ALS

November 2019

1 Objective

minimize

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

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_u, λ_v, λ_{bu}, λ_{bv} are regularization hyper-parameters.

2 Update Steps

ALS

Update steps are:

$\forall m' \in m$:

$$\mathrm{U'_m} = (\sum_{D'} \mathrm{V_n}\mathrm{V_n}^{\mathrm{T}}\lambda I)^{-1} \sum_{D'} (r_{m',n} - b_{m'} - b_n - \mu)\,\mathrm{V_n}$$

$$b'_m = (|D'| + \lambda_{bu})^{-1} \sum_{D'} \left(r_{m',n} - \mathrm{U'_m}^{\mathrm{T}}\mathrm{V_n} - b_n - \mu\right)$$

$\forall n' \in n$:

$$\mathrm{V'_n} = (\sum_{D'} \mathrm{U_m}\mathrm{U_m}^{\mathrm{T}}\lambda I)^{-1} \sum_{D'} (r_{m,n'} - b_m - b_{n'} - \mu)\,\mathrm{U_m}$$

$$b'_n = (|D'| + \lambda_{bv})^{-1} \sum_{D'} \left(r_{m,n'} - \mathrm{U_m}^{\mathrm{T}}\mathrm{V'_n} - b_m - \mu\right)$$

3 Pseudo Code

Algorithm 1: ALS pseudo code

```
Result: Write here the result
init $\mathrm{U_m}$, $\mathrm{V_n}$, $\mathrm{b_m}$, $\mathrm{b_n} \sim N(0, 0.1)$ ;
while true do
    print (iteration, train error, validation error);
    for $\forall m' \in m$ do
        Update
            $\mathrm{U'_m} = (\sum_{D'} \mathrm{V_n} \mathrm{V_n}^{\mathrm{T}} \lambda I)^{-1} \sum_{D'} (r_{m',n} - b_{m'} - b_n - \mu) \mathrm{V_n}$
        Update
            $b'_m = (|D'| + \lambda_{bu})^{-1} \sum_{D'} \left( r_{m',n} - \mathrm{U'_m}^{\mathrm{T}} \mathrm{V_n} - b_n - \mu \right)$
    end
    for $\forall n' \in n$ do
        Update
            $\mathrm{V'_n} = (\sum_{D'} \mathrm{U_m} \mathrm{U_m}^{\mathrm{T}} \lambda I)^{-1} \sum_{D'} (r_{m,n'} - b_m - b_{n'} - \mu) \mathrm{U_m}$
        Update
            $b'_n = (|D'| + \lambda_{bv})^{-1} \sum_{D'} \left( r_{m,n'} - \mathrm{U_m}^{\mathrm{T}} \mathrm{V'_n} - b_m - \mu \right)$
    end
    if validation error did not improve for the last $k$ iterations then
        break;
    end
end
```

4 Hyper-parameters

The hyper-parameters we need to tune are: λ_u, λ_v, λ_{bu}, λ_{bv} and also d the embedding dimension or the latent dimension of the user and item matrices. We used Bayesian hyper-parameter optimization, which is a probabilistic model based approach for finding the minimum of any function that returns a real-value metric. An efficient implementation for this optimization is given in the Python package hyperopt

5 Validation set

The validation set is given to us (we assume it contains (m, n, $r_{m,n}$) triplets that were recorded AFTER the training set). We will decide on R^2 as our key evaluation metric and init $BestR^2 = -9999$ and $cnt = 0$. Each iteration (after all user vectors and biases were updated and alternately also item vectors and biases) we will generate predictions on the entire validation set. We will evaluate the R^2 ,if it is higher than $BestR^2$ we will update $BestR^2 = R^2$ and set $cnt = 0$, otherwise we set $cnt + = 1$. Once we cnt reaches a fixed value (say 10) we can stop the iterations and figure out the the model is not improving. In addition, we would print the R^2 for the training set to check if our model is learning and to see we are not over-fitting (and if so, tune the regularization hyper-parameters accordingly). Providing further metrics such as RMSE and MAE will also help us evaluate the model better.

6 Implementation

The implementation is provided in the attached .zip file. Please follow the instructions for running the code. the main modules in the code are:

1. ”data-preprocess” which contains the DataSet() class that holds the training,validation and test data sets along with the ratings in matrix form and other utilities.

2. ”resources” which contains the original data sets and configurations

3. ”ALS” which contains the Model() class. This class holds the model’s hyper-parameters, user and item matrices and biases and other utilities. This class has the train, evaluate, predict and tune methods.

Main work items:

1. read the training and validation sets and mapping them to continuous indexes
2. deal with cold users or items in the validation set
3. training and evaluating the model (already explained in previous sections)
4. giving predictions for the test set include cold users/items.

7 Results

The best results on the validation set were achieved with the following hyper-parameters $\lambda_u = 88.572$, $\lambda_v = 0.746$, $\lambda_{bu} = 8.076$, $\lambda_{bv} = 1.48$ and $d = 11$. The results on validation set showed $R^2 = 0.366$, MAE=0.6902, RMSE=0.888

8 ALS vs SGD

ALS iterates over the model parameters, whereas SGD iterates over training samples. In order for these models to run efficiently we must store the data differently. For ALS we use two mappings user-items ¡user, [(item,rating), (item,rating), ...]¿ and item-users ¡item, [(user,rating), (user,rating) ...]¿. Each time we update for a single user or item we will use the data available in the dictionary instead of slicing the entire data-set. In SGD (batch size=1) we iterate on one sample and update all the parameters. Therefore there is no need to store the data the way we did in ALS. Hyper parameter tuning is easier in ALS since there is no learning rate, therefore only the latent dimension and regularization hyper-parameters are tuned, in SGD we add the learning rate to this.