Applications of the Conjugate Gradient Method for Implicit Feedback Collaborative Filtering

(draft version)

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

István Pilászy
Gravity Research and
Development Ltd.
Expo tér 5–7.
Budapest, Hungary
pila@gravityrd.com

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

ABSTRACT

The need for solving weighted ridge regression (WRR) problems arises in a number of collaborative filtering (CF) algorithms. Often, there is not enough time to calculate the exact solution of the WRR problem, or it is not required. The conjugate gradient (CG) method is a state-of-the-art approach for the approximate solution of WRR problems. In this paper, we investigate some applications of the CG method for new and existing implicit feedback CF models. We demonstrate through experiments on the Netflix dataset that CG can be an efficient tool for training implicit feedback CF models.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation

Keywords

conjugate gradient method, collaborative filtering

1. INTRODUCTION

Recommender systems suggest personalized recommendations on items to users. Collaborative filtering (CF) approaches exploit user event history related to items to model user preferences. Though the CF literature discusses overwhelmingly the explicit feedback case [8], when users typically *rate* items, in many application domains recommender systems have to infer the user preference from implicit user feedback [1, 2], such as the presence or absence of purchase, view, rent or search events.

In this paper, we investigate thus the implicit feedback CF problem. Many CF algorithms require to solve weighted ridge regression (WRR) problems. Often, there is not enough time to calculate the exact solution, therefore approximative WRR solvers are necessary. The paper compares two approximate approaches: coordinate descent (CD), and conjugate gradient (CG) methods. While the former was already used in CF applications (see e.g. [6]), to our best knowledge, CG has not yet been applied.

We show that CG has advantages over CD: (1) CG tends to have shorter iteration time and this allows for various running time–accuracy trade-offs [6]. (2) CG also performs substantially better on large, sparse WRR problems that appear e.g. in the implicit version of Paterek's NSVD1 approach.

1.1 Notation

The following notation will be used in the paper. N and M denote the number of users and items. We use $u \in \{1, \ldots, N\}$ as index for users, and $i, j \in \{1, \ldots, M\}$ as indices for items. The rating of user u on item i is r_{ui}, and its prediction is $\hat{r}_{ui}$. Ratings r_{ui} are arranged in the matrix R; $\mathcal{T}$ denotes the set of (u, i) indexes of R where (a) a rating is provided (for explicit feedback), (b) a positive feedback is provided (for implicit feedback). $\mathcal{T}^{\pm}$ denotes each (u, i) pair of R, i.e. $|\mathcal{T}^{\pm}| = N \cdot M$. $r_u \in \mathbb{R}^{M \times 1}$ denotes the u-th row of R. $\bar{r}_i \in \mathbb{R}^{N \times 1}$ denotes the i-th column of R. $\mathcal{I} = \{1, \ldots, M\}$ and $\mathcal{U} = \{1, \ldots, N\}$ denote the set of items and users respectively. $\mathcal{I}_u = \{i : (u, i) \in \mathcal{T}\}$ denotes the set of items for which a positive feedback is provided by user u. $n_u = |\mathcal{I}_u|$ denotes the number of positive feedbacks of user u. We assume that users give feedback for items at most once. Let n_i denote the number of positive feedbacks for item i.

2. WEIGHTED RIDGE REGRESSION

Here we briefly revisit WRR and some common solution techniques for it. In the WRR problem we have a dataset (x_1, y_1, c_1), …, (x_n, y_n, c_n), where $x_i \in \mathbb{R}^d$ is the i-th input, y_i is its corresponding target, and $c_i \in \mathbb{R}$ is the importance value associated with the i-th example. We are also given the nonnegative regularization coefficients $\lambda_1, \ldots, \lambda_d \in \mathbb{R}$.

The inputs are arranged in the matrix $X \in \mathbb{R}^{n \times d}$, and the targets in the vector $y \in \mathbb{R}^n$. Let us denote the j-th column of X by $\bar{x}_j$. The importance values and the

regularization coefficients define the diagonal matrices $C = \mathrm{diag}(c_1, \ldots, c_n) \in \mathbb{R}^{n \times n}$ and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_d) \in \mathbb{R}^{d \times d}$.

WRR seeks the minimum of the error function

$$g(w) = \sum_{i=1}^{n} \left(c_i (x_i^T w - y_i)^2 + \frac{1}{n} \sum_{j=1}^{d} \lambda_j w_j^2 \right)$$
$$= (Xw - y)^T C (Xw - y) + \frac{1}{2} w^T \Lambda w.$$

Let us introduce the notation $A = X^T C X + \Lambda$ and $b = X^T C y$. Note that A is symmetric positive semidefinite, and if every regularization coefficient λ_j is positive, then it is also positive definite. Using A and b the error function can be written as $g(w) = w^T A w - 2 w^T b + y^T C y$. The gradient of g is $\nabla g(w) = 2Aw - 2b$, and the optimal w is

$$w^* = A^{-1} b.$$

The straightforward way of calculating w^* is constructing the matrix A and the vector b, and then solving $Aw = b$ via Cholesky decomposition. The construction of A requires $O(N_X d)$ operations, where N_X is the number of nonzero elements in X. Note that A is dense in general, even if X is sparse. The total time and space requirement of the Cholesky decomposition based approach for solving WRR is $O(N_X d + d^3)$ and $O(N_X + d^2)$.

2.1 Approximate WRR

Some approaches to perform faster, approximate WRR are the following:

- Coordinate descent: In each step, g is minimized in one coordinate of w.
- Gradient descent: In the k-th step, g is minimized along the direction of the negative gradient: $w_{k+1} = \arg\min_\alpha w_k - \alpha \nabla g(w_k)$.
- Conjugate gradient method: In each step, g is minimized along a search direction. The search directions are chosen so that w^* is reached in at most d steps.

2.2 Preconditioning

Preconditioning is an important technique in iterative methods for solving systems of linear equations. The idea of preconditioning is to solve $(M^{-1}A)w = (M^{-1}b)$ instead of $Aw = b$, where the nonsingular matrix $M \in R^{d \times d}$ is called the preconditioner. The role of the preconditioner is to reduce the condition number of the system at a relatively low cost, and therefore accelerate the convergence of the iterative solver.

In order to define two simple preconditioners for symmetric systems, let us partition A into three parts as $A = A_L + A_D + A_L^T$, where A_L is a lower triangular matrix containing the lower triangular part of A, and A_D is a diagonal matrix containing the diagonal part of A. In the case of the Jacobi preconditioning, $M = A_D$. In the case of symmetric successive over-relaxation (SSOR) preconditioning, $M = (A_D + A_L) A_D^{-1} (A_D + A_L^T)$.

2.3 Algorithms for approximate WRR

2.3.1 Coordinate descent

Coordinate descent (CD) is a natural technique to perform approximate minimization. The efficiency of CD based WRR solvers for certain collaborative filtering models was shown in [6]. The pseudocode of CD for WRR is as follows:

Input: $X \in \mathbb{R}^{n \times d}$, $y \in \mathbb{R}^n$, $C \in \mathbb{R}^{n \times n}$, $\Lambda \in \mathbb{R}^{d \times d}$, $w_0 \in \mathbb{R}^d$, $E \in \mathbb{N}$
Output: $w \in \mathbb{R}^d$
$w \leftarrow w_0, \quad r \leftarrow y - Xw$
for $k \leftarrow 1, \ldots, E$ **do**
  if $\|r\|$ *is small enough* **then return** w;
  for $j \leftarrow 1, \ldots, d$ **do**
    $\Delta w_j \leftarrow (\bar{x}_j^T C r - \lambda_j w_j) / (\bar{x}_j^T C \bar{x}_j + \lambda_j)$
    $w_j \leftarrow w_j + \Delta w_j, \quad r \leftarrow r - \Delta w_j \bar{x}_j$
  end
end

Algorithm 1: Coordinate descent method for WRR

The method can be applied both for dense and sparse X matrices. The usual choice for the initial solution is $w_0 = 0$. An advantage of starting from the zero vector is that no matrix-vector multiplication is needed for determining the initial r.

The time requirement of the algorithm is $O(N_X E)$, where E is the number of iterations. The space requirement of the algorithm is $O(N_X)$. Note that our version of CD is slightly more efficient than the one described in [6], since it needs only 2 iterations over the examples in the innermost loop instead of 3.

2.3.2 Conjugate gradient method

The conjugate gradient (CG) method [3] is a long known yet state-of-the-art iterative method for solving the linear system $Aw = b$, where A is symmetric positive definite. We recall that in the case of WRR, A and b are obtained as $A = X^T C X + \Lambda$ and $b = X^T C y$. The pseudocode of preconditioned CG for solving $Aw = b$ is the following:

Input: $A, M \in \mathbb{R}^{d \times d}$, $b, w_0 \in \mathbb{R}^{d \times 1}$, $E \in \mathbb{N}$
Output: $w \in \mathbb{R}^d$
$w \leftarrow w_0, \quad r \leftarrow b - Aw, \quad z \leftarrow M^{-1} r, \quad p \leftarrow z$
for $k \leftarrow 1, \ldots, E$ **do**
  if $\|r\|$ *is small enough* **then return** w;
  $\gamma \leftarrow r^T z, \quad \alpha \leftarrow \gamma / (p^T A p)$
  $x \leftarrow x + \alpha r, \quad r \leftarrow r - \alpha A p, \quad z \leftarrow M^{-1} r$
  $\beta \leftarrow \gamma / (r^T z), \quad p \leftarrow z + \beta p$
end

Algorithm 2: Preconditioned conjugate gradient method for solving $Aw = b$.

Note that matrix A is never used directly. It is enough if we can calculate the product of A with an arbitrary vector.

The computationally most expensive step of the algorithm is the matrix-vector multiplication Ap, that has to be evaluated once in each iteration. In the case of WRR, the product Ap can be calculated in $O(N_X)$ time as $Ap = X^T(C(Xp)) + \Lambda p$. The time requirement of the algorithm is $O(N_X E)$, and the space requirement is $O(N_X)$. Again, the cost of allowing an arbitrary w_0 instead of the zero vector is an extra matrix-vector multiplication.

An important property of the CG method is that the search directions $p_0, \ldots, p_E$ form an A-conjugate system, meaning that $p_j^T A p_k = 0$, if $j \neq k$. This property implies that the solution of the k-th iteration minimizes $f(w) = w^T A w - 2 w^T b$ in the affine subspace $w_0 + \mathrm{span}\{p_0, p_1, \ldots, p_{k-1}\}$. As a consequence, the CG method finds the exact solution w^* in at most d iterations.

For comparison purposes, we also define here the gradient

(G) method that is considered a less effective WRR solver than CG. It can be obtained from CG, by using $\beta = 0$ in every iteration.

3. WRR BASED CF METHODS

3.1 Matrix factorization

Matrix factorization (MF) approaches for CF approximate the rating matrix R as the product of two lower rank matrices:

$$R \approx PQ^T,$$

where $P \in \mathbb{R}^{N\times K}$ is the user feature matrix, $Q \in \mathbb{R}^{M\times K}$ is the item feature matrix, K is the number of features that is a predefined constant, and the approximation is only performed at $(u,i) \in \mathcal{T}$ positions. If the last column of P and the penult column of Q are constant, then we get an MF model with user and item bias [5]. A training algorithm for the plain MF model can be easily modified to handle this case, therefore we do not introduce separate parameters for biases.

The prediction formula for user u and item i is

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

where $p_u \in \mathbb{R}^K$ the u-th row of P and $q_i \in \mathbb{R}^K$ is the i-th row of Q.

The first implicit feedback version of the MF model was introduced by Hu et al. [4]. Their method is a weighted least squares approach, meaning that a squared error is defined for each user-item pair, and the weighted sum of squred errors is minimized. Another solution to obtain implicit feedback MF is to formulate training as a ranking optimization problem [7]. The second approach contains fewer assumptions than the first, however, it also has a disadvantage: ranking optimization is in general harder than solving least squares.

This paper follows the path of Hu et al. and defines the following least squares cost function for the implicit MF model:

$$g(P,Q) = \sum_{(u,i)\in\mathcal{T}} \left(c_{ui}(\hat{r}_{ui} - r_{ui})^2 + \lambda_P\|p_u\|^2 + \lambda_Q\|q_i\|^2\right).$$

Here c_{ui} denotes the importance weight associated with the (u,i)-th element of the matrix. Hu et al. assume that c_{ui} and r_{ui} values are constant, except at the positions of positive implicit feedback. Let c_0 and r_0 denote these constant values.

Since g is a nonconvex function of $(N+M)K$ variables, its exact minimization is difficult. Hu et al. proposed an alternating least squares approach for training the implicit MF model efficiently. The key idea of their approach is to alternate between the minimization of g in P and Q. In a naive implementation, one would minimize P as follows: $p_u \leftarrow (Q^TC_uQ + n_u\lambda_P I)^{-1}(Q^TC_ur_u)$, where $C_u \in \mathbb{R}^{M\times M}$ is a diagonal matrix, generated from c_{ui} values of user u.

Hu et al. propose the following speed-up: Let $u = 0$ denote a user who has no positive implicit feedback. we can precompute values Q^TC_0Q and $Q^TC_0r_0$ for this user. For an arbitrary user u, the number of differences between C_0 and C_u is small, since users give feedback only on a small subset of items. We can efficiently compute Q^TC_uQ by $Q^TC_0Q + Q^T(C_u - C_0)Q$, i.e. we need to consider only the differences between the two users.

3.2 NSVD1

Paterek introduced an interesting asymmetric factor model for explicit feedback, called NSVD1 [5]. Here we adapt this model for implicit feedback. The free parameters of this model are two sets of item feature vectors ($q_1,\dots,q_M$, $w_1,\dots,w_M \in \mathbb{R}^K$).[1] The prediction of the (u,i)-th rating is calculated as the following:

$$\hat{r}_{ui} = p_u^T q_i,$$
$$\text{where: } p_u = s_u \sum_{j\in\mathcal{I}_u} w_j, \quad s_u = (n_u+1)^{-\frac{1}{2}}$$

The error function associated with the model is the same as in the case of MF. The difference is that now P is not a parameter but it is the function of W and the training data. Again, the error function is nonconvex which makes its exact minimization difficult. Here we present an approximate solution which is based on IALS.

Let $W \in \mathbb{R}^{M\times K}$ denote the matrix of w_j vectors. Let $B \in \mathbb{R}^{N\times M}$ a matrix with elements $[B]_{ui} = r_{ui}s_u$. With this notation, $P = BW$. Note that B is a sparse matrix. During the optimization we have three sets of parameters: P, Q and W. The proposed algorithm is the following:

- For $e = 1,\dots,I$:
  1. Q-step: For $i \in \mathcal{I}$: $q_i \leftarrow \text{WRR}(P, \bar{r}_i, \bar{c}_i, n_i\lambda_Q I)$.
  2. P-step: For $u \in \mathcal{U}$: $p_u \leftarrow \text{WRR}(Q, r_u, c_u, n_u\lambda_P I)$.
  3. W-step: For $k \in \mathcal{K}$: $w_k \leftarrow \text{WRR}(B, \bar{p}_k, 1, \lambda_W I)$.
  4. For $u \in \mathcal{U}$: $p_u \leftarrow p_u = s_u \sum_{j\in\mathcal{I}_u} w_j$

The first two steps are the steps of IALS [4]. The first step optimizes Q in the equation $R \approx PQ^T$. Here $\bar{r}_i \in \mathbb{R}^{N\times 1}$ and $r_u \in \mathbb{R}^{M\times 1}$ are the i-th column and the u-th row of R, respectively. The vectors $c_i \in \mathbb{R}^{N\times 1}$ and $c_u \in \mathbb{R}^{M\times 1}$ are generated from the respective c_{ui} values. The WRR algorithm can be any weighted ridge regression solver, and vectors r_i and c_i need not be computed from scratch between consecutive invocations, because only a small fraction of them changes.

The second step is very similar to the first step, except that it optimizes only P. The third step is different: it optimizes only W via the $P \approx BW$ approximation. Here $\bar{p}_k \in \mathbb{R}^{N\times 1}$ is the k-th column of P. Here, the dimension of the input is M, which is often very large (a typical value is 20 000), however, the input is sparse. In this step, the weights of weighted ridge regression (the third argument of WRR) are constant 1. Note that W-step aims at a better approximation of P, not R.

4. EXPERIMENTS

We compared our proposed CG based training methods with other approaches on an implicit version of the well-known Netflix dataset. The implicit rating matrix was defined by assigning 1 with confidence 100 to user-item pairs with a Netflix rating value 5, and 0 with confidence 1 to other user-item pairs. We used the Netflix training set minus the Netflix probe set for training, and the Netflix probe set for testing.

The WRR solvers included in the experiments were the coordinate descent (CD), the gradient (G), the conjugate gradient (CG), and the Cholesky-factorization based method. The evaluation metrics were:

[1] Again, we do not introduce separate parameters for biases.

Table 1: Recall values for implicit MF using different preconditioners ($K = 20$)

#	None	Jacobi	SSOR
G	0.3600	0.4416	0.4440
CG	0.4407	0.4444	0.4438

- **TrTime**: CPU time in seconds needed for one epoch of training.
- **Recall**$_{1\%}$: For each user-item pair (u, i) of the test set, we calculate the predicted rating of user u for every item, and rank the items based on these values. Recall$_{1\%}$ is the the relative frequency of item i being in the top 1 % of the ranking.

In the first experiment we compared different preconditioners on an implicit MF model with $K = 20$ factors. The measured recall values are shown in Table 1. The results are much better with preconditioning than without it. Since SSOR needed about twice as much time per iteration as Jacobi, and they achieved similar results, we opted for Jacobi preconditioning.

Next, we compared different WRR solvers for training the implicit MF model and the implicit NSVD1 model. The metaparameters of the model were set to $\lambda_P = \lambda_Q = \lambda_B = 0.05$. The number of internal iterations in the WRR solvers was $E = 2$, while the number of external iteration steps, in general, the number of P-steps, was $I = 10$.

Table 2: TrTime values for implicit MF (sec)

# features	CD	G	CG	Chol
5	91.7	**84.0**	90.4	93.7
10	107.8	**94.1**	97.0	117.4
20	133.0	110.5	**110.3**	176.1
50	233.2	164.0	**149.6**	493.8

Table 3: Recall$_{1\%}$ values for implicit MF

# features	CD	G	CG	Chol
5	0.3756	0.3775	**0.3797**	0.3794
10	0.4249	0.4222	0.4249	**0.4257**
20	0.4435	0.4416	0.4444	**0.4452**
50	0.4543	0.4510	**0.4557**	0.4536

The results for implicit MF are summarized in Tables 2 and 3. It can be seen that the G method had similar TrTime as CG, but it was worse in terms of recall. The CD method produced similar recall to CG, but its TrTime was higher. The Cholesky method was accurate, but it was the slowest approach.

The results for implicit NSVD1 are tabulated in Tables 4 and 5. The W-step of NSVD1 training was done by a CG based WRR solver, run for 20 iterations. We also tried to compute the W-step with a CD based solver, but it produced a much worse reconstruction of P.

The first row of the tables now indicates the method applied for computing the P-step and the Q-step. The TrTime value of the different approaches was quite similar. This is because here the computationally dominant part of training is the W-step.

5. CONCLUSION

Collaborative filtering algorithms often require the solution of weighted ridge regression problems. Here we proposed the use of the conjugate gradient method as an approximate WRR solver for such problems. We showed that with increasing factor value, the training time of the CG is typically smaller than of other WRR solvers, while its accuracy measured in terms of recall is on par or even better. We demonstrated our results using the Netflix Prize dataset with implicit feedback data with matrix factorization (ALS) and implicit NSVD1.

Table 4: TrTime values for implicit NSVD1 (sec)

# features	CD	G	CG
5	126	**119**	122
10	**202**	203	204
20	377	**353**	361
50	860	801	**794**

Table 5: Recall$_{1\%}$ values for implicit NSVD1

# features	CD	G	CG
5	0.3704	0.3721	**0.3729**
10	0.3978	**0.4037**	0.4028
20	0.4141	0.4207	**0.4213**
50	0.4333	0.4371	**0.4463**

6. REFERENCES

[1] K. Ali and W. van Stam. TiVo: making show recommendations using a distributed collaborative filtering architecture. In *Proc. of the 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD '04, pages 394–401, New York, NY, USA, 2004. ACM.

[2] L. Baltrunas and X. Amatriain. Towards time-dependant recommendation based on implicit feedback. In *Proc. of the RecSys 2009 Workshop on Context-aware Recommender Systems*, 2009.

[3] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

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

[5] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proc. of KDD Cup Workshop at SIGKDD-07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 39–42, San Jose, California, USA, 2007.

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

[7] 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.

[8] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656, 2009.