Bonus Project: A Recommender System

E4525 Fall 2019,

IEOR, Columbia University

Due: Dec 6th, 2019

1 Recommendation Systems

A recommender system seeks to predict a user *rating* or *preference* for a particular item. They have a large number of applications in e-commerce suggesting visitors new books, movies, etc based on their past behavior or transaction history.

There are, broadly speaking, two approaches to build a recommender system:

Content-based Filtering For each user, a profile is build based on particular features of the items he has liked in the past. For example, movies are selected based on their genre and actors, or books are suggested based on subject matter and author, etc.

Collaborative Filtering A particular users ratings are predicted using the ratings of other users that have had similar rating behavior in the past.

Content-based filtering requires significant feature engineering and, as such, is domain specific. Features that are informative for a movie rating problem may not be relevant to a book rating system.

Collaborative filtering, on the other hand, is completely agnostic to the item features and, as such, has broad applicability. See the Recommender Systems Handbook [1] for a thorough description of recommender systems and their implementation.

In this exercise we will bill a toy rating prediction system based on the MovieLens'[2] data set of one million movie ratings.

Our inputs are *some* elements of the ranking matrix

$$r_{u,i} \tag{1}$$

where $u = 1, 2, \ldots, U$ is an index that runs trough all the users of the system, $i = 1, 2, \ldots, I$ is an index that runs through all the items (movies) available. $r_{u,i} = 1, 2, \ldots, R = 5$ is the ranking given by user u to item i. In general there are $I \times U$ entries in this matrix, but we only have available $N \approx 1,000,000 \ll I \times U$ observations. The goal is to predict $r_{u',i'}$ for yet unobserved combinations (u', i') of users and items.

We will assess performance using the **mean square error** between observed rankings $r_{u,i}$ and the predicted rankings $\hat{r}_{u,i}$.

$$L(\theta; \{r\}) = \frac{1}{N_{\mathcal{S}}} \sum_{u,i \in \mathcal{S}} \left(r_{u,i} - \hat{r}_{u,i}\right)^2 \tag{2}$$

for a test set $\mathcal{S}$ of observations (u, i) that have not been use during model training. $N_{\mathcal{S}}$ is the number of elements of $\mathcal{S}$, and θ is a vector of model parameters that we will have to learn from the pairs (u, i) in the training set $\mathcal{T}$.

2 Matrix Factorization

We will follow the exposition on `https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf` [3], but see also section 5.3 of [1] .

A successful approach to the prediction of the ranking matrix $r_{i,u}$ is based on a **matrix decomposition** model, where our ranking predictions are

$$\hat{r}_{i,u} = \mu + b_u + b_i + p_u^T q_i \tag{3}$$

The model parameters $\theta = (\mu, b_u, b_i, p_u, q_i)$ are defined as

Mean rating μ is the average rating of all users over all movies in our training set $\mathcal{T}$

$$\mu = \frac{1}{N_{\mathcal{T}}} \sum_{(i,u) \in \mathcal{T}} r_{u,i} \tag{4}$$

User Bias b_u is a *per user* bias that will be higher for users that give high average ratings to all movies.

Item Bias b_i is a *per item* bias that will be higher for the more popular (higher ranked) movies.

User Embedding $p_u \in \mathbb{R}^F$ is a *per user* F dimensional vector that maps user u into some kind of abstract *taste space*.

Item Embedding $q_i \in \mathbb{R}^F$ is a *per item* F dimensional vector that maps item i into the taste space.

The **interaction** dot product $p_u^T q_i$ drives users and items pointing in nearly parallel directions in the *latent space* $\mathbb{R}^F$ towards higher ratings.

The dimension F of the latent space is unknown a priory and is a model hyper-parameter that must be chosen by cross validation.

In the special case $F = 0$, where there is no interaction term $p_u^T q_i$, the *relative* ranking of items are the same for all users

$$r_{u,i} - r_{u,i'} = b_i - b_{i'} \tag{5}$$

and we say that items are ranked by **popularity**.

A recommender system that wishes to provide **personalized** rankings instead of just suggesting the same items to all users needs an interaction term $p_u^T q_i$ because the preferences of each user for particular kinds of items is encoded exclusively on the interaction term $p_u^T q_i$.

As neither the biases, not the embeddings are know a priory, to train the model we need to learn the following

- b_u: U coefficients
- b_i: I coefficients
- p_u: $U \times F$ coefficients
- p_i: $I \times F$ coefficients.

Provided that the embedding dimension F is small, $U \times (F+1) + I \times (F+1)$ will me much smaller than the $U \times I$ entries on the ranking matrix.

2.1 Loss Function

We can write the mean square error loss function expliclity in terms of the model parameters, including a regularization penalty λ_p as

$$L(b_u, b_i, p_u, q_i; R) = \frac{1}{N} \sum_{(u,i)\in\mathcal{T}} (r_{i,u} - \hat{r}_{i,u})^2 + \frac{\lambda_p}{2}(p_u^2 + q_i^2) \tag{6}$$

where, for convenience, we have assumed no regularization penalty for the biases ($\lambda_b = 0$).

If we define the prediction error

$$\Delta r_{i,u} = r_{i,u} - \hat{r}_{i,u} \tag{7}$$

The gradient of L can be computed as[1]

$$\frac{\partial L}{\partial b_u} = -\Delta r_{u,i} \tag{8}$$

$$\frac{\partial L}{\partial b_i} = -\Delta r_{u,i} \tag{9}$$

$$\frac{\partial L}{\partial p_u} = -q_i \Delta r_{u,i} + \lambda_p p_u \tag{10}$$

$$\frac{\partial L}{\partial q_i} = -p_u \Delta r_{u,i} + \lambda_p q_i \tag{11}$$

[1] we are ignoring a factor $\frac{1}{N_{\mathbb{T}}}$ that does not matter as it can be absorbed into the learning rate γ.

2.2 Learning Algorithm

We can learn the model parameters $\theta = (b_u, b_i, p_u, q_i)$ by gradient descent. Assuming a learning rate γ we have

$$\theta_{t+1} = \theta_t - \gamma \frac{\partial L}{\partial \theta} \tag{12}$$

or, explicitly, for our parameters, we have the update rule per (u, i) pair

$$b_u \leftarrow b_u + \gamma \Delta r_{u,i} \tag{13}$$
$$b_i \leftarrow b_i + \gamma \Delta r_{u,i} \tag{14}$$
$$p_u \leftarrow p_u + \gamma \left(q_i \Delta r_{u,i} - \lambda p_u \right) \tag{15}$$
$$q_i \leftarrow q_i + \gamma \left(p_u \Delta r_{u,i} - \lambda q_u \right) \tag{16}$$

We will train the model using **stochastic gradient descent** with a batch size B.

2.3 Parameter Initialization

To fully specify the model we must initialize model parameters. Initialization should not matter for final results provided gradient descent has converged, but it may impact significantly the rate of convergence.

We following parameters provide good convergence in practice[2]

$$b_u^0 \sim \mathcal{N}(0, 10^{-4}) \tag{17}$$
$$b_i^0 \sim \mathcal{N}(0, 10^{-4}) \tag{18}$$
$$p_{u,f}^0 \sim \mathcal{N}\left(0, \frac{1}{\max(1, \sqrt{F})}\right) \tag{19}$$
$$q_{i,f}^0 \sim \mathcal{N}\left(0, \frac{1}{\max(1, \sqrt{F})}\right) \tag{20}$$
$$\tag{21}$$

where the $\sqrt{F}$ factor guaranties that the expected value of $p_u^T q_i$ is of $O(1)$ independently of F, for $F = 0, 1, \ldots F$.

3 Project Description

You must submit a jupyter notebook for this assignment.

1. Download the **1 million** Movilens data set from `https://grouplens.org/datasets/movielens/1m/`.

[2]Mathematically, there are no p_u, q_i parameters to initialize when $F = 0$ so the max in the initialization formula is unnecessary. We write the expression this way to that it can be implemented emphas is with `numpy.random.normal` function.

2. Implement the model predictions $\hat{r}$ as a function of the parameters $(\mu, b_u, b_i, p_u, q_i)$.

3. Split the data into a training, validation and test sets.

4. Set aside the test data, and do not use until the very last step.

5. Write a function to initialize parameters as described in 2.3.

6. Implement (and check!) the loss function given in equation 6 and a function performing one full epoch of **stochastic gradient descent** using batches of size B.

7. Choose the learning rate γ by fitting the model to the training data with $F = 0$.

   This should converge in very few epochs (complete iterations over the training set). The $F = 0$ model is the **popularity** model.

8. Record, and estate clearly the performance on the validation set of the popularity model.

   Our goal will be to find a **personalized** model ($F > 0$) with lower mean square error.

9. Select a small F (say less or equal to 10) and, by training different models, select hyper-parameters (using the mean square error in the validation set as selection criteria)

   batch size B

   regularization penalty λ_p

   epochs T (this is the number of steps that the stochastic gradient descent algorithm iterates over the whole training set).

   learning rate γ, you already selected it by fitting with $F = 0$, but you can fine tune it here if you wish.

   F Determine a range of values of F were performance looks good.

   The performance of the model is not very sensitive to most of the hyper-parameters provided they are in a sensible range. Do not spend a lot of time trying to find the absolute best parameters. Just select parameters that give good results (better than the popularity $F = 0$ model) and do not require a lot of computation to train.

   Be sure to record clearly in your notebook the models you tried and their performance in both the training and validation set.

10. **Select three candidate models with different values of** F that look the most promising to you.

    Using the total training + validation set data, select the best out of those three models using 5-Fold cross validation.

11. Measure the performance of your best model on the test set data.

4 Implementation Advice

- Read carefully the `README` file of the MovieLens data set to understand the layout of the data.
- You can use `panda.read_csv` file to parse the ratings data, the arguments `sep`,`header` and `names` can be useful.
- There is no guarantee that the `userId` and `movieId` variables are consecutive. It is good practice to use `sklearn.preprocessing.LabelEncoder` to map `userId` to the range $u = 0, \ldots, U-1$ and `movieId` to the range $i = 0, \ldots, I-1$.
- To generate training batches you will need parameters b_u, b_i, p_u, q_i for a collection of users and items of size B. If `users` is an `numpy.array` of B users , and `b_u` is the bias array of size U, `b_u[users]` will be an array with the biases of those B users. In the same fashion, if `p_u` is a $U \times F$ array of user embeedings, `p_u[users]` will be a $B \times F$ array of the embeedings for those users.
- If you need an example of an stochastic gradient descent optimization, the function `optimize_poisson` in the module `poisson_regression` should be a small, self contained example.
- You should check your loss and and gradient carefully before trying to do any significant amount of training.
- The computational cost grows with the size F of the latent space. Start with small values of F and work your way up.
- `numpy` does the right think with vectors of size zero, so you should be able to write only on model for all $F's$ provided initialization is as per section 2.3.
- Results are not very sensitive to most hyper parameters. Do not expend too much time trying to optimize all of them. Just send most parameters to sensible values and focus on F.

5 Bibliography

References

[1] Francesco Ricci, Lior Rokach, and Bracha Shapira, editors. *Recommender Systems Handbook*. Springer, 2015.

[2] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4), 2015.

[3] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.