HW1 - advanced topics in recommender systems

Tel Aviv University
Department of industrial engineering

Mor Krispil
Or Wolkomir
Afek Adler

Code is attached but available as well in our git repository –
https://github.com/AfekIlayAdler/UNI-2020_AdavancedRecoSystems/tree/master/HW1

Part 1 - SGD

- ***Write the objective of a regression model with global bias, user bias, item bias and L2 regularization.***
  $\min\,(x_u, y_i) \sum_D (r_{ui} - \mu - b_u - b_i - x_u^T y_i)^2 + \lambda (\sum_u \|x_u\|^2 + {b_u}^2 + \sum_i \| y_i\|^2 + {b_i}^2)$

  - ***Write the update step for each parameter.***
    update steps:
    $x_u \leftarrow x_u + (e_{ui} \cdot y_i - \lambda\_u \cdot x_u)$
    $y_i \leftarrow y_i + \gamma (e_{ui} \cdot x_u - \lambda_i \cdot y_i)$
    $b_u \leftarrow b_u + (e_{ui} - \lambda_{ub} b_u)$
    $b_i \leftarrow b_i + \gamma (e_{ui} - \lambda_{ib} b)$
    Where $e_{ui} = r_{ui} - \hat{r}_{ui} = r_{ui} - (x_u^T y_i + b_u + b_i + \mu)$

  - *Write a pseudo code for the algorithm.*
    Answer:
    Pseudo code doesn't contain momentum for simplicity. More heuristics will be discussed in the hyper parameters section ($LearningRateShecduler, Earlystopping$).

```
$initalize\ parameters\ (biases\ and\ hidden\ matrices)$
$for\ epoch\ in\ n\_epochs$:
        $for\ u, i, r_{ui}\ in\ Foreach\ (u, i, r_{ui}) \in D$:
                $compute\ error$
                $compute\ derivaitives$
                  $compute\ derivaitives\ with\ momentum$
        $update\ x_u, b_u, y_i, b_i$
        $compute\ train, validation\ metrics$
        $lr = LearningRateShecduler.update(epoch)$
        $If\ Earlystopping(epoch, validation\ error)$: STOP
```

  - ***What hyper-parameters do you need to tune?***
    In the process of hyper parameter tuning we fixed some numbers to defaults in order to lower the search dimension since evaluating the function is expensive (time)|.

LearningRateScheduler – takes 1 hyper parameter α.
Anneals the learning rate by a factor each epoch.

$$lr := \alpha * lr$$

EarlyStopping: takes 3 hyper parameters $\alpha, \beta, \gamma, \delta$.
General idea – if there error is not decreasing, anneal stronger the learning rate, if it doesn't work after a couple of times – stop training.
$\alpha := minimun\ eooch\ to\ update$
$\beta := maximum\ anneal\ times$
$\gamma := \#\ of\ times\ the\ error\ is\ increasing$
$\delta := \#\ anneal\ factor, usually\ 0.1$

```
def stop(self, mf, epoch, error):
    if epoch >= alpha:
        if self.annealing_counter == beta:
            return True
        if error > self.last_value:
            self.consecutive_increasing_errors += 1
        if self.consecutive_increasing_errors >=
gamma
            # anneal
            mf.lr *= 0.1
            self.annealing_counter += 1
            self.consecutive_increasing_errors = 0
    self.last_value = error
    return False
```

Latent Factor dimension – D.
We have seen for SGD that D = 16 is quite good (by the "knee" method). Because there is a diminishing return when increasing D. The plot is available at in - knee_method_choose_hidden_dim.ipynb
Regularization terms: takes 4 hyper parameters $\lambda_u, \lambda_{ub}, \lambda_i, \lambda_{ib}$ for the weights of the hidden factors and biases for items and user (L2 regularization)
Weight 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 $N(0, \alpha/Latent\ Factor\ dimension)$
Number of iterations – number of iterations to preform, if early stopping
Momentum term: for applying exponential moving average over the gradients. Usually $\beta_{momentum} \in (0.1, 0.95)$ we set it to 0.9.

- **Explain how would you work with the validation set and how would you check for convergence?**
  *As we explained above (**EarlyStopping)** if the validation error does not increase for a fixed number of iterations, we anneal the learning rate and continue.*
  If it happens more then maximum anneal times we stop.
  Generally speaking, we found that with our default hyperparameters we are converging even after 50 – 80 iterations (with small learning rate) so we stop the algorithm after **Number of iterations**.

It is common to set a threshold for improvement, for example

$$if\ ValidationError_{i+m} - ValidationError_i < \alpha: \ STOP$$

Where α and m are hyperparametrs. But as we explained we didn't need to use it.

- **Validation set- how did *we* use it in our work?**
  1. In case of increasing error of validation, set M (3 in our case) iteration in arrow ->early stopping. We will stop running the model because it implies that we have an overfitting problem.
  2. After choosing all the parameters for the model, we will train the model again using the training and validation set.
  3. We are using the validation set for hyper-parameters tuned. We will see which hyper-parameters values maximized performance on the validation set. Finally, score the test set to see how well we did.

- ***How* would *you* train *the last \ best model?***

  We would train the best model on all the data (train + validation).

- **Implement an SGD solution** for **the model and train it using the training and validation data. Explain the main work items you had to take.**

  We attach the code in python 3.5 based on all the assumptions and pseudo code above.

- **What is the RMSE, MAE, R^2 and MPR of your model based on the validation set?**

  We choose our best model based on the RMSE, because hyper-parameter tuning did not outperform our base solution considerably, we decided to present it. The results:

RMSE	**MAE**	**R^2**
0.89462	**0.68918**	**0.36989**

<u>**part 2**</u>

- ***Regarding the objective of a regression model- Is there any difference from the SGD objective.***

  The answer is no difference. Our goal is to minimize the loss function-which is the same in both cases. ALS and SGD are just the derivative-based methods for minimizing the functions.
- ***What hyper-parameters do you need to tune?***
  1. regularizations-we can set different value for each regularization: L2_user, L2_item, L2_user_bias, L2_item_bias
  2. Weight initialization-first initialization of the vector is done by taking random values from the normal distribution N (0, V). We can set all values to be from a normal distribution with different V values for each vector.
  3. number of latent factors.

- ***Write the update step for each parameter***

$x_u \leftarrow \sum_{\mathrm{D\,u}} \left(y_i * y_i^T + \lambda \mathrm{I}\right)^{-1} \sum_{\mathrm{D\,u}} (r_{ui} - \mu - b_u - b_i) y_i$
$y_i \leftarrow \sum_{\mathrm{D\,i}} (x_u * x_u^T + \lambda \mathrm{I})^{-1} \sum_{\mathrm{D\,i}} (r_{ui} - \mu - b_u - b_i) x_u$
$b_u \leftarrow (|\mathrm{Du}| + \lambda)^{-1} \sum_{\mathrm{D\,u}} (r_{ui} - \mu - b_i - x_u^T * y_i)$
$b_i \leftarrow (|\mathrm{Du}| + \lambda)^{-1} \sum_{\mathrm{D\,i}} (r_{ui} - \mu - b_u - x_u^T * y_i)$

- ***Write a pseudo code for the algorithm***

```
$initalize\ parameters$
$counter = 0$
Do until convergence:
    $Foreach\ u \in D$:
        $x_u = \left(\sum_D y_i y_i^T + \lambda I\right)^{-1} \sum_D (r_{ui} - \mu - b_u - b_i) y_i$
        $b_u = (|D_u| + \lambda)^{-1} + \sum_D (r_{ui} - \mu - b_i - x_u^T y_i)$
    $Foreach\ i \in D$:
        $y_i = (\sum_D x_u x_u^T + \lambda I)^{-1} \sum_{D_u} (r_{ui} - \mu - b_u - b_i) x_u$
        $b_i = (|D_i| + \lambda)^{-1} + \sum_{D_i} (r_{ui} - \mu - b_u - x_u^T y_i)$
    $print\ error(train, validation)$
    $If\ Earlystopping(epoch, validation\ error)$: STOP
```

- **Explain how you would work with the validation?**
  Same answer as the one in part 1.

- **What is the RMSE, MAE, R^2 and MPR of your model based on the validation set?**
  We choose our best model based on the RMSE, because hyper-parameter tuning did not outperform our base solution considerably, we decided to present it. The results:

RMSE	**MAE**	**R^2**
0.9195	**0.7012**	**0.3323**

In the next page we provide a detailed comparison between our SGD model and our ALS model.

Compare the ALS and SGD solutions in terms of implementation, training, and quality.

	ALS	SGD
Implementation	Data structure: dictionary of user as a key and all his ratings and indices of all the items he rated, dictionary of item as a key and all its ratings and indices of users who rated the item.	Data structure: array of ratings.
Training	Updates the parameters for all users/items. Stopped improving in iteration 18.	Updates the parameters after each rating.
Quality	We will present the results after hyper tuning the parameters of each model and selecting the best parameters that got best results on validation set. The Final results are compared by using the next initialization of the parameters: • SGD: lr=0.25, l2_users=0.01, l2_items=0.01, l2_users_bias=0.001, l2_items_bias=0.001, h_len=18 • ALS: l2_users=0.892, l2_items=0.897, l2_users_bias=0.11, l2_items_bias=0.565, h_len=8 We could also see that in meaning of running time, the running SGD till convergence is longer than running ALS till convergence. We can see that ALS and SGD got close results, however SGD got slightly better results in all measures.	
	RMSE: • train: 0.761 • validation: 0.91954 • train_loss: 512680.90445 • validation_loss: 20097.205 R2: • train: 0.53283 • validation: 0.3323 • train_loss: 514095.16372 • validation _loss: 20240.99961 MAE: • train: 0.5934 • validation:0.70123 • train_loss: 512680.90445 • validation_loss: 20097.205	RMSE: • train: 0.72659 • validation: 0.89462 • train_loss: 453815.7012 • validation_loss: 4943.22754 R2: • train: 0.5343 • validation: 0.36681 • train_loss: 497479.68307 • validation_loss: 4925.50093 MAE: • train: 0.56867 • validation: 0.68965 • train_loss: 456949.14494 • validation _loss: 4937.7756