Market-Basket Analysis

Algorithms for Massive Datasets Project

Matteo Farè ID: 989345

University of Milan, Department of Computer Science.

author: matteo.fare1@studenti.unimi.it;

Abstract

Implementation of a system for finding frequent itemsets (market-basket analysis) working with a movies related dataset, where each movie represents a basket and the associated actors are the items. The task is performed through the use of the A-priori algorithm and the algorithm of Park, Chen, and Yu (PCY).

1 Introduction

Market-basket analysis was originally employed by retailers to find out items relationship among the customers transactions, with the main goal of reveling products that are often brought together, **optimizing product placement** and proposing targeted offers to clients.

Today, this technique is employed in a variety of applications, such as performing **fraud detection**, understanding customer behavior under different conditions, and in **healthcare**, where it is used to identify the relationship between different diseases and symptoms. In general terms, it represents a **many-to-many association** between two kinds of entities.

This study focuses on finding frequent itemsets by working on a dataset that collects various information about **movies**, treating movies as baskets and **actors** as items. to achieve the intended goal, two algorithms were implemented from scratch: the **A-priori algorithm** and the algorithm of **Park, Chen, and Yu (PCY)**.

2 Dataset

The dataset used to carry out the project contains informations related to the web platform known as **Letterboxd**[1], a social network about the discussion and discovery of films. As shown in **Figure** [1], the dataset on **Kaggle**[2] consists of multiple `csv` files, each focused on an aspect related to the world of cinema (actors, genres, movies etc.), along with a folder containing `jpg` files of movie posters.

```
name                   size
-------------------   -----
actors.csv            186MB
countries.csv          10MB
crew.csv              154MB
genres.csv             17MB
languages.csv          27MB
movies.csv            243MB
posters.csv            90MB
posters/1000001.jpg    31KB
...                     ...
```

Fig. 1: Partial synthetic view of the contents of the dataset

Since the objective of the project is to identify frequent itemsets, treating movies as baskets and actors as items, the focus was placed exclusively on the `actors.csv` file, considering only the `id` column, which represents the movie identifiers, and the `name` column, which lists the actors who participated in each movie.

[1]Letterboxd web page: https://letterboxd.com
[2]Letterboxd dataset: https://www.kaggle.com/datasets/gsimonx37/letterboxd/data?select=actors.csv

At this point, the actor names are mapped to unique integer identifiers using the `mapping_actors_to_ids` function. This function generates two dictionaries: one for encoding strings to integers and another for decoding integers back to strings. The dataset is then grouped by `id`, resulting in a structure where each movie identifier is associated with a set of actors, as briefly illustrated below in Figure 2.

```
id       basket
-------  ----------------------------------------------------
1000001  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, ...}
1000002  {168, 169, 170, 171, 172, 173, 174, 175, 176, ...}
```

Fig. 2: Synthetic view of the pre-processed dataset

In order to speed up the process during the execution of the algorithms, only a section of the entire dataset was used, specifically **10%**.

3 Algorithms Implementation

3.1 Utils: support functions used mid-process

- **Filtering frequent items**: the `filter_frequent_items` function filters items based on a **support threshold**, retaining only those items that appear frequently enough in the dataset.

- **Hash table and Bitmap**: employed exclusively in the PCY algorithm implementation, the `pcy_hash` function is used to map item pairs to buckets. the **hash table** obtained, containing the frequency of each bucket, is then summarized by the `generate_bitmap` function into a **bitmap**, indicating which buckets meet the defined support threshold.

3.2 A-priori algorithm

Introduced in 1994, the A-priori algorithm is the fundamental algorithm in the field of data mining for extracting **frequent itemsets** from a dataset. The process is based on the principle of **monotonicity**, which states that if an itemset is frequent, then all its subsets must also be frequent. The algorithm is performed in two steps on the data to identify itemsets that meet a support threshold.

The **first pass** of the algorithm is carried out by the `apriori_first_pass` function, which counts the occurrences of each unique actor name in the dataset. The result is then filtered by the function `filter_frequent_items`, going from the set of candidates to the set of **frequent items**.

The process then continues with the **second pass** performed by the `apriori_second_pass` function, during which all combination pairs are formed and their occurrences are counted, and finally the last support threshold filter is applied to pass from the set of **candidate pairs** to the set of **frequent pairs**.

3.3 PCY algorithm

The algorithm developed by Park, Chen, and Yu is an enhanced version of the A-priori algorithm designed to reduce the number of candidate pairs that need to be counted, using a hashing technique to summarize potential candidate pairs into a bitmap.

In the **first pass** of the PCY algorithm, performed by the `pcy_first_pass` function, the frequency of each unique item is counted, similar to the initial step of the A-priori algorithm. Additionally to this last one, all combination pairs of items are formed and inserted into a **hash table**, maintaining for each bucket the sum of the counts of all the pairs that hash to it. If the bucket's count is less than the support threshold no pairs of item that hashes in it can be considered frequent, defining what is called an **infrequent bucket**. The hash table is then summarized into a **bitmap** using the `generate_bitmap` function, which maps each frequent bucket to a value of 1 and infrequent buckets to 0. Finally, the **second pass** focuses on counting the frequencies of candidate pairs that map to the frequent buckets in the bitmap, using this last one to filter out infrequent pairs early.

3.4 Association Rules

An **association rule** I $\rightarrow$ j, with I representing a set of items and j a single item, could be described in a basic way as a kind of **if-then implication**, which states that if all the elements of I appear in a baskets then it is likely that j also appears. This aspect is developed implementing the `compute_association_rules` function, through which for each frequent pairs, **confidence** and **interest** are found.

- **Confidence**: measures the likelihood that an item j is present in transactions where item I is also present. It is calculated as

$$\text{Confidence}(I \rightarrow j) = \frac{\text{Support}(I \cup \{j\})}{\text{Support}(I)}.$$

  High confidence indicates a **strong predictive relationship** between items, showing how strongly the presence of one item implies the presence of another.

- **Interest**: helps to understand how much more or less likely it is for item j to appear in transactions that contain item I, compared to what would be expected if items were unrelated. It is calculated as

$$\text{Interest}(I \rightarrow j) = \text{Confidence}(I \rightarrow j) - \text{P}(j).$$

  If the interest value is **positive** (usually above 0.5), it means that the presence of I makes it more likely that j will appear. If the interest value is **negative**, it means that the presence of I makes it less likely that j will appear.

4 Scalability

After working on a small sample of the dataset, test runs were performed on **increasing samples** of the dataset to verify the algorithms behavior. The performed evaluation involves measuring the **execution time** of each algorithm and analyzing how it scales with varying data sizes. To carry out this task two functions are employed. The `measure_time` function, which records the time taken by the specified algorithm, and the `measure_scalability` function, which checks how the execution time changes based on the size of the data sample, corresponding to the following values: *100000*, *200000*, *300000*, and *600000*.

The Figure 3 below shows the result of this process.

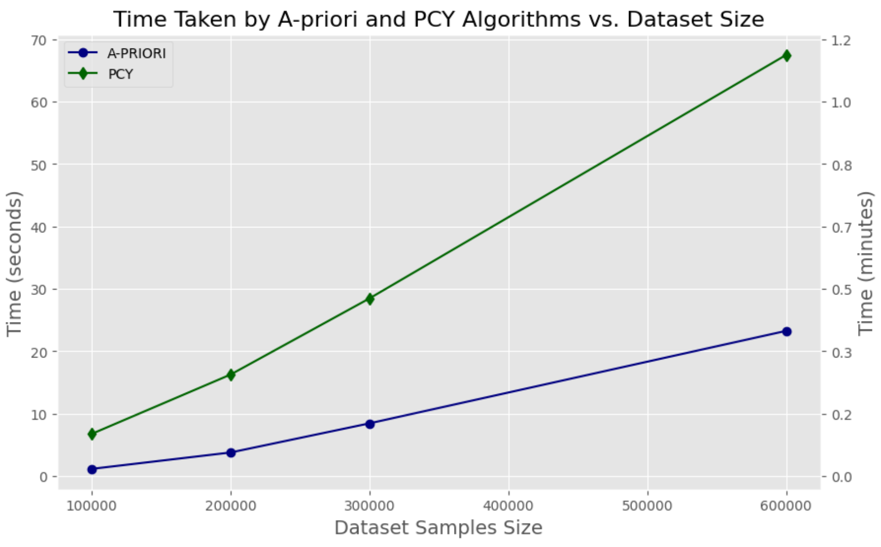

Fig. 3: A-priori and PCY Algorithms: Time vs. Dataset Size

In this case, as shown in the generated graph, given the empirical choices of a **support threshold** equal to 10 and a **number of buckets** of 50123, over the different samples of the dataset, the A-priori algorithm takes a shorter time to execute than the PCY algorithm. This could be due to the fact that, as pointed out in [1], the algorithm of Park, Chen, and Yu is more effective than A-priori only if it allows to avoid counting at least 2/3 of the pairs of frequent items, if this does not happen it is not possible to gain from using PCY instead of A-priori. Otherwise it could be related to the low number of both frequent items and frequent pairs. For example, working with the sampled dataset and the above parameters, 4490 frequent items and 75 frequent pairs were found, respectively. Having to generate fewer pairs, the A-priori algorithm could therefore benefit in terms of time because it avoids the additional **computational overhead** involved in PCY, such as the creation and processing of hash tables and bitmaps.

5 Conclusion

In this project, the **A-priori** and **PCY** algorithms were implemented and compared, with the aim of finding frequent itemsets within a movie-related dataset. Treating movies as baskets and actors as items, both algorithms were used to discover patterns of actor co-appearances in movies.

The obtained results show that the A-priori algorithm outperforms the PCY algorithm in terms of execution time given the empirical parameters used in the experiments.

References

[1] J. Leskovec, J.U. A. Rajaraman: Mining of Massive Datasets, pp. 231–232. Cambridge University Press