Advanced topics in Machine Learning. Concept of the solution of the programming assignment

Team Chanjo.
Johannes Jendersie, Anton Niadzelka

June 10, 2013

1 Challenge decription

The challenge's objective is easily described: Create a recommender system for first names! Given a set of names for which a user has shown interest in, the recommender should provide suggestions for further names for that user. The recommender's quality will be assessed on an evaluation data set. Thus, the task can be considered a standard item recommendation task.

2 Solution

We received data about user activities from nameling.net website. Data consists of 515,848 activities made by 60,922 users. Our idea is to assign to each activity corresponding rating. The most explicit user expression will be ranked higher. *ENTER SEARCH* will receive the highest rank, as the evaluation is restricted only to this activity and all other activities are biased towards the lists of names which were displayed.

So, afterwards we will get a table where for each name we will have some user rating or nothing. It is not possible to store the whole table on a usual laptop RAM, as the number of user names is too high. Moreover, the table is sparse as we have less than 10 activities in average pro user. So, we store only existing ratings for each user with the index number of the corresponding name.

The recommendation approach we are going to use is based on a model (1) described in the book Recommender Systems Handbook [1].

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(|R(U)|^{-\frac{1}{2}} \sum_{j \in R(u)} (r_{uj} - b_{uj}) x_j + y_j \right), \text{ where} \quad (1)$$

μ - average overl all table ,

$R(u)$ - set that contains all items rated by user u,

b_u and b_i indicate the observed deviations of user u and item i, respectively, from the average.

$$b_{ui} = \mu + b_u + b_i$$

y_j - factors used to characterize users based on the set of items they rated.

x_j - factors used to characterize items based on the set of items they rated.

For a given item i, the elements of q_i measure the extent to which the item possesses those factors, positive or negative.

The factors mentioned above are obtained using a singular value decomposition (SVD) of an initial matix, as SVD maps both users and items to a joint latent factor space of dimensionality f. As stated in book [1] the latent space tries to explain ratings by characterizing both products and users on factors automatically inferred from user feedback. Usual SVD algorithms are not easily applicable in our case, as we have a really huge sparse matrix. Unfortunately there is no optimized Java library to do this. So, we are going to use the algorithm described in [2] in chapter 4.3 to compute q_j.

We are going to try to modify the algorithm such that we obtain x_j and y_j. In case that we do not achieve an initialization of these we will use the following method as fall back.

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T q_i \tag{2}$$

In this method item-item and user-user relationships are not expressed explicit. To compensate the loss of information the rank f has to be increased by a factor of 2.

2.1 Learning

The values b_u and b_i has to be learned after the initialization. During that learning x_j and y_j will be adapted too.

LearnFactorizedNeighborhoodModel(Known ratings: r_{ui}, rank: f)
% For each item i compute $q_i, x_i, y_i \in \mathbb{R}^f$
% which form a neighborhood model
Const $\#Iterations = 20, \gamma = 0.002, \lambda = 0.04$
% Gradient descent sweeps:
for *count* $= 1, \ldots, \#Iterations$ **do**
  for $u = 1, \ldots, m$ **do**
    % Compute the component independent of i:
    $p_u \leftarrow |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j + y_j$
    $sum \leftarrow 0$
    for all $i \in \mathrm{R}(u)$ **do**
      $\hat{r}_{ui} \leftarrow \mu + b_u + b_i + q_i^T p_u$
      $e_{ui} \leftarrow r_{ui} - \hat{r}_{ui}$
      % Accumulate information for gradient steps on x_i, y_i*:*
      $sum \leftarrow sum + e_{ui} \cdot q_i$
      % Perform gradient step on q_i, b_u, b_i*:*
      $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$
      $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda \cdot b_u)$
      $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda \cdot b_i)$
    for all $i \in \mathrm{R}(u)$ **do**
      % Perform gradient step on x_i*:*
      $x_i \leftarrow x_i + \gamma \cdot (|\mathrm{R}(u)|^{-\frac{1}{2}} \cdot (r_{ui} - b_{ui}) \cdot sum - \lambda \cdot x_i)$
      % Perform gradient step on y_i*:*
      $y_i \leftarrow y_i + \gamma \cdot (|\mathrm{R}(u)|^{-\frac{1}{2}} \cdot sum - \lambda \cdot y_i)$
return $\{q_i, x_i, y_i | i = 1, \ldots, n\}$

Figure 1: Learning algorithm from [1]

3 Evaluation

The quality of the result will be measured by the root mean squared error (3).

$$\sqrt{\sum_{(u,i) \in TestSet} (r_{ui} - \hat{r}_{ui})^2 / |TestSet|} \tag{3}$$

We have a few parameters that have to be tuned. These are initial ratings for each action and the number of latent factors f. For that we are

going to cross validate our model with different values of these parameters and afterwards choose the one with the best result.

4 Work Plan

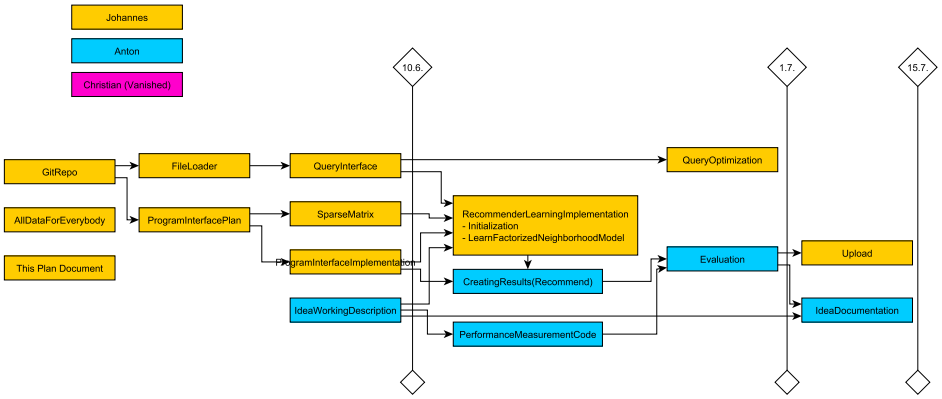

Figure 2: Graphical plan representation with dependencies and deadlines

Johannes Jendersie:

1. Team work organisation,
2. Program Interface,
3. Sparse Matrix,
4. Recommender Algorithm implementation.
5. Query Interface

Anton Niadzelka:

1. Idea Description,
2. Documentation,
3. Evaluation,
4. Performance Measurement implementation,
5. Recommendation creation

References

[1] Recommender Systems Handbook by Ricci, F.; Rokach, L.; Shapira, B.; Kantor, P.B. 2011, XXIX, Springer, Chapter 5

[2] Modeling Relationships at Multiple Scales to Im-prove Accuracy of Large Recommender Systems, Bell, R.M., Koren, Y., and Volinsky, C., Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2007.