 [3] have shown an evidence that ranking based approaches are able to handle implicit feedback efficiently on large-scale problems. Ranking based methods had significant weight in the winning solution [6], and they were the most important components of the second runner-up solution [5]. In particular, the key idea of the latter is to define a ranking based objective function, and apply a stochastic gradient descent (SGD) optimizer on a sampled approximation of the objective function.

Another ranking based approach for implicit feedback is Rendle et al's Bayesian Personalized Ranking (BPR) [9]. The objective function of BPR is derived from the Bayesian analysis of the problem. The optimization technique used for training is bootstrapping based SGD.

Our proposed RankALS method differs from all of the above approaches in various aspects:

- Unlike [4], [8] and [7], it is ranking based.
- Unlike [9] and the ranking based methods in [5] and [6], it does not approximate the original objective function, and applies ALS for training instead of SGD.

3.1 ALS for prediction

Here we overview the main points of Hu et al.'s ImplicitALS [4] that can be considered as a straight predecessor of our proposed RankALS method. The prediction formula of ImplicitALS for user u and item i is $\hat{r}_{ui} = p_u^T q_i$. The parameters of the model $p_1, \dots, p_U \in \mathbb{R}^F$ and $q_1, \dots, q_I \in \mathbb{R}^F$ are called user and item feature vectors; F denotes the number of features. [2]

Let P and Q denote the matrices that contain user and item feature vectors as rows. The objective function associated with the model is f_I, thus the goal of ImplicitALS is to approximate the rating matrix as $R \approx PQ^T$ so that the weighted squared error of the approximation is low.

The key idea of ImplicitALS is that although the objective function consists of $U \cdot I$ terms, its alternating least squares (ALS) based minimization can be done efficiently. The outline of the training of ImplicitALS is given in Algorithm 1.

Algorithm 1: Alternating least squares based training.

```
Initialize $Q$ with small random numbers.
for $E$ times do
    Compute the $P$ that minimizes $f_I$ for fixed $Q$.
    Compute the $Q$ that minimizes $f_I$ for fixed $P$.
end
```

To see why this scheme can be implemented efficiently, let us rewrite the derivative of f_I with respect to p_u as

$$\frac{\partial f_I(P,Q)}{\partial p_u} = \sum_{i \in \mathcal{I}} c_{ui} \left(q_i^T p_u - r_{ui} \right) q_i =$$
$$\underbrace{\left(\sum_{i \in \mathcal{I}} c_{ui} q_i q_i^T \right)}_{\bar{A}_u} p_u - \underbrace{\sum_{i \in \mathcal{I}} c_{ui} r_{ui} q_i^T}_{\bar{b}_u} = \bar{A}_u p_u - \bar{b}_u.$$

Minimizing f_I in P means that we make the derivatives vanish by setting p_u to $\bar{A}_u^{-1}\bar{b}_u$ for all u. Note that $\bar{b}_u$ can be simplified to $\sum_{i \in \mathcal{I}_u}$ and $\bar{A}_u$ can be decomposed into a user-independent and a user-dependent part as $\bar{A}_u = \sum_{i \in \mathcal{I}} c^- q_i q_i^T + \sum_{i \in \mathcal{I}_u} (c^+ - c^-) q_i q_i^T$. As a consequence, the evaluation of $\bar{A}_u^{-1}\bar{b}_u$ for all u can be done in $O(TF^2 + UF^3)$ time.

Applying the same argument for items yields that item feature vectors can be updated in $O(TF^2 + IF^3)$ time. Therefore, the computational cost of one training step is $O(TF^2 + (U+I)F^3)$ which means much less operations than iterating over the terms of the objective function.

A nice property of ImplicitALS is that it does not replace objective function by an approximation, as other approaches often do. It achieves speedup by applying mathematical simplification.

4. ALS FOR RANKING

This section presents RankALS, our proposed method for personalized ranking. The prediction formula of RankALS is $\hat{r}_{ui} = p_u^T q_i$ as in the case of ImplicitALS. The difference is that the function to minimize is now the ranking objective function f_R.

Recall that the ranking objective function contains $T \cdot I$ terms. We will show that even though f_R contains more and more complex terms than f_I, its ALS based minimization can be done at the same asymptotic computational cost. We will use a similar ideas as for ImplicitALS to break down the computational complexity of the naive implementation of RankALS, but the mathematical derivation and the resulting training algorithm will be more complex.

At first, let us rewrite the derivative of f_R with respect to

[2] We omitted user and item bias parameters, because they are not necessary for illustrating the ideas.

the user feature vector p_u.

$$\frac{\partial f_R(P,Q)}{\partial p_u} =$$

$$\sum_{i\in\mathcal{I}} c_{ui} \sum_{j\in\mathcal{I}} s_j \left[(q_i - q_j)^T p_u - (r_{ui} - r_{uj}) \right] (q_i - q_j) =$$

$$\underbrace{\left(\sum_{j\in\mathcal{I}} s_j\right)}_{\tilde{1}} \underbrace{\left(\sum_{i\in\mathcal{I}} c_{ui} q_i q_i^T\right)}_{\bar{A}} p_u - \underbrace{\left(\sum_{i\in\mathcal{I}} c_{ui} q_i\right)}_{\bar{q}} \underbrace{\left(\sum_{j\in\mathcal{I}} s_j q_j^T\right)}_{\tilde{q}^T} p_u -$$

$$\underbrace{\left(\sum_{j\in\mathcal{I}} s_j q_j\right)}_{\tilde{q}} \underbrace{\left(\sum_{i\in\mathcal{I}} c_{ui} q_i^T\right)}_{\bar{q}^T} p_u + \underbrace{\left(\sum_{i\in\mathcal{I}} c_{ui}\right)}_{\bar{1}} \underbrace{\left(\sum_{j\in\mathcal{I}} s_j q_j q_j^T\right)}_{\tilde{A}} p_u -$$

$$\underbrace{\left(\sum_{i\in\mathcal{I}} c_{ui} q_i r_{ui}\right)}_{\bar{b}} \underbrace{\left(\sum_{j\in\mathcal{I}} s_j\right)}_{\tilde{1}} + \underbrace{\left(\sum_{i\in\mathcal{I}} c_{ui} q_i\right)}_{\bar{q}} \underbrace{\left(\sum_{j\in\mathcal{I}} s_j r_{uj}\right)}_{\tilde{r}} +$$

$$\underbrace{\left(\sum_{i\in\mathcal{I}} c_{ui} r_{ui}\right)}_{\bar{r}} \underbrace{\left(\sum_{j\in\mathcal{I}} s_j q_j\right)}_{\tilde{q}} - \underbrace{\left(\sum_{i\in\mathcal{I}} c_{ui}\right)}_{\bar{1}} \underbrace{\left(\sum_{j\in\mathcal{I}} s_j q_j r_{uj}\right)}_{\tilde{b}} =$$

$$\left(\tilde{1}\bar{A} - \bar{q}\tilde{q}^T - \tilde{q}\bar{q}^T + \bar{1}\tilde{A}\right) p_u -$$

$$\left(\bar{b}\tilde{1} - \bar{q}\tilde{r} - \bar{r}\tilde{q} + \bar{1}\tilde{b}\right).$$

We converted the expression of the derivative to 8 double sums by expanding the inner term. Then, we factorized each double sum in order to make the evaluation of the expression more efficient. The advantage of this rewritten form of the derivative is that computing the statistics $\bar{A}$, $\bar{b}$, $\bar{q}$, $\bar{r}$, $\bar{1}$, $\tilde{A}$, $\tilde{b}$, $\tilde{q}$, $\tilde{r}$, $\tilde{1}$ for all u can be done in $O(TF^2)$ time, therefore the evaluation of $\frac{\partial f_R(P,Q)}{\partial P}$ takes $O(TF^2)$ time. Note that except $\tilde{A}$, $\tilde{q}$ and $\tilde{1}$ the statistics depend on u, but we do not subscript them by u in order to make formulae cleaner.

Now let us rewrite the derivative of f_R with respect to q_i.

$$\frac{\partial f_R(P,Q)}{\partial q_i} =$$

$$\sum_{u\in\mathcal{U}} \sum_{j\in\mathcal{I}} c_{ui} s_j p_u \left[p_u^T (q_i - q_j) - (r_{ui} - r_{uj}) \right] +$$

$$\sum_{u\in\mathcal{U}} \sum_{j\in\mathcal{I}} c_{uj} s_i (-p_u) \left[p_u^T (q_j - q_i) - (r_{uj} - r_{ui}) \right] =$$

$$\underbrace{\left(\sum_{u\in\mathcal{U}} c_{ui} p_u p_u^T\right)}_{\bar{A}} \underbrace{\left(\sum_{j\in\mathcal{I}} s_j\right)}_{\tilde{1}} q_i - \underbrace{\left(\sum_{u\in\mathcal{U}} c_{ui} p_u p_u^T\right)}_{\bar{A}} \underbrace{\left(\sum_{j\in\mathcal{I}} s_j q_j\right)}_{\tilde{q}} -$$

$$\underbrace{\left(\sum_{u\in\mathcal{U}} c_{ui} p_u r_{ui}\right)}_{\bar{b}} \underbrace{\left(\sum_{j\in\mathcal{I}} s_j\right)}_{\tilde{1}} + \underbrace{\left(\sum_{u\in\mathcal{U}} c_{ui} \overbrace{\left(\sum_{j\in\mathcal{I}} s_j r_{uj}\right)}^{\tilde{r}} p_u\right)}_{\bar{\bar{p}}_1} -$$

$$\underbrace{\left(\sum_{u\in\mathcal{U}} p_u p_u^T \overbrace{\left(\sum_{j\in\mathcal{I}} c_{uj} q_j\right)}^{\bar{q}}\right)}_{\bar{\bar{p}}_2} s_i + \underbrace{\left(\sum_{u\in\mathcal{U}} p_u p_u^T \overbrace{\left(\sum_{j\in\mathcal{I}} c_{uj}\right)}^{\bar{1}}\right)}_{\bar{\bar{A}}} s_i q_i +$$

$$\underbrace{\left(\sum_{u\in\mathcal{U}} \overbrace{\left(\sum_{j\in\mathcal{I}} c_{uj} r_{uj}\right)}^{\bar{r}} p_u\right)}_{\bar{\bar{p}}_3} s_i - \underbrace{\left(\sum_{u\in\mathcal{U}} p_u r_{ui} \overbrace{\left(\sum_{j\in\mathcal{I}} c_{uj}\right)}^{\bar{1}}\right)}_{\bar{\bar{b}}} s_i =$$

$$\left(\bar{A}\tilde{1} + \bar{\bar{A}}s_i\right) q_i + \left(-\bar{A}\tilde{q} - \bar{b}\tilde{1} + \bar{\bar{p}}_1 - \bar{\bar{p}}_2 + \bar{\bar{p}}_3 s_i - \bar{\bar{b}}s_i\right).$$

We applied the same idea as before. The expression of the derivative was converted into 8 double sums, and then the double sums were factorized. Again, computing the statistics $\bar{A}$, $\bar{b}$, $\bar{1}$, $\tilde{q}$, $\tilde{1}$, $\bar{\bar{A}}$, $\bar{\bar{b}}$, $\bar{\bar{p}}_1$, $\bar{\bar{p}}_2$, $\bar{\bar{p}}_3$ for all i can be done in $O(TF^2)$ time, therefore the evaluation of $\frac{\partial f_R(P,Q)}{\partial Q}$ takes $O(TF^2)$ time.

Note that $\bar{A}$ and $\bar{b}$ are defined differently as in the user-derivative. Also note that except $\tilde{A}$, $\tilde{q}$ and $\tilde{1}$ the statistics depend on i or u, but we avoided subscripting them.

The previous results imply that the ALS based minimization of f_R with a matrix factorization model can be implemented efficiently. The two main steps of ALS are evaluating the derivatives and setting them to zero. We have seen that the first step can be speeded up by rewriting the derivatives. The second step is relatively cheap, because it consists of solving linear systems.

Now we are ready to put blocks together and specify the model building procedure of RankALS. The pseudocode of RankALS training can be seen in Algorithm 2.

Each iteration consists of a P-step and a Q-step. Most of the pseudocode deals with the calculation of statistics needed for derivative evaluation. The statistic $\tilde{1}$ is constant during the algorithm, therefore it is computed before the main loop.

The P-step starts with calculating the statistics $\tilde{q}$ and $\tilde{A}$ that are common for each user. Then, for each user u, the user-dependent statistics $\bar{1}$, $\bar{r}$, $\bar{q}$, $\bar{b}$, $\bar{A}$, $\tilde{r}$, and $\tilde{b}$ are computed, and p_u is updated by setting the derivative $\frac{\partial f_R(P,Q)}{\partial p_u}$ to zero.

The Q-step starts with calculating the item-independent statistics $\tilde{q}$, $\bar{\bar{A}}$ and $\bar{p}$. Next, the more complex item-independent statistics $\tilde{r}$, $\bar{\bar{p}}_2$ and $\bar{\bar{p}}_3$ are calculated by iterating over users. Then, for each item i, the item-dependent statistics $\bar{b}$, $\bar{A}$, $\bar{\bar{b}}$ and $\bar{\bar{p}}_1$ are computed, and q_i is updated by setting the derivative $\frac{\partial f_R(P,Q)}{\partial q_i}$ to zero.

The tunable parameters of the algorithm are as follows:

- $E \in \mathbb{N}$: The number of iterations.
- $\sigma \in \mathbb{R}$: Initialization range parameter.
- $s \in \mathbb{R}^I$: Item importance weights in the objective function f_R. We experimented with two settings:
  - (i) $s_i = 1$, i.e. items are equally important,
  - (ii) $s_i = |\mathcal{U}_i|$, i.e. popular items are more important.

The computationally most expensive parts of the P-step are calculating $\bar{A}$ and updating p_u. Computing $\bar{A}$ and p_u for

Algorithm 2: Alternating least squares for ranking.

```
Input: $R \in \mathbb{R}^{U \times I}$, $E \in \mathbb{N}$, $\sigma \in \mathbb{R}$, $s \in \mathbb{R}^{I}$
Output: $P \in \mathbb{R}^{U \times F}$, $Q \in \mathbb{R}^{I \times F}$

$Q \leftarrow$ uniform random numbers from $[-\sigma, \sigma]$
$\tilde{1} \leftarrow \sum_{i \in \mathcal{I}} s_i$

for $e \leftarrow 1, \ldots, E$ do
    // P-step
    $\tilde{q} \leftarrow Q^T s,\ \tilde{A} \leftarrow Q^T \mathrm{diag}(s) Q$

    for $u \leftarrow 1, \ldots, U$ do ;                    // for each user
        $r_u \leftarrow R^T_{[\{u\}, \mathcal{I}_u]},\ \ \bar{1} \leftarrow z_u$
        $\bar{r} \leftarrow r_u^T 1,\qquad \bar{q} \leftarrow Q^T_{[\mathcal{I}_u]} 1$
        $\bar{b} \leftarrow Q^T_{[\mathcal{I}_u]} r_u,\qquad \bar{A} \leftarrow Q^T_{[\mathcal{I}_u]} Q_{[\mathcal{I}_u]}$
        $\tilde{r} \leftarrow s^T_{[\mathcal{I}_u]} r_u,\qquad \tilde{b} \leftarrow Q^T_{[\mathcal{I}_u]} \mathrm{diag}(s_{[\mathcal{I}_u]}) r_u$

        $M \leftarrow \tilde{1}\bar{A} - \bar{q}\tilde{q}^T - \tilde{q}\bar{q}^T + \bar{1}\tilde{A}$
        $y \leftarrow \bar{b}\tilde{1} - \bar{q}\tilde{r} - \bar{r}\tilde{q} + \bar{1}\tilde{b}$
        $p_u \leftarrow M^{-1} y$
    end
    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    // Q-step
    $\tilde{q} \leftarrow Q^T s,\ \bar{\bar{A}} \leftarrow P^T \mathrm{diag}(z) P,\ \bar{p} \leftarrow P^T z$

    $\tilde{r},\ \bar{r},\ \bar{Q} \leftarrow 0$
    for $u \leftarrow 1, \ldots, U$ do ;                    // for each user
        $r_u \leftarrow R^T_{[\{u\}, \mathcal{I}_u]},\ \ \tilde{r}_u \leftarrow r_u^T s_{[\mathcal{I}_u]}$
        $\bar{r} \leftarrow r_u^T 1,\qquad \bar{Q}_{[\{u\}]} \leftarrow Q^T_{[\mathcal{I}_u]} 1$
    end
    $\bar{\bar{p}}_2 \leftarrow P^T \mathrm{tr}(P, \bar{Q}),\ \bar{\bar{p}}_3 \leftarrow P^T \bar{r}$

    for $i \leftarrow 1, \ldots, I$ do ;                    // for each item
        $r_i \leftarrow R_{[\mathcal{U}_i, \{i\}]}$
        $\bar{b} \leftarrow P^T_{[\mathcal{U}_i]} r_i,\qquad \bar{A} \leftarrow P^T_{[\mathcal{U}_i]} P_{[\mathcal{U}_i]}$
        $\bar{\bar{b}} \leftarrow P^T_{[\mathcal{U}_i]} \mathrm{diag}(z_{[\mathcal{U}_i]}) r_i,\quad \bar{\bar{p}}_1 \leftarrow P^T_{[\mathcal{U}_i]} \tilde{r}_{[\mathcal{U}_i]}$

        $M \leftarrow \bar{A}\tilde{1} + \bar{\bar{A}} s_i$
        $y \leftarrow \bar{A}\tilde{q} + \bar{b}\tilde{1} - \bar{\bar{p}}_1 + \bar{\bar{p}}_2 - \bar{\bar{p}}_3 s_i + \bar{\bar{b}} s_i$
        $q_i \leftarrow M^{-1} y$
    end
end
```

all users takes $O(TF^2)$ and $O(UF^3)$ time respectively. The most expensive parts of the Q-step are calculating $\bar{A}$ and updating q_i Computing $\bar{A}$ and q_i for all items takes $O(TF^2)$ and $O(IF^3)$ time respectively. The overall complexity of one iteration is $O(TF^2 + (U + I)F^3)$, which is the same as for ImplicitALS.

RankALS training is a relatively complex algorithm, and implementing it without error is not trivial. We validated our code by also implementing naive derivative evaluation and comparing the two variants. Apart from the very tiny difference due to round-off errors, the two versions produced the same result.

5. EXPERIMENTS

We compared our proposed method with other approaches on "implicitized" versions of the Y!Music [3] and the Netflix [2] data sets. The characteristics of the data sets are as follows:

- **Y!Music**: The Y!Music data set for Track2 of KDD Cup 2011 consists of 62,551,438 ratings ranging from 0 to 100 from 249,012 users on 296,111 items. The items are arranged in a taxonomy: the leaf nodes are tracks, the higher level nodes are albums, artists and genres. We considered a rating as a positive implicit feedback, if the rating value was greater than or equal to 80. Moreover, we considered each node of the hierarchy as a separate item, and did not exploit the parent-child relationship between items. For training and testing we used the official Track2 training and test sets (containing 22,699,314 and 303,516 implicit ratings respectively).

- **Netflix**: The Netflix training set consists of 100,480,507 ratings ranging from 1 to 5 from 480,189 users on 17,770 items. We defined the implicit rating matrix by assigning 1 to user–item pairs with Netflix rating value 5, and 0 to other user–item pairs. As training and test sets, we used the Netflix training set minus the Netflix probe and the Netflix probe set (containing 22,783,659 and 384,573 implicit ratings respectively).

We compared the following methods:

- **Pop**: Item popularity based, non-personalized baseline method. Its prediction for user u and item i is $|\mathcal{U}_i|$, the number of positive feedbacks for i.

- **ImplicitALS**: The approach described in [4]. It uses an MF model, a prediction based objective function and an ALS optimizer.

- **RankSGD**: The approach described in [5] under the name SVD. It uses an MF model, a ranking based objective function and an SGD optimizer.

- **RankSGD2**: The approach described in [5] under the name AFM. It uses an asymmetric factor (NSVD1) model, a ranking based objective function and an SGD optimizer.

- **RankALS**: Our proposed method that uses an MF model, a ranking based objective function, and an ALS optimizer.

We measured the following performance indicators:

- **ErrorRate**: The evaluation metric used in Track 2 of KDD Cup 2011 [3]. For ErrorRate measurement, the test set contains two sets of items $\mathcal{I}_u^+$ and $\mathcal{I}_u^-$ for each user u. $\mathcal{I}_u^+$ ($\mathcal{I}_u^-$) contains items for that u provided (did not provide) positive feedback. The negative item set $\mathcal{I}_u^-$ is created by drawing $|\mathcal{I}_u^+|$ items randomly so that the probability of item i to be chosen is proportional to $|\mathcal{U}_i|$.[3] For each user u, the predicted rating of u for items in $\mathcal{I}_u^+ \cup \mathcal{I}_u^-$ is calculated and items are ranked based on these values. ErrorRate is the relative frequency of items from $\mathcal{I}_u^+$ not being among the top $|\mathcal{I}_u^+|$ items of the ranking. A lower ErrorRate value means better performance.

- **ARP**: The test set contains a set of preferred items $\mathcal{I}_u^+$ for each user u, and another set of items $\mathcal{I}'$ that is shared between users. Normally $\mathcal{I}'$ contains all items, but it can be a random subset, if there are too many items in the data set. For each user u, the elements of $\mathcal{I}'$ are ranked, based on prediction values. ARP is the average relative position of items from $\mathcal{I}_u^+$ in the ranking [8]. A lower ARP value indicates better performance.

- **Recall**: The structure of the test set, and the ranking procedure is the same as for ARP. Recall is the relative frequency of items from $\mathcal{I}_u^+$ being among the top K items of the ranking. A higher Recall value means better performance.

The set $\mathcal{I}'$ of ARP and Recall measurement contained all 17,770 items in the case of the Netflix data set, and 17,770 uniformly drawn items in the case of the Y!Music data set. For Y!Music, the sets $\mathcal{I}_u^-$ were taken from the official test set of KDD Cup 2011, Track 2. For Netflix, the sets $\mathcal{I}_u^-$ were drawn randomly, since no official negative item sets are available. The parameter K of Recall measurement was 50.

The learning rate was $\eta = 0.008$ for RankSGD and $\eta = 0.016$ for RankSGD2 in all experiments. We did not apply regularization for RankSGD, RankSGD, and RankALS.[4] For ImplicitALS and RankALS, training started with a Q-step and not with a P-step as in the pseudocodes. For SGD-based approaches we ran $E = 30$, for ALS-based approaches we ran $E = 10$ iterations. We stopped training, if the performance started to decrease on the test set. The confidence level parameters of the objective function f_I were set to $c^+ = 100$ and $c^- = 1$.

5.1 Support based weighting

The choice of the item weight vector s can greatly influence the behavior of the ranking based approaches (RankSGD, RankSGD2, RankALS). In the first experiment, we tried to find the appropriate variant of the methods for the different evaluation criteria. The results for the Y!Music and the Netflix data set are shown in Table 1 and 2. The column s.w. indicates if support based weighting was applied ($s_i = |\mathcal{U}_i|$) or not ($s_i = 1$).

The results suggest that support based weighting should be turned on if one optimizes for ErrorRate. This is not surprising, since the negative items of the test set are drawn with probability proportional to $|\mathcal{U}_i|$ in the case of ErrorRate. In further experiments with ErrorRate, we will always use the setting s.w. = yes.

method	**s.w.**	**ErrorRate**	**ARP**	**Recall**
RankSGD	no	0.1874	**0.0362**	0.2153
RankSGD	yes	**0.0617**	0.0387	**0.3574**
RankSGD2	no	0.1163	**0.0275**	**0.3815**
RankSGD2	yes	**0.0642**	0.0416	0.3333
RankALS	no	0.0961	**0.0220**	**0.3719**
RankALS	yes	**0.0529**	0.0375	0.2562

Table 1: Comparison of ranking based method variants on the Y!Music data set ($F = 20$)

method	**s.w.**	**ErrorRate**	**ARP**	**Recall**
RankSGD	no	0.3103	**0.0509**	0.1525
RankSGD	yes	**0.1861**	0.1234	**0.1770**
RankSGD2	no	0.2651	**0.0572**	**0.1757**
RankSGD2	yes	**0.2011**	0.1291	0.1589
RankALS	no	0.2878	**0.0478**	**0.1731**
RankALS	yes	**0.1842**	0.1278	0.1331

Table 2: Comparison of ranking based method variants on the Netflix data set ($F = 20$)

If the evaluation criterion is ARP, then support based weighting should not be applied. This result was again expectable, because negative items of the test set are drawn uniformly in the case of ARP. In further experiments with ARP, we will always use the setting s.w. = no.

The results for Recall are diversified: support based weighting helps significantly in the case of RankSGD, but has negative effect for RankSGD2 and RankALS. In further experiments with Recall, we will always use the setting s.w. = no for RankSGD and RankALS, and s.w. = yes for RankSGD.

We also mention for comparison that the ImplicitALS approach with $F = 20$ achieves ErrorRate = 0.1154, ARP = 0.0291, Recall = 0.3181 on the Y!Music, and ErrorRate = 0.2732, ARP = 0.0494, Recall = 0.1835 on the Netflix data set. These numbers suggest that ImplicitALS tends to be less accurate for a given evaluation metric than the best ranking based method variant, but is is less sensitive to the choice of the metric.

5.2 Accuracy versus model size

In this experiment, we ran all methods on all datasets with different number of features. The results are shown in Figure 1–6. It can be observed that RankALS clearly outperforms other methods on the Y!Music data set in terms of ErrorRate and ARP independently from the number of features, while it is on par with SGD-based method according to Recall. On the Netflix data set, RankALS shows similar behavior, with the exception of $F = 50$ and ARP. Interestingly, on this data set ImplicitALS outperforms other methods for Recall.

5.3 Accuracy per iteration

In this experiment, we compared the convergence speed of RankSGD and RankALS. In particular, we investigated how the ErrorRate decreases with the number of iterations on the Y!Music data set. The number of features was set to $F =$

[3] This rule could be relaxed. We apply it, because it was used in Track2 of KDD Cup 2011.

[4] According to [5], regularizing RankSGD and RankSGD2 models did not help too much in Track 2 of KDD Cup 2011.

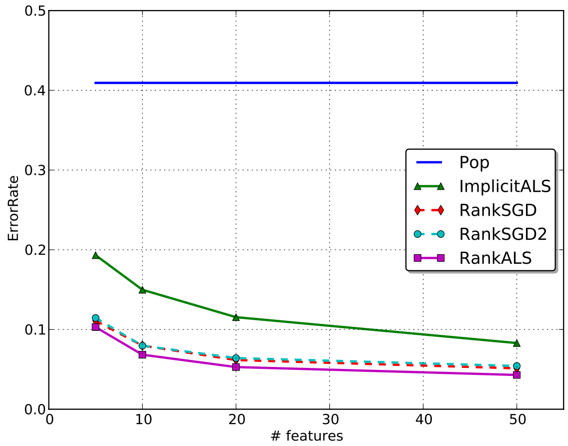

Figure 1: ErrorRate on the Y!Music data set

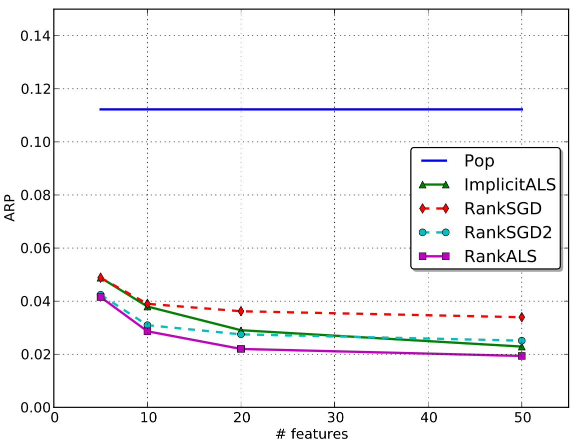

Figure 2: ARP on the Y!Music data set

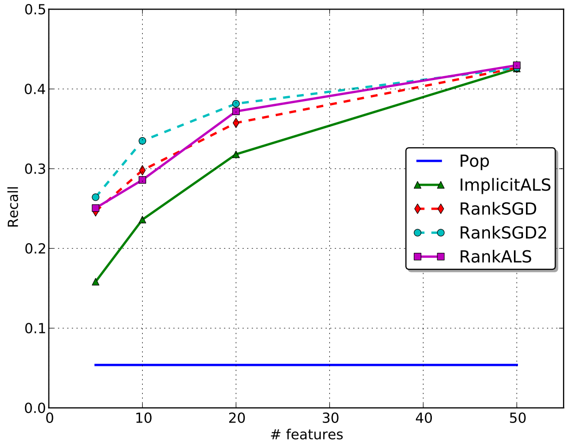

Figure 3: Recall on the Y!Music data set

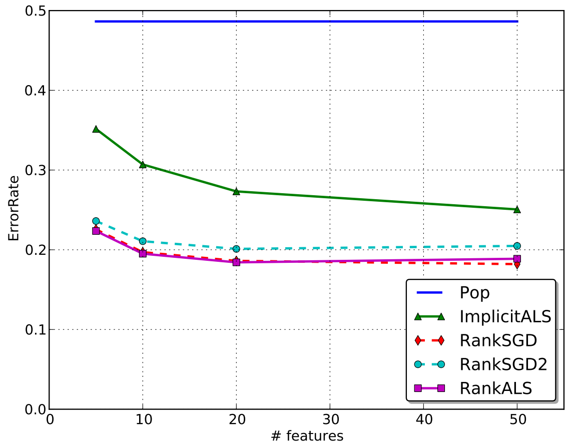

Figure 4: ErrorRate on the Netflix data set

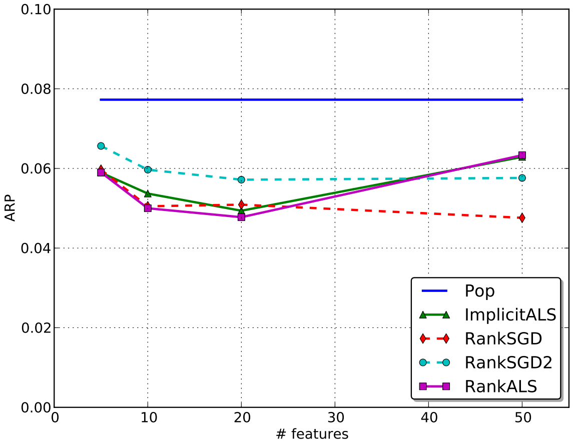

Figure 5: ARP on the Netflix data set

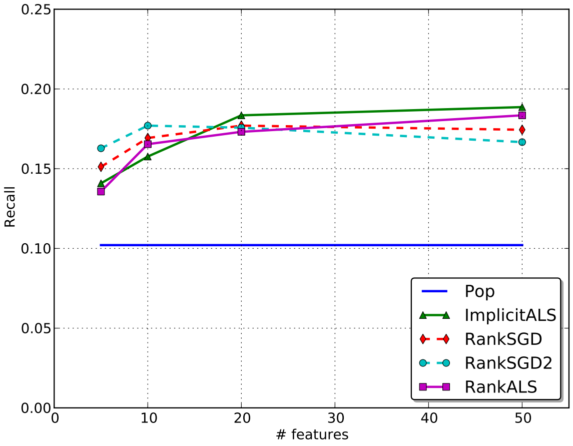

Figure 6: Recall on the Netflix data set

100 for both methods. The results are shown in Figure 7, where it can be observed that RankALS convergences faster.

5.4 Iteration time

We also compared the iteration times of the methods on the Y!Music data set. In the case of ImplicitALS and RankALS an iteration contains both a P- and a Q-step. We used one core of a Intel Xeon E5405 CPU for the measurement. The results are shown in Table 3.

As expected, RankSGD and RankSGD2 methods that use

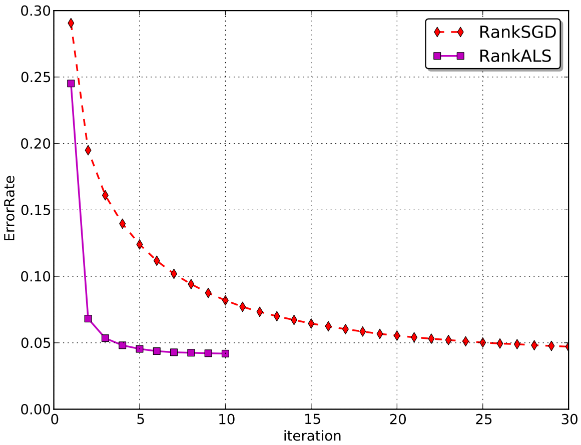

Figure 7: ErrorRate per iteration for RankSGD and RankALS on the Y!Music data set.

method	#5	#10	#20	#50	#100
ImplicitALS	82	102	152	428	1404
RankSGD	21	25	30	42	65
RankSGD2	26	32	40	61	102
RankALS	167	190	243	482	1181

Table 3: Time per iteration in seconds on the Y!Music data set (column labels refer to the number of features, F)

SGD and sampling are faster then ALS-based methods that optimize the original objective function. Note that SGD-based methods have the learning rate as additional parameter, which need to be adjusted appropriately; this introduces another optimization round into the parameter setting. We also mention that our RankSGD and RankSGD2 implementations were more optimized than our ImplicitALS and RankALS implementations.

6. CONCLUSION

Learning user–item preference from implicit feedback is a key problem of recommender systems research. One approach for handling implicit feedback is to minimize a ranking objective function instead of the conventional prediction mean squared error. Ranking objective functions are typically expensive to optimize, and this difficulty is usually overcome by sampling the objective function.

In this paper, we proposed a computationally efficient ranking based method RankALS that optimizes the original objective function, without sampling. RankALS was inspired by the prediction based method ImplicitALS [4]. The key components of RankALS are a matrix factorization model, a ranking based objective function, and an alternating least squares optimizer. Speedup is achieved by the mathematical simplification of the objective function's derivative.

As we have shown, RankALS outperforms the sampling based methods used in the comparison in terms of the ErrorRate and ARP in most cases, and is also on par with SGD-based methods in terms of Recall. Another advantage of the method is its robustness: it is not sensitive to the order of training examples (unlike SGD-based models) and with the fewer parameters (no learning rate) its adaptation to a new data set can be done with less experiments.

There are various ways to extend the RankALS method. One possible direction is to make RankALS faster by replacing the exact least squares solver by an approximate one [8]. Furthermore, its modeling capability can be improved by adding context-awareness and introducing time-dependent or user/item metadata-dependent terms into the prediction formula.

7. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17:734–749, 2005.

[2] J. Bennett and S. Lanning. The Netflix Prize. In *Proc. of KDD Cup Workshop at 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '07, pages 3–6, San Jose, CA, USA, 2007.

[3] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer. The Yahoo! Music dataset and KDD-Cup'11. In *Proc. of KDD Cup 2011*, 2011.

[4] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proc. of 8th IEEE Int. Conf. on Data Mining*, ICDM '08, pages 263–272, Pisa, Italy, 2008.

[5] M. Jahrer and A. Töscher. Collaborative filtering ensemble for ranking. In *Proc. of KDD Cup Workshop at 17th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '11, San Diego, CA, USA, 2011.

[6] T. G. McKenzie, C. S. Ferng, Y. N. Chen, C. L. Li, C. H. Tsai, K. W. Wu, Y. H. Chang, C. Y. Li, W. S. Lin, S. H. Yu, C. Y. Lin, P. W. Wang, C. M. Ni, W. L. Su, T. T. Kuo, C. T. Tsai, P. L. Chen, R. B. Chiu, K. C. Chou, Y. C. Chou, C. C. Wang, C. H. Wu, H. T. Lin, C. J. Lin, and S. D. Lin. Novel models and ensemble techniques to discriminate favorite items from unrated ones for personalized music recommendation. In *Proc. of KDD Cup Workshop at 17th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '11, San Diego, CA, USA, 2011.

[7] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. M. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *Proc. of 8th IEEE Int. Conf. on Data Mining*, ICDM '08, pages 502–511, Pisa, Italy, 2008.

[8] I. Pilászy, D. Zibriczky, and D. Tikk. Fast ALS-based matrix factorization for explicit and implicit feedback datasets. In *Proc. of the 4th ACM conference on Recommender Systems*, RecSys '10, pages 71–78, Barcelona, Spain, 2010.

[9] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proc. of the 25th Conf. on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461, 2009.

[10] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intellegence*, 2009, 2009. Article ID: 421425, 19 pages.

Erratum

1. The formula for $\bar{\bar{p}}_2$ in Algorithm 2 is errorneous. Correctly it is $\bar{\bar{p}}_2 = \sum_{u \in \mathcal{U}} p_u p_u^T \bar{q}$ as it is written in the left column of page 4.

2. There is an error in expressing the q_i that sets $\partial f_R / \partial q_i$ to zero. The quantities $\tilde{q} = \sum_{j \in \mathcal{I}} s_j q_j$ and $\bar{q} = \sum_{j \in \mathcal{I}} c_{uj} q_j$ are incorrectly treated as if they were independent of q_i and thus they are incorporated into y in the Q-step of Algorithm 2. To fix this issue, $(\bar{A} s_i + \sum_{u \in \mathcal{U}} p_u p_u^T c_{ui}) q_i$ should be subtracted from y and $(\bar{A} s_i + \sum_{u \in \mathcal{U}} p_u p_u^T c_{ui})$ should be subtracted from M before setting q_i to $M^{-1} y$.

   Interestingly, this error did not break the algorithm down, probably because the dependence of $\tilde{q}$ and $\bar{q}$ on q_i is weak. I believe that fixing this issue would give a small accuracy gain.