Applied Machine Learning Exercise #1

Gal Lavee
March 28, 2018

In this exercise we will implement and evaluate algorithms to learn a matrix factorization model from explicit ratings data. Code can be written in the programming language of your choice, but no external libraries are allowed, with the sole exception of libraries that perform vector and matrix calculation (e.g. numpy in python or Math.Net Numerics for C#).

Part 1. The Data

Download and unzip the movie lens 1M dataset: [1] A description of the different files can be found in *README*. For our purposes we will make use of *ratings.dat* and *movies.dat*.

In this part you are required to:

1. implement a data structure to store ratings data with a constructor that takes the path to *ratings.dat* as an argument
2. implement a data structure to store the item data with a constructor that takes the path to *movies.dat* as an argument
3. implement a function to divide data randomly into training and test according to an 80-20 split. Data should be split so that each user has some data points in the train set and some datapoints in the test set.

Part 2. The Model

Recall that when using Matrix Factorization with biases, our model for prediction is given by:

$$\hat{r}_{m,n} = \mu + \mathbf{u}_m^\top \mathbf{v}_n + b_n + b_m$$

further our loss function is given by:

$$\frac{1}{2}\sum_{(m,n)\in\mathcal{I}}\left(r_{m,n} - \left(\mu + \mathbf{u}_m^\top\mathbf{v}_n + b_n + b_m\right)\right)^2 + \frac{\lambda_v}{2}\sum_n \|\mathbf{v}_n\|^2 + \frac{\lambda_u}{2}\sum_m \|\mathbf{u}_m\|^2 + \frac{\lambda_{b_u}}{2}\sum_m b_m^2 + \frac{\lambda_{b_v}}{2}\sum_n b_n^2 \tag{1}$$

$\mathcal{I}$ - our dataset of ratings
μ - the average rating of all data in dataset $\mathcal{I}$
$r_{m,n}$ - the rating given by user m to item n
$\mathbf{u}_m$, $\mathbf{v}_n$ - vector valued parameters representing the “taste” of user m and item n, respectively
b_m, b_n - scalar valued parameters representing the “bias” for user m and item n, respectively
λ - regularization hyperparameters

In this part you are required to:

1. define all hyperparameters required by the model and implement a class to encapsulate these hyperparameters
2. define a class called `MFModel` that represents all the model parameters. This class should also initialize parameter values.

[1] `http://files.grouplens.org/datasets/movielens/ml-1m.zip`

Part 3. **The Algorithm** In class we discussed two possible approaches to minimizing the cost function in Equation 1: *Alternating Least Squares* and *Stochastic Gradient Descent*.
In this part you are required to:

1. Derive the gradient of Equation 1 with respect to each of the variables $\mathbf{u}_m$, $\mathbf{v}_n$,b_m, and b_n.
2. *Stochastic Gradient Descent*
   (a) Use the derivation in step 1 to derive the update equations for each of the variables $\mathbf{u}_m$, $\mathbf{v}_n$,b_m, and b_n needed for *Stochastic Gradient Descent*
   (b) Implement a class encapsulating all the hyperparameters needed for *Stochastic Gradient Descent*
   (c) Implement a function `LearnModelFromDataUsingSGD` which takes in a dataset, a model and a set of algorithm hyperparameters and learns the model parameters using *Stochastic Gradient Descent*
3. *Alternating Least Squares*
   (a) Use the derivation in step 1 to derive the update equations for each of the variables $\mathbf{u}_m$, $\mathbf{v}_n$,b_m, and b_n needed for *Alternating Least Squares*
   (b) Implement a class encapsulating all the hyperparameters needed for *Alternating Least Squares*
   (c) Implement a function `LearnModelFromDataUsingALS` which takes in a dataset, a model and a set of algorithm hyperparameters and learns the model parameters using *Alternating Least Squares*

Part 4. **Machine Learning Code Diagnostics**
According to the theory our optimization procedure should be decreasing our the value of our error function with each iteration of our algorithm. One faithful way to debug our code is to make sure this is indeed the case. One way to determine if we are overfitting the noise in our training data is to measure the error on a validation set as well.

In this part you are required to:

1. Modify the functions `LearnModelFromDataUsingSGD` and `LearnModelFromDataUsingALS` to output the sum of squared errors on the training data before each iteration to a file
2. Modify the two functions above to optionally take a second dataset and if given output the sum of squared errors on this dataset to a second file

Part 5. **Evaluation**
In class we discussed a number of evaluation metrics that can be used to assess the performance of your approach:

- Root Mean Squared Error (RMSE)
- Mean Percentile Rank (MPR)
- Precision At K (P@k)
- Recall At K (R@k)

- Mean Average Precision(MAP)

In this part you are required to:

1. Implement a function that takes a (learned) model and a test set as input and outputs a ranked list of items for each user in the test set
2. implement a function for each of the evaluation metrics above, which takes the above list as input and computes the metric value (for P@k and R@k, k will be a variable)
3. implement a function that takes in a user index and prints the following in human readable format:
    (a) the user's history- the names of items the user rated in the training set and the rating given to those items
    (b) the top h items recommended to the user by our model, where h is a parameter

Part 6. The Main Flow

In this part we will put all the pieces together. Implement the main flow of your program to do the following:

1. Read in data and split into train and test
2. Set model and algorithm hyperparameters based on input arguments/configuration file
3. Learn model using algorithm specified by input arguments/configuration file (should output train and test error to files)
4. Compute the following evaluation metrics on the test set: RMSE, MPR, P@2,P@10,R@2,R@10,MAP
5. Output the hyperparameters,algorithm and metric evaluations to a file, the file should also include the amount of time needed for training

Part 7. Diagnostics and Analysis

In this part you will use the code you implemented above to create several plots (marked by green Deliverable tag) that you will turn in along with your code

In order to make sure our algorithm is implemented correctly we want to see that the training error is being reduced over time in our algorithm

Deliverable 1. Plot the train and test error as a function of training iteration. Train and test plots should appear in the same figure, clearly marked. Each algorithm (ALS and SGD) should have a separate plot. You may use whatever configuration of hyperparameters you wish for this plot, but make sure you specify the choice.

Hyperparameters can sometimes have a great effect on generalization performance. We will consider the effect of regularization and the number of dimensions.

Deliverable 2. For one of the two algorithms fix all hyper-parameters except for the regularization parameters. Make the simplifying modeling choice $\lambda_v = \lambda_u = \lambda_{b_u} = \lambda_{b_v} = \lambda$ (i.e. all the regularization parameters are set to the same value). Vary the value of λ from 0.1 to 1000 in multiples of 10. Choose two of the generalization metrics on the test set output by your code and plot the results as a function of $\log \lambda$. All hyper-parameter configurations of the algorithm should be clearly specified.

Deliverable 3. For the same algorithm you picked above and for (one of) the best setting(s) of λ based on your analysis above do the following: fix all hyper-parameters except for d the number of latent dimensions. Now run the flow several times varying d for all values in the set $\{2, 4, 10, 20, 40, 50, 70, 100, 200\}$. Plot the values of your choice of generalization metrics (same choice as above). Again specify all your modeling choices clearly

Some hyper-parameters can also significantly effect the training time of the algorithm.

Deliverable 4. For the same experiments as above plot the training run-time as a function of d.

Deliverable 5. For 5 users who have rated 3 or more items in the training set, provide a human readable format of the recommendations that would result from applying the learned model.

Deliverable 6. Your Code. Be prepared to explain where and how you implemented each of parts 1-6 above.