LSINF2275 - Project2

Battaglia Gabriele - 29181900 - MAP(Double degree)
De Winter Nathan - 35171600 - SINF

May 2020

1 Introduction

For this project, we were told to implement a recommendation algorithm using the k-neighborhood method. We also implemented a second algorithm using another method using singular value decomposition (SVD) so we can compare their performances. Here is the details of the different algorithms.

2 Theory

2.1 K-neighborhood

The k-neighborhood algorithm consists in finding the k closest users, which have rated the film j, in order to find the rating for the user i that we want to predict. For doing this, we calculate a similarity value based on the joint ratings of a user i and the other users who saw the film j. In our implementation we used the cosine to calculate the similarity value so we can describe it like this:

$$RawCosine(u,v) = \frac{\sum_{k \in I_u \bigcap I_v} r_{uk} \cdot r_{vk}}{\sqrt{\sum_{k \in I_u} r_{uk}^2} \cdot \sqrt{\sum_{k \in I_v} r_{vk}^2}}$$

Once we have the similarity value of each user we take the k users with the biggest similarity value and we use this equation for finding the predicted rating:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} Sim(u,v) \cdot (r_{uj} - \mu_v)}{\sum_{v \in P_u(j)} |Sim(u,v)|}$$

where P_u corresponds to the set of k closest users to target user u.

2.2 Singular value decomposition

2.3 Theory

The SVD algorithm decomposes the matrix containing the different ratings into three matrices. The decomposition can be represented like this:

$$R = M\Sigma U^T$$

Since Σ is a diagonal matrix and simply scales the other two matrices, we can imagine to have merged it in one of them:

$$R = MU^T$$

Because of the missing values, R is sparse, so we cannot simply compute its eigenvectors; so we need to compute an approximation of the factorization; namely we need to compute two vectors p and q where p makes up the rows of $\hat{M}$ and q makes up the columns of $\hat{M}^T$ such that:

$$r_{uj} = p_u \cdot q_j$$

$$p_u \perp p_v \quad q_i \perp q_j \quad \forall i, j, u, v$$

To do this solve the following minimization problem with gradient descent, considering only the ratings r_{ui} we have:

$$\min_{\substack{p_u, q_i \\ p_u \perp p_v \\ q_j \perp q_i}} \sum_{r_{ui} \in R} (r_{ui} - p_u \cdot q_i)^2$$

Once the minimization problem is solved, we can compute the prediction matrix:

$$\hat{R} = \hat{M}\hat{U}^T$$

3 Validation method

For the MSE and MAE calculation, we do a cross-validation with different folds containing different ratings randomly shuffled. We run the cross-validation with n different folds, 10 by default, and for each value contained in the fold, we delete the corresponding value in our data. After that, we run each algorithms and we calculate the errors based on the difference between the original data and the newly missing value that we deleted earlier. Once we did it for each fold, we compute the average MSE and MAE.

4 Results

To contextualize the result we use a simple algorithm as baseline, that simply predict the rating of an item as its average rating.
The validation gives us these results:

Metric \Algorithm	Baseline	User-based knn	svd method
Average MSE	1.05	0.96	0.90
Average MAE	0.81	0.78	0.74

5 Parameters

5.1 K-neighborhood

When exploring different values for the parameter k, i.e. the number of neighbors to consider when predicting the rating, the best results are obtained with a small k; we settled on $k = 5$.
Looking at the data we can also understand why: the similarity rating are generally low, the average is around 0.2 while the average of the most similar neighbour is 0.5.
Since the algorithm relies on the k neighbour to be a fair approximation of the user, and since in this case it is generally not true, we end up with a group of neighbours with fairly different item rating and comparable similarity ratings; this leads to predictions that are similar to simply taking the average among all users, as with the baseline.

5.2 Singular value decomposition

The first interesting parameter for this algorithm is the dimension k of matrices $\hat{M}$ and $\hat{U}$, namely the columns of the former and the rows of the latter, the other dimension of both is determined by the shape of R.
The higher this value is the more similar the known values of R will be in $\hat{R}$ but also the less performing the prediction will be: the higher k the higher is the risk of overfitting.
As before we searched the possible values, ending with 5 as best possible value.

The other parameters involve the optimization problem, namely the learning rate α and the number of epochs; looking at the convergence rate we settled on $\alpha = 0.01$ and 10 maximum epochs.

6 Conclusion

With this algorithm, tuned as described, we end up with a distinct but small increase in performance from the baseline; the SVD method reaches the best results and has the added benefit of being the fasted method.

7 Code

7.1 Files

There are three files delivered:

- data.csv - contains the data converted in a more practical format
- algorithms.py
  - contains the functions *ubknn_algorithm* and *svd_algorithm* that take as input R and return $\hat{R}$
  - contains the two classes enclosing the algorithms
- main.py - when executed reads the data from data.csv and performs cross validation for each of the three algorithms

7.2 External resources

- external library **sklearn.metrics** to compute the cosine similarity