POLITECNICO
MILANO 1863

Systems and Methods for Unstructured Data Project: Amazon Reviews

Author: Tommaso Capacci (10654230)

Link to the project's GitHub repository

AY 2023/2024

Contents

1 Introduction

The dataset we've chosen to use represents a collection of reviews of products sold on Amazon. Datasets like this are usually used for **analytic purposes**, such as understanding the customers' needs and preferences, for **marketing purposes**, such as collecting the customers' opinions on a product, or for **sentiment analysis**, such as training a classifier to predict the sentiment of a review basing on its content, since information about the review's text is also provided.

In our case, we will consider the first task, since it can be reduced to the problem of writing a suited set of queries and can be also efficiently supported by the database technology we decided to use, which is MongoDB. We've decided to interpret the task of strategic analysis as the problem of extracting meaningful statistics from the dataset (such as the number of reviews per product, the average rating per product, the review with highest score, ...) which could be then included on the company's reports and used to make strategic decisions.

One of the main reasons behind our technical choice of using MongoDB as database technology is the fact that the dataset is in JSON format, which is natively supported by MongoDB. Moreover, it is composed of a collection of, possibly, nested documents, which is also a good fit for MongoDB, since it is a document-oriented database, meaning that it is specifically designed to store data as documents instead of relational tables.

2 Dataset

The dataset of our choice is managed by the University of California San Diego (UCSD) and is presented in this website. It is composed of multiple collections of documents, each one containing some reviews, registered between the 1996 and 2018, about a specific kind of products sold on Amazon. This fact makes the dataset suitable for our purposes, since it contains a lot of reviews but also would make it possible to filter the reviews by product category, allowing to eventually focus the analysis on a specific kind of goods.

In particular, in the website cited above, are provided 2 versions of the same datasets:

- *5-core*: this is a reduced version of the original, complete dataset, in which are included at least 5 of the most representative reviews per product and user

- *ratings only*: in this version of the dataset only information about the user, the product, the rating and the timestamp of pubblication time is kept for each review

While the second version of the dataset contains an higher amount of reviews, we decided to use the first one since it contained more interesting fields.

Amazon Fashion	5-core (3,176 reviews)	ratings only (883,636 ratings)
All Beauty	5-core (5,269 reviews)	ratings only (371,345 ratings)
Appliances	5-core (2,277 reviews)	ratings only (602,777 ratings)
Arts, Crafts and Sewing	5-core (494,485 reviews)	ratings only (2,875,917 ratings)
Automotive	5-core (1,711,519 reviews)	ratings only (7,990,166 ratings)
Books	5-core (27,164,983 reviews)	ratings only (51,311,621 ratings)
CDs and Vinyl	5-core (1,443,755 reviews)	ratings only (4,543,369 ratings)
Cell Phones and Accessories	5-core (1,128,437 reviews)	ratings only (10,063,255 ratings)
Clothing, Shoes and Jewelry	5-core (11,285,464 reviews)	ratings only (32,292,099 ratings)
Digital Music	5-core (169,781 reviews)	ratings only (1,584,082 ratings)
Electronics	5-core (6,739,590 reviews)	ratings only (20,994,353 ratings)
Gift Cards	5-core (2,972 reviews)	ratings only (147,194 ratings)
Grocery and Gourmet Food	5-core (1,143,860 reviews)	ratings only (5,074,160 ratings)
Home and Kitchen	5-core (6,898,955 reviews)	ratings only (21,928,568 ratings)
Industrial and Scientific	5-core (77,071 reviews)	ratings only (1,758,333 ratings)
Kindle Store	5-core (2,222,983 reviews)	ratings only (5,722,988 ratings)
Luxury Beauty	5-core (34,278 reviews)	ratings only (574,628 ratings)
Magazine Subscriptions	5-core (2,375 reviews)	ratings only (89,689 ratings)
Movies and TV	5-core (3,410,019 reviews)	ratings only (8,765,568 ratings)
Musical Instruments	5-core (231,392 reviews)	ratings only (1,512,530 ratings)
Office Products	5-core (800,357 reviews)	ratings only (5,581,313 ratings)
Patio, Lawn and Garden	5-core (798,415 reviews)	ratings only (5,236,058 ratings)
Pet Supplies	5-core (2,098,325 reviews)	ratings only (6,542,483 ratings)
Prime Pantry	5-core (137,788 reviews)	ratings only (471,614 ratings)
Software	5-core (12,805 reviews)	ratings only (459,436 ratings)
Sports and Outdoors	5-core (2,839,940 reviews)	ratings only (12,980,837 ratings)
Tools and Home Improvement	5-core (2,070,831 reviews)	ratings only (9,015,203 ratings)
Toys and Games	5-core (1,828,971 reviews)	ratings only (8,201,231 ratings)
Video Games	5-core (497,577 reviews)	ratings only (2,565,349 ratings)

Figure 1: List of the collections provided by the dataset

Among all the available collections, we've decided to use the one about **Digital Music** (which can be retrieved at this link) since this file includes a suitable number of datapoints (169.781) while still containing an interesting amount of attributes per document. Specifically, the documents inside the selected collection have the following shape:


```
{
    "image": ["https://images-na.ssl-images-amazon.com/images/..."],
    "overall": 5.0,
    "vote": "2",
    "verified": true,
    "reviewTime": "01 8, 2015",
    "reviewerID": "A36GE53TK8V94L",
    "asin": "B000T1EJ0W",
    "style": {"Format:": "MP3 Music"},
    "reviewerName": "MysticWolf229",
    "reviewText": "the theme song to the one and only movie that ...",
    "unixReviewTime": 1420675200
}
```


whose fields have the following meaning:

- **image**: an array of URLs pointing to the images of the product review;
- **overall**: the rating of the product, a float number going from 1 to 5;
- **vote**: the number of people that found the review helpful, saved as a string;
- **verified**: a boolean value indicating whether the purchase of the product from the user making the review has been verified or not;
- **reviewTime**: the date of the review, saved in RAW date format as a string;
- **reviewerID**: the alphanumeric ID of the user who wrote the review;
- **asin**: acronym of Amazon Standard Identification Number, is the alphanumeric ID that represents a specific product;

- **style**: a subdocument containing additional data about the version of the product. In the case of this collection it just contains information about the format of the product, whose possible values will be retrieved with a suited query;
- **reviewerName**: a string containing the name of the user who wrote the review;
- **reviewText**: a string containing the text of the review;
- **summary**: a string containing a summary of the review;
- **unixReviewTime**: the date of the review in Unix epoch time format.

3 Data Wrangling

After a quick inspection of the dataset we understood that the data entries it contained were mostly complete but, at the same time, we didn't have enough information to infer the value of missing attributes for incomplete documents.

One useful thing that could have been done in the scenario of a real analysis would be to find another dataset containing technical data about single products, together with their ASIN, and use this information to compute a join over this attribute and complete our dataset with this additional data: anyway, while in the presented setting this information would be very easy to be retrieved, this was not our case, since we weren't able to find any additional dataset with these characteristics and we couldn't afford (both from resources and time perspective) to perform a snapshot of the Amazon website through scalping.

The only interesting thing that is worth mentioning is the fact that we used a python script to extract a subset of the dataset:

```
import os

os.chdir("../Datasets")
json_file_path = os.path.join(os.getcwd(), "Digital_Music.
    json")
n = 150_000
suffix = str(n//1000) + "k"
output_file_path = json_file_path[:-5] + "_" + suffix + ".
    json"

with open(json_file_path, 'r') as input_file:
    with open(output_file_path, 'w') as output_file:
        for i in range(n):
            # EACH OBJECT MUST OCCUPY *EXACTLY* ONE LINE
            output_file.write(input_file.readline())
```

Specifically, as it can be seen, we used this script to round the number of documents to the nearest multiple of 50.000 by extracting the first lines of the original file. It is worth noticing that the core lines of this script can be easily modified to perform any kind of filtering over the JSON documents in this way:

```
import os
import json

os.chdir("../Datasets")
json_file_path = os.path.join(os.getcwd(), "Digital_Music.
    json")
n = 150_000
suffix = str(n//1000) + "k"
output_file_path = json_file_path[:-5] + "_" + suffix + ".
    json"

with open(json_file_path, 'r') as input_file:
    with open(output_file_path, 'w') as output_file:
        i = 1
        while i <= n:
            # EACH OBJECT MUST OCCUPY *EXACTLY* ONE LINE
            line = input_file.readline()

            # Be sure that the line is not empty
            if not line:
                break

            json_object = json.loads(line)

            # Generic filter
            if filter:
                output_file.write(line)
                i += 1
```

Anyway we decided to not apply any further filtering since we wanted to keep as many attributes as possible in order to be more free in the writing of the queries and eventually realize these filters through them: just keep in mind that, with larger data collections, it could be more efficient to perform filtering beforehand (e.g. to consider only reviews published from a specific point in time on) and this could be a way of doing it.

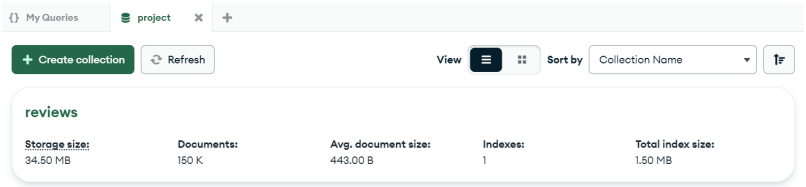

Figure 2: Physical occupation of the database on disk

4 Queries

1. Order Products by Average Rating
2. Most Reviewed Products
3. Average Helpful Votes per Reviewer
4. Number of Reviews with Images and positive Verfied Status
5. Average Review Length
6. Temporal Distribution of Reviews
7. Distribution of Overall Ratings
8. Number of Reviews by Verified Status and Rating
9. Average Number of Votes for Verified vs. Non-Verified Reviews
10. Average and Highest Rating per Format Style

4.1 Order Products by Average Rating

The most straightforward query that can be performed is to order the products by their average rating, in order to understand which are the most appreciated ones.

```
db.reviews.aggregate([
  { $group: { _id: "$asin", avg_rating: { $avg: "$overall" } } },
  { $sort: { avg_rating: -1 } }
]);
```

Results:

```
>_MONGOSH

> db.reviews.aggregate([
    { $group: { _id: "$asin", avg_rating: { $avg: "$overall" } } },
    { $sort: { avg_rating: -1 } }
  ]);
< {
    _id: 'B0016CNY5Y',
    avg_rating: 5
  }
  {
    _id: 'B0061RAN76',
    avg_rating: 5
  }
  {
    _id: 'B00VJBB9JA',
    avg_rating: 5
  }
  {
    _id: 'B00136NEQW',
    avg_rating: 5
  }
```

Figure 3: Query 1 (partial) results

As we can see from Figure 3, there are many products sharing the highest value for average rating, meaning that all the reviews that involve those products have the highest possible value, which is 5.

4.2 Most Reviewed Products

Another interesting query is to find the most reviewed products, in order to understand which are the most popular ones.

```
db.reviews.aggregate([
  { $group: { _id: "$asin", count: { $sum: 1 } } },
  { $sort: { count: -1 } }
]);
```

Results:

Figure 4: Query 2 (partial) results

4.3 Average Helpful Votes per Reviewer

We can write a query to find the average number of helpful votes per reviewer, in order to understand which are the most appreciated ones.

```
db.reviews.aggregate([
  { $group: { _id: "$reviewerID",
    avg_votes: { $avg: {$ifNull:[{$toInt: "$vote"}, 0]} } }
   },
  { $sort: { avg_votes: -1 } }
]);
```

In this case we decided to use the `$ifNull` operator to handle the case in which the `vote` field is missing: in practice, we filled the missing values in this field by putting them equal to 0.
Results:

```
>_MONGOSH

> db.reviews.aggregate([
    { $group: { _id: "$reviewerID", avg_votes: { $avg: {$ifNull:[{$toInt: "$vote"}, 0
    { $sort: { avg_votes: -1 } }
  ]);
< {
    _id: 'A3HFPK3AMDNGBI',
    avg_votes: 96.5
  }
  {
    _id: 'A2XXBZPQT5EXHV',
    avg_votes: 81
  }
  {
    _id: 'A1VH6QWC6JZRAC',
    avg_votes: 74.6
  }
  {
    _id: 'A2E5GLJPQAMOBS',
    avg_votes: 67.16666666666667
  }
```

Figure 5: Query 3 (partial) results