Recommendation on Yelp Businesses

Luca Bellani and Andrea Longoni

Algorithms for Massive Dataset

Abstract

This work focuses on recommending businesses to an user retrieved from `Yelp` Dataset [1]. The first part focuses on manipulating data to calculate the review rating in a more clever way. The collaborative filtering based on user similarity was later implemented. In the final section the UV decomposition phase was carried out: the performance of this dimensional reduction approach (written from scratch) was compared with that of the `spark.mllib.ALS` [2] through training and testing phases.

1 Introduction

Recommendation systems are applications which try to predict the taste of an user and suggest items that might like. These systems are classified into two broad groups:

- Content-based systems examine properties of the items recommended;
- Collaborative filtering systems recommend items based on similarity measures between users and/or items.

The data itself is represented as a utility matrix, giving for each users/items pair a value (generally the rating that the user gives to a certain item, from 1 to 5). In this environment the matrix is really sparse. The goal of recommendation systems is to predict the blank entries in the utility matrix. However it is not necessary to predict every blank entry, but to discover some entries in each row that are likely to be high.

2 Dataset Manipulation

Yelp provides JSON files regarding principally businesses, users and reviews. For this project three specific datasets have been taken into account, including `yelp_academic_dataset_review`, `yelp_academic_dataset_business` and `yelp_academic_dataset_user`. Once the `Spark` session is started, the function `from_json_to_RDD()` converts the JSON file to a resilient distributed dataset

(RDD), used for spreading across the nodes of the cluster collections of data [3]. The features of the three RDDs have been reduced, since we have discarded the columns that are not worthwhile, whereas the rows have been converted in a tuple format. Since the RDD keys are strings, we use the hash function SHA1 to reduce the amount of memory. This process permits to obtain integer values for `business_id, user_id` and `review_id`. The cardinality of either users, reviews or business does not go beyond 2^{32} (the maximum value representable by `int` type), thus we can use an integer to store our unique keys, occupying 4 bytes each (instead of keeping 20 bytes that the hash function generates).

2.1 Recalculation of the ratings

In a big data recommendation world, like Netflix or Amazon, the items are suggested by fetching and managing all possible information about its contents. The datasets considered in this report are populated with a lot of particular and efficient attributes: the review RDD provides columns such as `useful`, `funny` and `cool`. These fields lend us to a more accurate rating of the review, thus increasing (or decreasing) its star valuation and consequently the user consideration of the business. The item `fans`, belonging to the user dataset, is meaningful in terms of how much a reviewer could influence the opinion of someone interested in that particular business. These info are used for recalculating the ratings of each review, hence they should enhance their relevance. The decision of whether add or subtract Δ, which is the value of how the review is significant, is determined by the number of the stars: if it is less than 3, meaning that the business has not been appreciated by the user, the amount is decreased else viceversa. The formula to determine Δ is the following:

$$\Delta = \frac{1}{2}\left(\frac{\text{useful} + \frac{1}{2}(\text{funny} + \text{cool})}{\text{best\_useful} + \frac{1}{2}(\text{best\_funny} + \text{best\_cool})} + \frac{\text{fans}}{\text{best\_fans}}\right)$$

$$\Delta : [0, 1]$$

$$\text{overall} = \begin{cases} \text{stars} + \Delta & \text{if stars } \geq 3 \\ \text{stars} - \Delta & \text{if stars } < 3 \end{cases}$$

The main principle of this formula is the relevance of how useful and influent the review is: in the first ratio we add the value of `useful` with the sum of `funny` and `cool` divided by 2, since are considered less determinant than the first one. This quantity is then divided by the best values of these features regarding the business that is reviewed, because we find out how the review is helpful in respect to the others. The influence is determined in the same way by considering the attribute `fans`. To limit delta between 0 and 1, the result is divided by 2. This approach is applied by mapping the reviews in order to take `useful`, `funny` and `cool` and reducing by the `business_id` to get the maxima values. Next, this resultant RDD is joined with the `user_id` and the number of fans of the user. Finally, the overall is calculated after recovering the maximum number of fans.

3 Basic Recommendation

In order to recommend an item properly, the recalculation of the rating is not sufficient. For example, if a business has only few but excellent reviews, it is not guaranteed that the considered place is magnificent too. Before suggesting the locations to the users, they have been filtered out the businesses whose number of reviews are below to a certain threshold. Next, for each business, it has been carried out the average of the whole set of its reviews and sorted by the star rate. We have tried to simulate a situation called "Cold Start", that is a potential case in which there are no information about the user experience. For instance, if a new subscriber enters in a platform and there are no details about his status and tastes, the suggestions are necessarily given by retrieving data from another customers.

4 Collaborative Filtering

The process of identifying similar users and recommending what similar users like is called collaborative filtering. Recommendation for a certain user is then made by looking at the users that are most similar to him in this sense, and recommending items that these users like. The measurement of the similarity is provided by the distance measures, in particular Jaccard distance and Cosine distance [4].

In this project it has been implemented this approach according to the Cosine distance, since the Jaccard one works better with binary values.

The Spark execution has been organized in two functions: `cosine_dist()` and `collaborative_filtering()`. Both method takes a `user_id` as an argument, that we called U.

4.1 cosine_dist()

This method outputs the computation of the formula below:

$$\cos(\Theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \sum_{i=1}^{n} \frac{A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

The process is splitted into four RDD functional operations:

- for each user U', multiply the ratings that both U and U' made in the same business and keeping the original rating for the evaluation of the denominator;
- reduce by `user_id` for calculating the sum of the products processed above and aggregate the U' ratings in a list;
- for each user, compute the final result;
- filter the reviews with rating equal to zero and discard the reviews that U made.

4.2 collaborative_filtering()

This process invokes `cosine_dist()` and instantiates the best cosine distance values and the businesses in which U made reviews. The output is a sorted collection of businesses that the most similar users had visited and U had not.

5 UV Decomposition

A different technique to obtain the blank values is to predict them. This is performed by multiplying two matrices, called U and V. The process itself is called "UV Decomposition". Considering an utility matrix M, with n rows and m columns (i.e., there are n users and m items), then we might be able to find a matrix U with n rows and d columns and a matrix V with d rows and m columns, such that UV closely approximates M in those entries where M is nonblank. If so, then we have established that there are d dimensions that allow us to characterize both users and items closely. We can then use the entry in the product UV to estimate the corresponding blank entry in utility matrix M. This process is called UV-decomposition of M. The accuracy of how the matrix UV is close to M is relied on the RMSE (Root Mean Squared Error):

$$RMSE(UV_u, M_u) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(M_{ui} - UV_{ui})^2}$$

where m is the number of nonblank entries of M and u is the row of the utility matrix referred to a specific user. The starting point is to arbitrarily choose U and V and repeatedly adjust them, by alternating the two matrices in order to reduce the RMSE as much as possible. It is typical that there will be many local minima, meaning that the matrices U and V are no more adjustable for reducing the RMSE. The goal is to find the global minimum: we need to pick many different starting points, that is, different choices of the initial matrices U and V. However, there is never a guarantee that our best local minimum will be the global minimum [4].

The optima formulas for computing elements of U and V are respectively:

$$x = \frac{\sum_j v_{sj}(m_{rj} - \sum_{k \neq s} u_{rk} v_{kj})}{\sum_i v_{sj}^2}$$

$$y = \frac{\sum_i u_{ir}(m_{is} - \sum_{k \neq r} u_{ik} v_{ks})}{\sum_i u_{ir}^2}$$

We shall use m_{ij}, u_{ij}, and v_{ij} for the entries in row i and column j of M, U, and V, respectively. Here, $\sum_j$ and $\sum_i$ are shorthands for the sum over all j and i such that m_{rj} and m_{is} are nonblank. $\sum_{k \neq s}$ and $\sum_{k \neq r}$ are the sums over all values of k between 1 and d, except for s and r respectively.

5.1 Implementation

In the first step we create the matrix U, with m rows and d columns (which in the program is called `rank`), and V, with n columns and d rows. All the entries of both of them are initially 1. Another test is performed initializing all the entries with the average value of all ratings in the dataset. However, this choice implicates that the elements of U and V should be $\sqrt{average\_value/rank}$. Then we divide the review dataset into training and test sets, with the aim of establishing the estimation of this model. The Spark alternation procedure of calculating the entries of U and V is based on broadcasting these two matrices to the nodes of the cluster, since at every stage we need the updated values. The two delegated functions, `update_U` and `update_V`, reflect the two general formulas:

- take those reviews in the training set belonging to the user/business;
- looping on the number of dimensions, calculate the formula for each non-blank entry of M pertained to the user/business;
- output the tuple of the id of user/business and its updated vector;
- after the map execution of one of the two functions, the new matrix must be broadcasted, because its data are necessary for recomputing the update of the other table.

This process is repeated for the number of iterations required. The evaluation of the performance of UV is actuated by the method `rmse`, which simply generates the actual RMSE loss of this model given a test set.

5.2 ALS

ALS is a class of `pyspark.mllib.recommendation` [2] that uses the alternating least squared technique [5] to predict the missing entries of an utility matrix. The method `ALS.train()` requires a training set of type RDD with integer keys, thus the pre-processing of hashing keys suits us. The implementation starts with subdividing the dataset into three parts: training set, validation set and test. The validation set give us the opportunity to check which matrix rank enhance the performance. In fact, a cross validation step is computed. Then the model is re-trained and tested with the test set. The RMSE calculation is implemented by another function called `compute_error()`.

6 Test

In this section we examine the performances of the models implemented in this work. The review RDD is sampled by 1/50 from the original one; thus, the user RDD is formed by all the users that made a review in the sampled RDD. The UV Decomposition and ALS steps are tested iterating 2 times each.

- UV Decomposition (the rank used is equal to 2):

  - UV entries equal to 1: the test error is 2.16.
  - UV entries equal to the $\sqrt{average\_value/rank}$: the test error is 1.87.

- ALS: we have performed the Cross Validation in order to retrieve the best matrix rank. The values considered are $2, 4, 8, 12$. The optimum rank is 12 and the validation error is 4.29. The test error is 4.25.

  To compare this model with the UV one, we have tested it without Cross Validation using rank equal to 2. The test error is 5.88.

7 Conclusion

The research of the global minimum in the UV Decomposition is more accurate when the UV entries are referred to the average value of the nonblank entries of the utility matrix, as seen in the previous section.

In terms of time execution, the last one takes around half of the time to compute with four different matrix ranks, despite the clear difference of RMSE between UV Decomposition model and the ALS model. This is probably due to the fact that the Spark method utilizes more cleverly the internal parallelization and approximates some calculus. At the contrary, the scratch method of UV Decomposition alternates the broadcast of data among the workers at every step, which significantly make the process slow and the most computational expensive. Even if these parallel processes increment the algorithmic cost, we should not avoiding the parallelization and the consequent broadcasting, because working on a single thread application is not a good idea.

References

[1] Yelp dataset.

[2] Als - spark.mllib documentation, 2023.

[3] Rdd programming guide, 2023.

[4] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive data sets*. Cambridge university press, 2020.

[5] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, aug 2009.