Algorithm for massive datasets

Mattia Ferraretto [00072A]

Academic year 2022-2023

Contents

1 Project requirements

The project refers to the *yelp* dataset published on kaggle under the CC-BY-SA 4.0 license, in particular, is focused on the analysis of the *yelp_academic_dataset _review.json* and *yelp_academic_dataset_business.json* files. The task consists of implementing from scratch a system that recommends businesses to users. You are free to choose the technique at the basis of the recommender system, as well as the strategy to be used in order to populate the utility matrix [1].

2 Developing approach

In the following section I will describe the approach used to develop the recommendation system. Let me introduce to you that the system recommend to users, interesting businesses based on their preferences (content-based filtering approach), is designed to scale up with large datasets and to be scalable with respect to the hardware resources available.

2.1 Defining the utility matrix

In a recommendation system the utility matrix represents the relationship between users and items. Its content might depend on what kind of relation the matrix is trying to define, for instance, when it represent a relation between users and their purchases, a common representation is a boolean matrix where an entry e_{ij} is 1 if the $i-th$ user has bought the $j-th$ item otherwise 0. However, when the relation is about users and movies watched, a real definition is more appropriate. Since the task of the project consists of recommending businesses based on user's preferences, the utility matrix is filled with the ratings (between 1 and 5) given by the user reviews. That is, the resulting sparse matrix is one where each row corresponds to a user, each column to a business and an entry is only filled in the case of $i-th$ user has rated the $j-th$ business. Data are loaded from *yelp_academic_dataset_review.json* and in the case a user a has reviewed more than one time the same item, the entry is filled with the review's average value.

2.2 Selecting features from items

Selecting features from items may be a challenging task, this, because is strictly related with the nature of items. In simple scenario, features could be selected manually, for example, when items are movies: director, cast, genre, publication year, and son on, are interesting elements to give a good representation of items. In more complex use case, when items are documents or images, a manual selection is infeasible. So a different a approach is needed. In the case of documents a statistical metric like TF.IDF is useful to remark the most important words of a document and use them as features, whereas, for images, tagging is one the most effective way. In the actual use case, items are collection of characteristics stored in the *yelp_academic_dataset_business.json* file which contains information such as *business_id*, *name*, *address*, *city*, *state*, *categories* and many others. Therefore, I simply selected, what in my opinion, are features useful to define a good representation of business, and in particular *city*, *state* and *categories* were chosen. Then, the selected features are represented as a collection of one-hot encoded vectors (items profiles), where each component is hot (1) if and only if the feature belongs to the business.

2.3 Scale up with large datasets

Working with huge amount of data implies to face limitations and issues which might occur during data processing. One of the most important aspects to consider are hardware resources, in particular CPU (see next subsection) and main memory bottlenecks. Focusing on memory bottlenecks, loading the entire dataset into the main memory is not feasible, in fact, to processes data from file I load small chunks with a fixed size. Using this approach I was able to process the entire reviews and business datasets. Another challenge in front of me was searching similar items in the dataset, and since that a naive approach might simply compare each pair in the dataset, when data required to be processed are a lots, such approach becomes very time-consuming. The solution proposed consists of Locality Sensitive Hashing (LSH) with random hyperplanes. In this way, when a query is issued to the system, only the candidate pairs are compared, avoiding to search within the entire dataset.

2.4 Hardware resources scalability

In a content based recommendation system, both, item profiles and user profiles are required to define the system. In my proposed solution, first of all, item profiles are computed (as just mentioned), then their dimensionality is reduced using Principal Component Analysis (PCA), finally user profiles are defined starting by item profiles and their corresponding rating assigned by the user. As you can imagine the workload needed to perform the overall operations grows as data grow and thinking of doing all in a sequential manner is not practicable, therefore, the system is designed to perform some operation in a parallel way, by means of parallel processes and hardware accelerator (GPU). In more details, parallel processes are involved to compute item profiles, user profiles and building the LSH index, while GPU is used to reduce the dimensionality of item profiles. At the beginning of the computation items are split into different subset with a fixed size in bytes called shards, then each shard is assigned to a process which will compute item profiles. The same approach is used to user profiles. It's a little bit different the construction of LSH index, where instead, even a shard is split into subset with respect to the number of available CPU cores. In conclusion, the system is designed to be scalable with respect to the hardware: the more resources are available than faster the system is to profile users and items.

3 Recommending

In a recommendation system the purpose is to recommend items to users. There are two different approaches: a content based and a collaborative filtering. The first one consists of recommending items to a user similar items rated highly by him, instead a collaborative filtering consists of selecting a set of users whose ratings are similar to the target user and then estimate target user's ratings based on ratings of the selected users.

As already mentioned, I proposed a content based recommendation system where items are represented with a one hot encoded vectors (then reduced with PCA) and users with vectors defined starting by item profiles. Looking at user profiles, how they are defined is very important and can lead to better or worse results. In particular, they are defined as follow: items rated by the user are selected, ratings are then normalized subtracting their average value, and finally, the user profile is obtained as the dot product between items and normalized ratings. The steps described are performed for each user and, during the process some rated items are hidden randomly allowing for better evaluation of system performance.

When a query is issued to the system the following steps are performed: a set of candidate pairs is returned through the LSH model, then, items rated in the past are removed, cosine similarity is computed between candidate pairs, and in the end, items with a similarity scores greater than zero are recommended to the user in descending order.

4 Evaluation of the system

Evaluating a recommendation system is a little bit different compared to a machine learning model. In fact, the list recommended to a user contains items never seen before by the user, and understand whether an items is of interest might be challenging. One of the most effective way is the user feedback, which simply consists of collecting user's reactions to new items, but in absence of it, as in my case, a different strategy is needed. In order to evaluate the system, two different independent analyses was provided. The first one is concerned about evaluating a specific query, the second one is trying to evaluate the overall performance of the system. Focusing on query analysis, to improve evaluation effectiveness, during the "training phase" (user profile definition) for each user the 30% of rated items were hidden randomly. The idea is to understand how well the system is then able to recommend interesting items (within the items previously hid) to a user. Then, the following metrics were used to evaluate the query performance:

- $Precision@K = \frac{\text{\# of recommended items @K that are relevant}}{\text{\# of recommended items @K}}$
- $Recall@K = \frac{\text{\# of recommended items @K that are relevant}}{\text{\# of total relevant items}}$

- $Diveristy@K = \frac{1}{K}\sum_{i=0}^{K}\frac{1}{K-1}\sum_{j=0}^{K} 1 - s_{ij}$, where s_{ij} is the Jaccard similarity

$Precision@K$ and $Recall@K$ are applied when the ground truth is available, on the other side, when items with no info are recommended, diversity is computed. The diversity aims to show how much items in the recommendation list are different from each other. In the situation where diversity is too much, might have as consequence that items in the list are not very aligned with the user's preferences, in a opposite situation, where diversity is too small, items recommended may be too similar with each other resulting boring to the user. Therefore, a good trade-off between precision and diversity should be found.

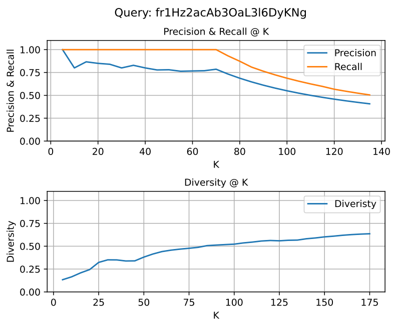
The metric used to evaluate the overall system performance is the Mean Average Precision@K (MAP@K). For a single query, Average Precision is the average of the precision value for the set of the top k documents retrieved, and this value is then averaged over queries. That is, let Q be the set of all possible queries, $q_j \in Q$ a query, and R_{jk} the set of ranked retrieval results in descending order, then

$$MAP@K = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{K_j}\sum_{k=1}^{K_j} Precision(R_{jk})$$

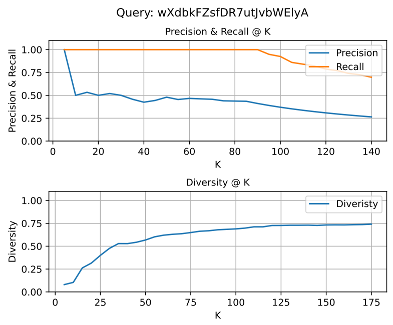
When a relevant document is not retrieved at all, the precision value in the above equation is taken to be zero. For a single query, the average precision approximates the area under the precision-recall curve, and so the MAP is roughly the average area under the precision-recall curve for a set of queries. Calculated MAP scores normally vary widely across queries when measured within a single system, for instance, between 0.1 and 0.7. [2]

5 Comments on results obtained

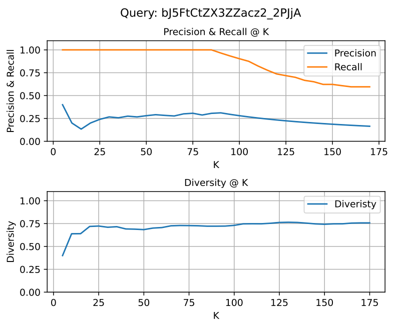
In the previously section the metrics used to evaluate the system were described, now, in this section, I will show to you the the results obtained over queries. In the following charts outcomes are shown.



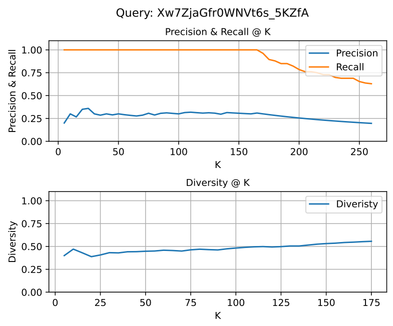
(a) Query 1



(b) Query 2



(c) Query 3



(d) Query 4

Figure 1: Query's charts

As can be seen from the above charts, the first query (1a) is performing very well. Indeed, the recall remains constant to 1 for, roughly, the first 75 items, then starts to decrease slowly. Instead, observing the precision, it also lies over 0.75 for the fist items and then decreases slowly. Looking at the diversity, it is close to 0 at the beginning then starts to increase gradually to 0.75. In my opinion, a correct behavior of the system is something like that, because when the precision is high, the expected diversity is close to 0, and as precision decreases the diversity among recommended items grows. Similar situation is evident in the second query (1b) where is more marked this kind of behavior. A different situation can be seen in query (1c and 1d) where the recall stays constant to 1 for the first items, then starts to decrease, while the precision is in general low and tends to decrease. Observing the diversity, I would say that in the query (1c) is high and flat, instead, in the query (1d) is low and flat. With high (from the beginning) and flat diversity, recommended items might result too diverse from each others and therefore not too aligned with user preferences. A little bit different is the situation in query (1d) where diversity is lower than with the query (1c); such outcome indicates that recommended items are more similar from each other, and when the curve is too flat (such as actual result) items may result too boring for the user.

In conclusion, queries (1a and 1b) are good and their outcomes might result interesting for the final user, on the other hand, queries (1c and 1d) are not performing very well due to the low precision and flat diversity.

Looking now at results obtained through the overall evaluation of the system with MAP@K computed on the top 200 most active users with a step size of 10.

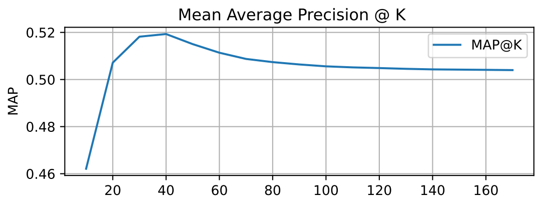


Figure 2: MAP@K with step size of 10

In the chart you see the curve at the beginning tends to increase from 0.46 to 0.52 and then, stabilizes around 0.5. Zooming out, it is easy to see a flat curve. When MAP@K curve is flat, it means that the precision values for different values of K remain relatively consistent or do not vary significantly as you increase K. In other words, as you consider larger and larger values of K the average precision for those values of K does not change much. A kind of this curve indicates that the system performance are relatively stable among different values of K, that is, the system maintains a consistent level of precision even when more items are considered.

6 Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

References

[1] Algorithms for massive datasets projects proposal, 2022 - 2023.

[2] An Introduction to Information Retrieval; Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze; Section 8.4