Advance Topics in Recommender Systems

HW2 : Solving one-class matrix factorization with Bayesian Personalized Ranking

Mor Krispil Or Wolkomir Afek Adler

February 12, 2020

1. Bayesian Personalized Ranking

1.a. Notations

Let U be the users hidden matrix, V be the item hidden matrix and B the item bias vector. We use the following subscripts u for a given user, p for a positive item user u consumed and n for a negative item user u did not consume.
We denote $\{U\}$ as the set of users and $\{I\}$ as the set of items. We also denote the parameters of the model as θ.

$$\theta := U \cup V \cup B$$

1.b. Objective of the model

Our objective is to maximize the posterior probability plus a regularization term.
The objective:

$$\max \log P(\theta|D) = \max_{\theta} \sum_{u,p,n} \log \sigma(B_p + U_u \cdot V_p - B_n - U_u \cdot V_n) - Regularization$$

Where

$$Regularization = \lambda_u \sum_{u \in \{U\}} ||U_u||^2 - \lambda_v \sum_{v \in \{V\}} ||V_v||^2 - \lambda_{v_b} \sum_{v \in \{V\}} ||B_v||^2$$

And $\lambda_u, \lambda_v, \lambda_{v_b}$ are the regularization hyper - parameters of the user, item and item bias respectively.

Motivation:
let's look at one example, without the log (since the log is there just in order to move from

a product of i.i.d samples to summation, such that we will be able to reach to a close form gradient) -

$$\sigma(B_p + U_u \cdot V_p - B_n - U_u \cdot V_n)$$

This is how we model the probability that a given user likes more a positive item than a negative item. We want to maximize this probability over all the seen data, such that the hidden vectors will represent the user characteristics.

1.c. Negative examples

Since the BPR model needs negative samples that we do not have in our data set - We sample new negative samples each epoch. Re-sampling is necessary because we do not know how a given user relates to an item whom the user did not rank, and anchoring an unknown item as negative across all iterations will harm our predictions on test time in expectancy. We create the train set at the beginning of the algorithm [1] such that each row is a triplet - $[user, positive\_item, negative\_item]$. We treat the negative item sampling method as an hyper-parameter, and choose from two different methods:

- Uniform distribution.
- Popularity based distribution - proportional to the times we encounter an item in the original train set (item popularity).

In both cases, we sample for each row in the original data set one negative item. We sample without replacement, unless the number of items the user purchased is bigger than the number of items the user did not purchased.

1.d. Update steps

Since we have seen the update steps deviation in class (by the chain rule), we will just note here the update steps without calculating derivatives. We provide here update steps for one sample:

Let:

$$e_{u,p,n} := error_{u,p,n} = [1 - \sigma(B_p + U_u \cdot V_p - B_n - U_u \cdot V_n)]$$

So the update steps are:

$$B_p \mathrel{+}= e_{u,p,n} - \lambda_{v_b} B_p$$

$$B_n \mathrel{+}= -e_{u,p,n} - \lambda_{v_b} B_n$$

$$U_u \mathrel{+}= e_{u,p,n} \cdot (V_u - V_n) - \lambda_u U_u$$

$$V_p \mathrel{+}= e_{u,p,n} \cdot U_u - \lambda_v U_u$$

$$V_n \mathrel{+}= -e_{u,p,n} \cdot U_u - \lambda_v U_u$$

[1] We created 50 sets of negative items, one per each row. Knowing that we will need less than 50 epochs to train.

1.e. BPR pseudo-code

Algorithm 1 BPR

Create negative samples per each epoch
Initialization parameters: biases and hidden matrices
for $k \leftarrow 1$ to n_{epochs} **do**
    Add the pre-computed negative samples for epoch k to the positive samples
    for $(u, p, n) \in D$ **do**
        `Compute error`
        `Compute derivatives`
        `Compute derivatives with momentum`
        `Update parameters`
    `Compute train,validation metrics`
    `LR = LearningRateShecduler.update(epoch)`
    `If Earlystopping(epoch,validation error): STOP`

* In the first n_bias_epochs we update only the bias terms.

1.f. Hyper-parameters

As each experiment is time consuming we decided mostly to use fixed hyper-parameters and manual optimization.

- **L**atent Factor dimension – D.
- **R**egularization terms - $\lambda_u, \lambda_v, \lambda_{v_b}$
- **W**eight initialization: There are many options here including setting the seed. We chose to initialize the biases to zero and the hidden factors to be samples from $\mathrm{N}(0, \frac{1}{d})$
- **N**umber of epochs – number of epochs to preform, if not early stopping occurred.
- **M**omentum term: for applying exponential moving average over the gradients. Usually $\beta_{momentum} \in (0.1, 0.95)$ we set it to 0.9.
- **L**earning rate and learning rate annealing multiplier.
- **E**arly stopping parameters. As discussed in 'Checking for convergence' section.
- **N**egative sampling strategy, as discussed in 'Negative examples' section.

1.g. validation set

1.g.i. validation set creation

We created the validation set just as the test set is built in order to optimize the hyper-parameters on a test like scenario. For each user we created a set of ranked and did not rank and sampled from those distributions one item according to the test type (random or popularity). The validation samples were taken our from the original data set to create the train set

1.g.ii. Checking for convergence

- **m**in_epoch - minimum eooch to update.
- **n**_iter - of times the error is increasing.
- **a**nneal_times - maximum anneal times.
- factor - the factor that decreases the learning rate by.

```
def stop(self, mf, epoch, value):
    if epoch >= self.min_epoch:
        if self.annealing_counter == self.anneal_times:
            return True
        if value < self.last_value:
            self.consecutive_increasing_errors += 1
        if value >= self.last_value:
            self.consecutive_increasing_errors = 0
        if self.consecutive_increasing_errors >= self.n_iter:
            # anneal
            mf.lr.lr *= 0.1
            self.annealing_counter += 1
            self.consecutive_increasing_errors = 0
            print('#' * 50 + 'learning rate annealed')
    self.last_value = value
    return False
```

Figure 1: Stopping Algorithm

1.h. Training the last model

We would train the best model on all the data (train + validation) with the same hyper - parameters of our chosen model.

1.i. Implement your model

We attached the code in python based on all assumptions and pseudo code above.

1.j. Hit Ratio@K + MPR

Negative Sampling method	Data-set	Loss	Accuracy
popularity	train	-2.972e+05	0.876
popularity	validation	-1.702e+03	0.8944
uniform	train	-1.956e+05	0.9197
uniform	validation	-1.284e+03	0.9109

Table 1: Results on Train and Validation

K	all items	items user did not rank
1	0.01821	0.06755
10	0.125	0.2483
50	0.332	0.4818

Table 2: Hit Rate at K on Validation

MPR : 0.08122