Assignment 7

You are modeling a recommender system for a lending club platform using model-based Collaborative Filtering (vs memory-based approach computing similarities between users and items to give recommendations using rating data) described at https://www.scriptedin.com/contests/view/13,

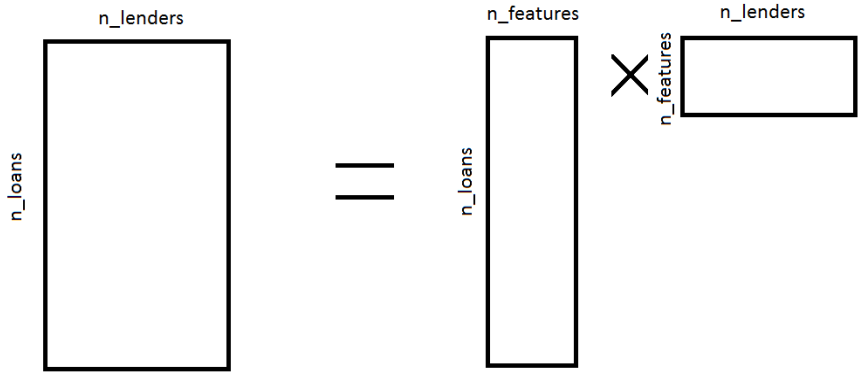
The data sets can be downloaded, and are provided on HuskyCT. A template notebook is provided to you, also posted at. https://www.scriptedin.com/notebooks/view/203.

The model-based CF is a latent factor model, more robust than the memory-based approach, and handles sparsity better. Consider a sparse rating matrix of which the elements are ratings given by lender j to loan i. The rating matrix is modeled by a matrix product of X (loan-feature matrix) and Θ (user-feature matrix) (see the figure). Each rating given by lender j to loan i is an inner product of row i in X and column j in Θ:

$$rating(i,j) = \theta^{j^T} x^i$$

The matrix form is as follows:

$$rating = X * \theta^T$$



The recommender system model helps estimate the missing ratings and recommends good loans to the lenders. The function costFunction() computes the regularized cost function as follows:

$$J\left(\theta^{(1)}\theta^{(2)} \dots \theta^{(j)} \dots \theta^{(n_{lender})}, x^{(1)}, x^{(2)}, \dots, x^{(i)}, \dots, x^{(n_{loan})}\right)$$
$$= \frac{1}{2}\left[\sum_{i=1}^{n_{loan}} \sum_{j:r(i,j)=1} \left(\theta^{(j)^T} x^{(i)} - y^{(i,j)}\right)^2\right] + \frac{\lambda}{2} \sum_{i=1}^{n_{loan}} \sum_{k=1}^{n_c} \left(x_k{}^{(i)}\right)^2$$
$$+ \frac{\lambda}{2} \sum_{j=1}^{n_{lender}} \sum_{k=1}^{n_c} \left(\theta_k{}^{(j)}\right)^2$$

In which the last 2 terms are the regularization terms. And the gradient of the cost function

$$\frac{\partial}{\partial {x_k}^{(i)}} J\left(\theta^{(1)}\theta^{(2)} \dots \theta^{(j)} \dots \theta^{(n_{lender})}, x^{(1)}, x^{(2)}, \dots, x^{(i)}, \dots, x^{(n_{loan})}\right) = \sum_{j:r(i,j)=1} \left({\theta^{(j)}}^T x^{(i)} - y^{(i,j)}\right) {\theta_k}^{(j)} + \lambda {x_k}^{(i)}$$

$$\frac{\partial}{\partial {\theta_k}^{(j)}} J\left(\theta^{(1)}\theta^{(2)} \dots \theta^{(j)} \dots \theta^{(n_{lender})}, x^{(1)}, x^{(2)}, \dots, x^{(i)}, \dots, x^{(n_{loan})}\right) = \sum_{i:r(i,j)=1} \left({\theta^{(j)}}^T x^{(i)} - y^{(i,j)}\right) {x_k}^{(i)} + \lambda {\theta_k}^{(j)}$$

In the code of **costFun**() function, X and θ are extracted from the parameter vector namely params. X has features/variables on its columns, and θ has the lenders' preference on it columns. The gradient terms are stored in the variables grad and returned to the calling function

grad = [X_grad(:); Theta_grad(:)];

The **optimizeCost()** function searches for the optimal parameter vector using gradient descent.

In the data there are 10 lenders and a large number of loans, in which many were rated by the lenders with ratings from 1-10 (in practice there are many lenders, who invest). The ones that are not rated yet have ratings of 0.

Running with top 3 recommendations for lender 1 would output something like

Top 3 recommendations for lender 1:

Predicted rating 6.8 for loan of 5000.0 for 36 months with credit card purpose at 10.7 percent interest

Predicted rating 6.6 for loan of 2500.0 for 60 months with car purpose at 15.3 percent interest

Predicted rating 6.1 for loan of 2400.0 for 36 months with small business purpose at 16.0 percent interest

2. (3 points) Write the **costFun()** function that takes the parameters to optimize, the ratings Y, the matrix r (r(i,j)=1 if there is a rating by j to i), number of lenders, number of features, lambda, returns the regularized cost.

3. (3 points) Write the **costGrad()** function that takes the parameters to optimize, the ratings Y, the matrix r (r(i,j)=1 if there is a rating by j to i), number of lenders, number of features, lambda, returns the gradient.

4. (4 points) Write the **optimize()** function that takes the parameters to optimize, the ratings Y, the matrix r (r(i,j)=1 if there is a rating by j to i), number of lenders, number of features, lambda, learning step, and the maximal number of iterations, returns the optimized parameters and the values of cost during the optimization.

Submission on Husky includes a zip file of:

a. A notebook named as “group_xxx_assignment_yyy.ipynb”

b. A Word document/article with detailed explanation

c. All the other files

Also archive the notebook and the article (only after the deadline) to the project site.