MARKET BASKET ANALYSIS - FINDING FREQUENT ACTORS IN THE IMDB DATSET

Joel Kischkel
Algorithms for massive datasets
Università degli Studi di Milano
Student-ID: 941959
joeldavid.kischkel@studenti.unimi.it

June 4, 2021

ABSTRACT

Finding frequent items in data is a memory expensive task. Several algorithms were proposed how to deal with this problem under the limitation of memory. In this paper we demonstrate how to apply the SON algorithm in a MapReduce approach to find frequent item pairs in the IMdb dataset. Setting a minimum support of 140, we find four frequent pairs of actors. The most frequent pair are Ôtani and Onoe. Together they appear 237 times on the screen with a confidence of 0.91 and 0.26.

***K*eywords** SON algorithm · MapReduce · Market basket analysis

1 Introduction

Performing a market basket analysis can be beneficial for companies. This allows them to focus marketing and selling campaigns on specific products and reorganize stores to animate customers to buy more products together. Market basket analysis is not limited to just the retail sector. It can be used in other context as well. In this paper we using the market basket analysis to find actors actresses in the IMdb dataset who are playing frequently together in movies. Typically we are confronted with a lot of items when performing such an analysis. Therefore a crucial point in the implementation of such an analysis is to deal with the memory bottleneck. In our approach we use the algorithm of Savasere, Omiecinski, and Navathe also known as SON. This algorithm allows us to find association rules between actors by taking advantages of the MapReduce approach. The implementation is done in PySpark.

2 Data description and preprocessing

For the market basket analysis we use the IMDb dataset from Kaggle. [Ash21] This dataset is a collection of several datasets covering the topics, movies, persons, ratings, etc.. For performing the market basket analysis we are interested in the following files:

- title.basics
- title.names
- title.principals

The file title.basics is a collection about information about titles of movies, tv episodes, ect. For a better understanding we will call this file movie. Each movie in this file is identified by an unique id called tconst. A record has several attributes as title type, primary title, genre, ect.. For our analysis we are solely interested just in the id, title type and primary title. We restrict our selection in data points where the title type is equal to $movie$. Very similar to the title.basic file is title.names. This file is a collection about persons who participated in movies, videos, ect.. Each person has an unique identifier called nconst. Again each record has several attributes like primary name, year of birth or death. For

our purposes we will use the attributes nconst and the primary name. The third file title.principals will provide the link between the two previous files. This file is a collection of the ids of movies and persons and their job specific to the movie. We are just interested in the connection between tconst and nconst and that the corresponding category is actor or actress. Performing a SQL Join operation on the movie and principal file and again on the name file allows us to retrieve all the persons participated in a movie. Using the category actor or actress in the principal file we filter out all the persons which are not an actor or actress from the SQL query. With this approach we can select all the actors and actresses who played in movies. It would be enough to just select movie id and name it, but for a meaningful interpretation of the data we also select the movie title and the name of actors. The applied selection returns 1,692,821 rows. Each row is now a record including movie id, person id, title and name. To control that just actors and actress are selected also the category is included in the result.

We are interested in finding actors that played in a lot of movies. A high number of appearances in movies could indicate that an actor may are frequent together with some other actors. The following figure **??** shows the diagram of the 10 most frequent actors in the dataset. The actor with the most appearances on the screen is Brahmanandam with 798 times. If we consulate Wikipedia we can confirm this and we also see that he holds the guiness world record with this number of movies. [Wik]

The word cloud in figure 2 tells us that if we expand the counting over 200 actors, we see that some actors are appearing very often while a majority of the actors are playing much fewer in movies.

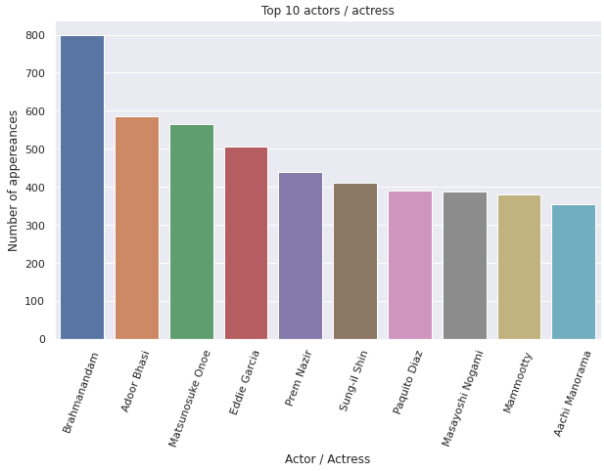

Figure 1: Frequency of the 10th top actors by appearance

Appearances	Name
Brahmanandam	798
Adoor Bhasi	585
Matsunosuke Onoe	565
Eddie Garcia	506
Prem Nazir	438
Sung-il Shin	411
Paquito Diaz	391
Masayoshi Nogami	387
Mammootty	380
Aachi Manorama	355

Table 1: Table of the appearances

3 A-Priori Algorithm

The A-Priori algorithm is well known for finding frequent items. The algorithm is designed to find in k steps sets of frequent items of the size k. The conceptual idea of the algorithm is simple. In the first step we just count the items. If

Figure 2: Word cloud

their frequency does not meet an predefined threshold, called support, an item will be filtered out. We interpret the support as the number of times we want to find this item at least in the frequent item set. In the next step we build a candidate set based on the frequent items. This set is a a pair of items of all possible combinations of the frequent items we found before. Now is counted again the frequency of the items and we also filter out non frequent pairs. After this round we are left with a set of frequent pairs. We repeat this computation till we achieved the predefined number of items inside a frequent item set. [AS94]

The main problem of the A-Priori algorithm is the second pass. Here we have to build the frequent candidate pairs. If the support threshold was set very low and we have a lot of frequent items, we are left with a lot of candidate pairs. We can imagine that such a scenario can lead into thrashing of the algorithm if the main memory is exceeded. Our goal is to use an algorithm that can also handle datasets which cannot be stored completely in the main memory. [JL14, p.218]

4 SON Algorithm

To overcome the problems with the memory issue, we are using the algorithm of Savarese, Omiecinski and Navathe. While the A-Priori relied on the idea to load all the data into the memory, this algorithm is splitting the data into partitions and thus reduces the memory usage. For our approach we use the SON algorithm and execute it parallel using the MapReduce framework. Data will be split into n chunks and on each of the chunks the A-Priori algorithm will be applied to find frequent item sets.

The map function returns for each single item in the chunk the $key - value$ pair (item, 1). Of course we expect that some item or in our context actor appear several times. We then perform a $reduce$ operation on the chunk based on the keys. We sum the values up for each key and filter out results which are lower than the adjusted support on the chunk. The adjusted support is nothing else than the support multiplied with the size of the chunk. If we have a chunk of size 0.1 and we assume that the sum of all chunks is equal to 1, using a support of 100, would give us the adjusted support of 10. This means we $filter$ out all the items that are not appearing at least 10 times in this chunk. We then form a $union$ over all the results and create a global frequent item set. Since we are not interested in the first round how frequent an item is we consider for the union just the $distinct$ value. Also in the second round we use the file split in chunks. Before finding frequent pairs we build a local set of frequent pairs on each worker based on the frequent items we found before. On each chunk we now using now again a map function which returns the pair (item, 1) if a candidate pair out of the local frequent item pair set is a subset of our data in this worker. We then $reduce$ again on each worker the result an we are left with a tuple (item, v). The v now stands how often an item was counted to be frequent. Forming again the $union$ over all the chunks we now filter out all the pairs where v is lower than the support. [JL14, p. 229 - 230] The figure 3 shows the conceptual idea of this approach.

5 Experiments

The support is set to 140 which means we are looking for actors / actresses who are appearing at least 140 times together in movies. We are using 3 chunks to perform the computation. This means the SON algorithm has an adjusted support of 46.66.

Running the algorithm on the data let us left with a set of 4 frequent actors / actresses. The following table 2 shows the result. With a slightly modified version of the algorithm, we let return the support of each of the actors. Combining the support of each actor in table 3 with the frequent item pairs in the 2 we can compute the confidence. Table 4 shows the confidence. We see that the distribution of the confidence is not uniform. We have for example the case between

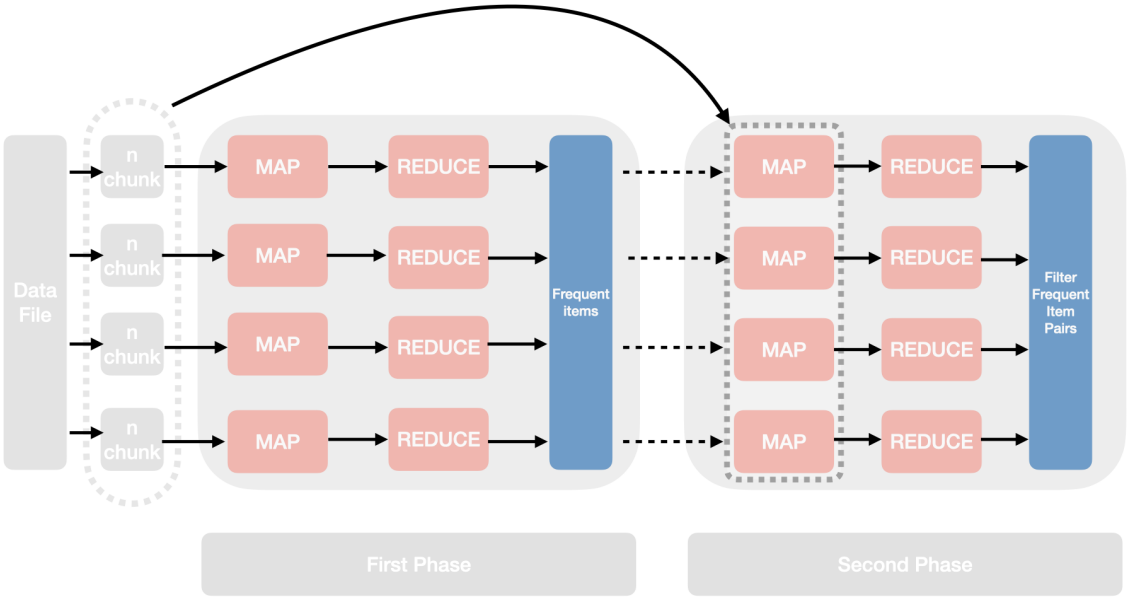

Figure 3: The SON algorithm in a MapReduced framework

{nm2082516 (Ôtani) -> nm0648803 (Onoe)} with a confidecne of 0.913. But if we check the opposite direction we find that the confidence between {nm0648803 -> nm2082516} is 0.260. This means that if we are looking for movies where the actor nm2082516 is playing a role, it is very likely that also the actor nm0648803 appears. But we cannot conclude this for the opposite case.

Frequent Item pairs	Number of appereances toegther
{nm0419653, nm0006982}	162
{nm0046850, nm0006982}	169
{nm2082516, nm0648803}	147
{nm0623427, nm0006982}	237

Table 2: Result of the SON algorithm

Actor	Support
nm0419653	303
nm0046850	348
nm2082516	161
nm0623427	438
nm0006982	585
nm0648803	565

Table 3: Support of each actor

6 Conclusion

The SON algorithm is an effective method to find frequent item pairs in data that is too large to be loaded into the memory. The downside of this method is, that the time to execute the algorithm is very long due to the I/O operations. We have seen it is not just sufficient to just find frequent pairs, instead we also have to consider the confidence of it into our analysis. Crucial for the market basket analysis is to choose the size of the minimum support. A low value will not just lead to many items, it will also increase the runtime of the algorithm. On the other side a to high value could eliminate lot of items and reduce the effectiveness of the analysis.

Frequent Item pairs	Confidence score
{nm0419653 -> nm0006982}	0.534
{nm0006982 -> nm0419653}	0.276
{nm0046850 -> nm0006982}	0.485
{nm0006982 -> nm0046850}	0.288
{nm2082516 -> nm0648803}	0.913
{nm0648803 -> nm2082516}	0.260
{nm0623427 -> nm0006982}	0.541
{nm0006982 -> nm0623427}	0.405

Table 4: Confidence

7 Disclaimer

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

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, page 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[Ash21] Ashirwad. IMDb Dataset, 2021. accessed: 2021-05-17.

[JL14] Jeffrey D. Ullman Jure Leskovec, Anand Rajaraman. *Mining of Massive Datasets*. Cambridge University Press, 2014.

[Wik] Wikipedia. Brahmanandam. Accessed: 2021-05-14.