Information Retrieval HW2

June 2020

1 Algorithms

Data Preprocessing

first of all, we read the Train.csv and Validation.csv files into memory. These data are triplets of (user-id, item-id, rating). In order to make the user-ids and item-ids Pythonic we create a two mapping from Train.csv:
1. original user-id to zero based index user-id
2. original item-id to zero based index item-id

We fix all of the user-id and item-ids in the Train set according to these mappings, and also for the Validation set (if there is a new user or item in the Validation set, we simply mark it as -999).
To conclude get-data() returns a Pandas representation of the Training and Validation sets with the user-id and item-id fixed to be zero based indexes.

Simple Mean

This simple algorithm does not use any special data structure. It simply sums the ratings and count the number of ratings across all user-ids. Then divides sum/count to output a vector of means.

Linear Regression

In this algorithm, we update a vector of user biases and a vector of item biases using stochastic gradient descent with a fixed batch size of 1. We iterate the original Train data set row by row, and update the parameters according to the Appendix (without Q and P).

Simple KNN

In order to calculate the correlation matrix between item to item we need to use a better data structure than the original Train set. Obviously the original data is very sparse, therefore we use CSR matrices. The CSR format stores a sparse (m, n) matrix M in row form using three arrays (v, c, r). The arrays v and c contain the non-zero values and the column indices of those values respectively. The array r has one element per row in the matrix and encodes the index in v where the given row starts. This data structure has several advantages: faster dot products, faster slicing and less memory is used to store the data. After the Train set is converted to CSR form marked as C, we can compute the co-variance matrix $K = CC^T - \mu_C \mu_C^T$ and then the correlation matrix $CORR = K(diag(K)^{-}0.5)$. After we have the correlation matrix we use a dictionary to sort each item to its most related item using correlation descending order.
For example item-id: [(most correlated item, corr), (2nd most correlated item, corr) , ... (lest correlated item, corr)]
In order to give a prediction for a user u and item v, we find all of the items J that user u had ranked (this is easily accesses using our CSR matrix). Then we can filter our dictionary to take only the top K , positive correlated items and their correlations in a quick manner. We can get the ratings of these items quickly from the CSR matrix as well,

and we are all set to give predictions quickly and efficiently.

Baseline KNN

The same as Simple KNN, with the exception we load the pre-trained user and item biases from the Linear Regression model, and apply to our prediction. There is no change in the efficiency here.

Matrix Factorization

Very similar to Linear regression, except that we add two additional learnt parameters Q and P which represent the latent embedding of the users and the items respectively. The derivatives of each parameter are presented in the Appendix.

2 Results

The results presented here are the RMSE score achieved on the Validation set. No hyper parameter optimization was done, and the original hyper parameters provided to us were used.

Table 1:

Model	**RMSE validation**
Simple Mean	1.049
Linear Regression	0.968
Simple KNN	0.965
Baseline KNN	0.915
Matrix Factorization	0.902

3 Appendix

Objective

minimize

$$L = \sum_D (r_{m,n} - \mathrm{U_m}^{\mathrm{T}}\mathrm{V_n} - b_m - b_n - \mu)^2 + \lambda \sum_m \|U_m\|^2 + \lambda \sum_n \|V_n\|^2 + \lambda \sum_m {b_m}^2 + \lambda \sum_n {b_n}^2$$

where $r_{m,n}$ is the explicit feedback from user m on item n, D is the data set, each sample is the triplet (m, n, $r_{m,n}$). $\mathrm{U_m}$ is user m latent vector and $\mathrm{V_n}$ is item n latent vector. b_m and b_n are the biases of user m and item n respectively, and μ is the mean of $r_{m,n}$ on the entire dataset. λ is a regularization hyper-parameters.

Update Steps

error term is

$$\mathrm{e_{m,n}} = (r_{m_n} - \mathrm{U_m}^{\mathrm{T}}\mathrm{V_n} - b_m - b_n - \mu)$$

Update steps are:

$$\mathrm{U_m} = \mathrm{U_m} + \alpha(e_{m,n}\mathrm{V_n} - \lambda \mathrm{U_m})$$

$$\mathrm{V_n} = \mathrm{V_n} + \alpha(e_{m,n}\mathrm{U_m} - \lambda \mathrm{V_n})$$

$$\mathrm{b_m} = \mathrm{b_m} + \alpha(e_{m,n} - \lambda \mathrm{b_m})$$

$$\mathrm{b_n} = \mathrm{b_n} + \alpha(e_{m,n} - \lambda \mathrm{b_n})$$