Large-scale Parallel Collaborative Filtering for the Netflix Prize

Yunhong Zhou, Dennis Wilkinson, Robert Schreiber and Rong Pan

TALK LEAD: PRESTON ENGSTROM

FACILITATORS: SERENA MCDONNELL, OMAR NADA

Agenda

Historical Background

Problem Formulation

Alternating Least Squares For Collaborative Filtering

Results

Discussion

The Netflix Prize

In 2006, Netflix offered 1 million dollars for an algorithm that could beat their algorithm, “Cinematch”, in terms of RMSE by 10%

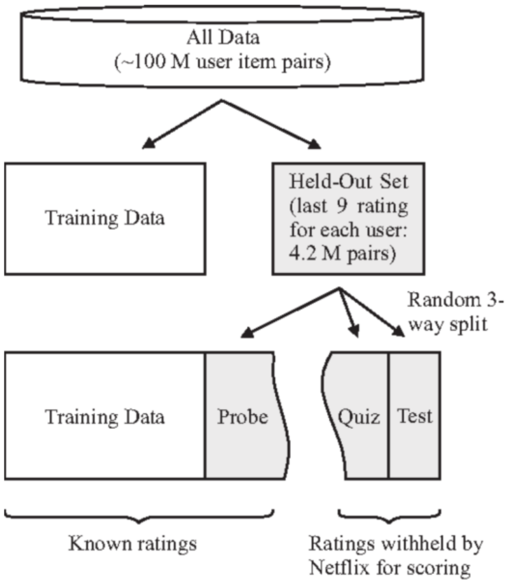
100MM ratings

(User, Item, Date, Rating)

Training data and holdout was split by date

- Training data: 1996 – 2005
- Test data: 2006

NETFLIX



The Netflix Prize: Solutions

Leaderboard

Showing Test Score. Click here to show quiz score

Display top 20 **leaders.**

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43

Recommendation Systems: Collaborative Filtering

Generate recommendations of relevant items from aggregate behavior/tastes over a large number of users

Contrasted against Content Filtering, which uses attributes (text, tags, user age, demographics) to find related items and users

Notation

n_u : Number of users

n_m : Number of items

R : $n_u \times n_m$ interaction matrix

r_{ij} : Rating provided by user i for item j

I_i : Set of movies user i has rated

n_{u_i} : Cardinality of I_i

I_j : Set of users who rated movie j

n_{m_j} : Cardinality of I_j

Problem Formulation

Given a set of users and items, estimate how a user would rate an item they have had no interaction with

Available information:

- List of (User ID, Item ID, Rating,.....)

Low Rank Approximation

- Populate a matrix with the known interactions, factor to two smaller matrices, and use them to generate unknown interactions

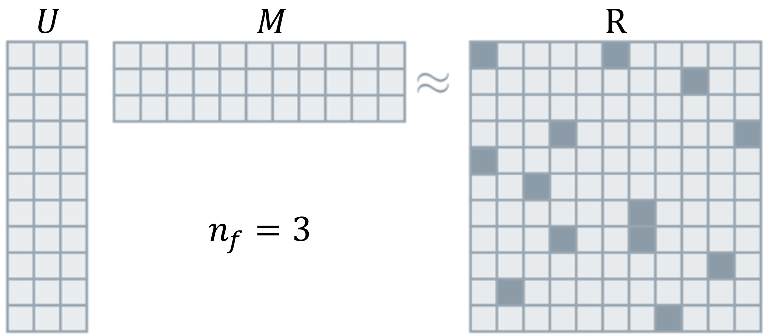
Problem Formulation

Let $U = [\boldsymbol{u_i}]$ be the user feature matrix, where $\boldsymbol{u_i} \subseteq \mathbb{R}^{\boldsymbol{n_f}}$ for all $i = 1 \ldots n_u$

Let $M = [\boldsymbol{m_j}]$ be the user feature matrix, where $\boldsymbol{m_j} \subseteq \mathbb{R}^{\boldsymbol{n_f}}$ for all $i = 1 \ldots n_m$

n_f is the dimension of the feature space. This gives $(n_u + n_m) \times n_f$ parameters to be learned

If the full interaction matrix R is known, and n_f is sufficiently large, we could expect that $r_{ij} =< \boldsymbol{u_i}, \boldsymbol{m_j} >, \forall i, j$



Problem Formulation

Mean square loss function for a single rating:

$$\mathcal{L}^2(r, \boldsymbol{u}, \boldsymbol{m}) = (r - < \boldsymbol{u}, \boldsymbol{m} >)^{\mathbf{2}}$$

Total loss for a given U and M, and the set of known ratings I:

$$\mathcal{L}^{emp}(R, U, M) = \frac{1}{n} \sum_{(i,j) \in I} \mathcal{L}^2(r_{ij} \boldsymbol{u_i}, \boldsymbol{m_j})$$

Finding a good low rank approximation:

$$(U, M) = \arg\min_{U,M} \mathcal{L}^{emp}\,(R, U, M)$$

Many parameters and few ratings make overfitting very likely, so a regularization term should be introduced:

$$\mathcal{L}_{\lambda}^{reg}(R, U, M) = \mathcal{L}^{emp}(R, U, M) + \lambda(\|U\Gamma_U\|^2 + \|M\Gamma_M\|^2)$$

Singular Value Decomposition

Singular Value Decomposition (SVD) is a common method for approximating a matrix R by the product of two rank k matrices $\tilde{R} = U^T \times M$

Because the algorithm operates over the entire matrix, minimizing the Frobenious norm $R - \tilde{R}$, standard SVD can fail to find U and M

Alternating Least Squares

Alternating Least Squares is one method used to find U and M, only considering known ratings

By treating each update of U and M as a least squares problem, we are able to update one matrix by fixing the other, and using the known entries of R

ALS with Weighted-λ-Regularization

Step 1: Set $\boldsymbol{M}$ to be a $\boldsymbol{n_f} \times \boldsymbol{n_u}$, assign the average rating for each movie to the first row

Step 2: Fix $\boldsymbol{M}$ solve $\boldsymbol{U}$ by minimizing the objective function

Step 3: Fix $\boldsymbol{U}$ solve M by minimizing the objective function

Step 4: Repeat steps 2 and 3 until a set number of iterations, or an improvement in error of less than 0.0001 on the probe dataset occurs

ALS with Weighted-λ-Regularization

The selected lost function uses Tikhonov regularization to penalize large parameters. This scheme was found to never overfit on test data when the number of features n_f or number of iterations was increased.

$$f(U,M) = \sum_{(i,j)\epsilon I} \left(r_{ij} - \boldsymbol{u_i^T m_j}\right)^2 + \lambda\left(\sum_i n_{u_i}\|\boldsymbol{u}_i\|^2 + \sum_j n_{m_j}\|\boldsymbol{m}_j\|^2\right)$$

λ : Regularization weight

n_{u_i} : Cardinality of I_i

n_{m_j} : Cardinality of I_j

Solving for U given M

$$\frac{1}{2}\frac{\partial f}{\partial u_{ki}} = 0, \qquad \forall i, k$$

$$\Rightarrow \sum_{j \in I_i} (\boldsymbol{u_i^T m_j} - r_{ij})\, m_{kj} + \lambda n_{u_i} u_{ki} = 0, \qquad \forall i, k$$

$$\Rightarrow \sum_{j \in I_i} m_{kj} \boldsymbol{m}_j^T \boldsymbol{u}_i + \lambda n_{u_i} u_{ki} = \sum_{j \in I_i} m_{kj}\, r_{ij}, \qquad \forall i, k$$

$$\Rightarrow \left(M_{I_i} M_{I_i}^T + \lambda n_{u_i} E\right) \boldsymbol{u}_i = M_{I_i} R^T(i, I_i), \qquad \forall i$$

$$\Rightarrow \boldsymbol{u}_i = A_i^{-1} V_j, \qquad \forall i$$

Update Procedure (Single Machine)

```python
def update_U(M, U, lmbda):
    lamI = lmbda * np.identity(nf)
    lU = np.zeros((nf, U.shape[1]))
    for i in range(U.shape[1]):
        movies_user_rated = csr[i,:].nonzero()[1]
        Mu = M[:, movies_user_rated]
        vector = Mu * csr[i, movies_user_rated].todense().T
        matrix = np.matmul(Mu,Mu.T) + len(csr[i, :].data) * lamI
        if np.linalg.lstsq(matrix, vector)[0].sum() == 0:
    return lU

def update_M(M, U, lmbda):
    lamI = lmbda * np.identity(nf)
    lM = np.zeros((nf, M.shape[1]))
    for i in range(M.shape[1]):
        users_who_rated = csc[:, i].nonzero()[0]
        Um = U[:, users_who_rated]
        vector = Um * csc[users_who_rated, i].todense()
        matrix = np.matmul(Um, Um.T) + len(csc[:, i].data) * lamI
        lM[:, i] = list(np.linalg.lstsq(matrix, vector)[0])
    return lM
```

Parallelization

ALS-WR can be parallelized by distributing the updates of U and M over multiple computers

When updating U, each computer need only hold a portion of U, and the corresponding portion of R. However, a complete copy of M is needed locally on each computer

Once all computers have updated their local partition of U, the results can be gathered, and distributed out to allow the following update of M

Post Processing

Global mean shift:

- Given a prediction P, if the mean of P is not equal to the mean of the test dataset, all predictions are shifted by a constant $\tau = mean(test) - mean(P)$
- This is shown to strictly reduce RMSE

Linear combinations of predictors:

- Given two predictors P_0 and P_1, a new family of predictors $P_x = (1 - x)P_0 + xP_1$ can be found, and x^* determined by minimizing $RMSE(P_x)$
- This yields a predictor P_{x^*}, which is at least as good as P_0 or P_1

2 Minute Break

Hyperparameter Selection : λ

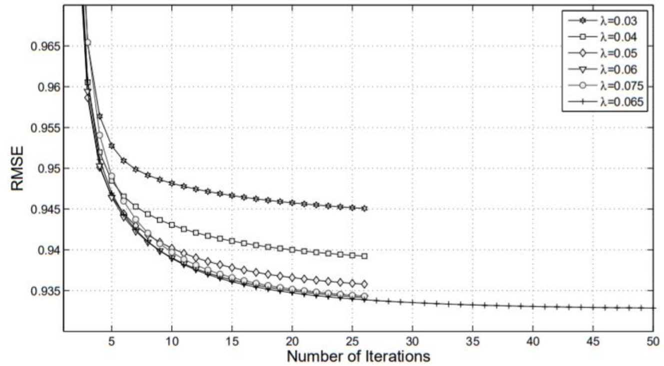


Fig. 1. Comparisons of different λ values for ALS-WR with $n_f = 8$. The best performer with 25 rounds is $\lambda = 0.065$. For this fixed λ, after 50 rounds, the RMSE score still improves but only less than 0.1 bps for each iteration afterwards.

Hyperparameter Selection : n_f

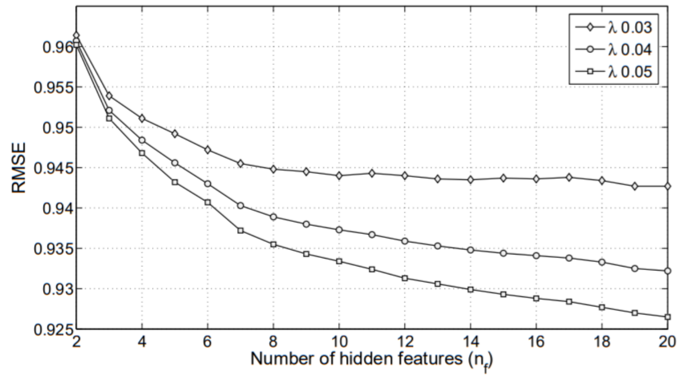


Fig. 2. Performance of ALS-WR with fixed λ and varying n_f.

Experimental Results

Where does ALS-WR stack up against the top Netflix prize winners?

The best RMSE score is an improvement of 5.56% over CineMatch

Rank	Team Name	Best Test Score	% Improvement
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos			
1	BellKor's Pragmatic Chaos	0.8567	10.06
2	The Ensemble	0.8567	10.06
3	Grand Prize Team	0.8582	9.90
4	Opera Solutions and Vandelay United	0.8588	9.84
5	Vandelay Industries !	0.8591	9.81
6	PragmaticTheory	0.8594	9.77
7	BellKor in BigChaos	0.8601	9.70
8	Dace_	0.8612	9.59

n_f	λ^*	$RMSE$	$Bias\ Correction$
50	0.065	0.9114	No
150	0.065	0.9066	No
300	0.065	0.9017	Yes
400	0.065	0.9006	Yes
500	0.065	0.9000	Yes
1000	**0.065**	**0.8985**	**No**

Combining ALS-WR With Other Methods

The top performing Netflix prize contributions were all composed of ensembles of models. Knowing this, combined with apparent diminishing returns of increasing n_f and λ, ALS-WR was combined with two other common methods:

- Restricted Boltzmann Machines (RBM)
- K-nearest Neighbours (kNN)

Both methods can be parallelized in a similar method to ALS-WR, and provide a modest improvement in performance.

Using a linear blend of ALS, RBM and kNN, an RMSE of **0.8952** (a 5.91% over CineMatch) was obtained

Discussion

While RMSE is the error metric both Netflix and the researchers choose, it's not necessarily the most appropriate. What other metrics could have been useful?

It is stated that the method of regularization will stop overfitting if either the number of features or number of iterations is increased- does this mean that if both are increased, it will overfit? Is their claim that the model never overfits valid?

S