

# Market Basket Analysis

Master in Data Science

Alberto Bertoni 983833

Massimo Cavagna 9838??

[Github repository](#)

July 8, 2021

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Dataset</b>	<b>4</b>
<b>3</b>	<b>Data structure</b>	<b>4</b>
<b>4</b>	<b>Preprocessing</b>	<b>6</b>
<b>5</b>	<b>Algorithms applied</b>	<b>6</b>
5.1	A-priori . . . . .	7
5.2	PCY (single hash) . . . . .	8
5.3	SON . . . . .	8
5.4	Toivonen . . . . .	9
5.5	Implementation . . . . .	9
<b>6</b>	<b>Scaling of proposed solutions</b>	<b>10</b>
<b>7</b>	<b>Experiments</b>	<b>10</b>
7.1	Size of candidates with different algorithms . . . . .	10
7.2	Size of candidates with different hash size PCY . . . . .	10
7.3	Time at change of threshold . . . . .	10
7.4	Size of negative border . . . . .	10
<b>8</b>	<b>Results</b>	<b>10</b>

# 1 Introduction

The purpose of this paper is to present some of the techniques used in order to perform the so called "market/basket" analysis for a huge amount of data. At first these techniques were exploited for the analysis of purchases in markets, trying to find some relationships among the goods bought by customers. The idea behind these algorithms is to find associations between goods so that can be claimed that if a customer buy item A he is also likely to buy item B and vice versa. This concept could be extended to association between sets of goods (not only single item pairs) and to generic items instead of just goods, so that, in the end, the aim of the algorithms that will be presented is to find frequent sets of items in all the baskets available. In particular, three main algorithms will be implemented:

- A-priori:
  - base
  - PCY
- SON
- Toivonen

Once the frequents sets are found, it is also important to check if all the items in one of these sets are actually associated one another: indeed, considering the environment these techniques come from, we could find that some goods, such as "milk" or "bread" are always frequent, but it cannot be claimed that there is a actual relationship with all the other items in the same frequent set, since it will be bought independently from the other.

## 2 Dataset

The dataset in analysis is called "IMDb dataset" (version 6) created by data collected from the homonymous site IMDb. This dataset contains information about movies, their ratings and workers that have taken part in them. The data is divided into several files to simplify the analysis over specific aspect.

- *title.akas.tsv* contains informations about the localized version of the movies.
- *title.basics.tsv* contains general information about the movies, not influenced by the localization.
- *title.principals.tsv* contains information about the cast and the crew for each movie.
- *title.ratings.tsv* contains the IMDb rating informations about the movies.
- *name.basics.tsv* contains informations about the cast and the crew.

The aim of the analysis presented is to find sets of actors that have frequently worked together so only a few of these datasets will be used: in particular *title.principal.tsv* from which is possible create baskets of actors for every movie and *name.basics.tsv* from which is possible retrieve the names of the actors from their IDs.

## 3 Data structure

*title.principal.tsv* is a tsv file with the subsequent structure:

- *tconst* (string) is an alphanumeric identifier of the movie.
- *ordering* (integer) is a number used to uniquely identify rows for a given movie.

- *nconst* (string) is an alphanumeric identifier of the cast/crew person.
- *category* (string) is the role of that person in the movie (a person can do different roles in the same movie).
- *job* (string) is a further specification of the role (can be empty, with the symbol "\N").
- *characters* (string) is the name of the character played (can be empty, with the symbol "\N").

There are 36.499.704 rows for 5.710.740 different movies.

Only rows with an actor are considered, so the analysis is done over 14.830.233 rows.

Movies without any registered actor need to be filtered, reducing the number of movies to 3.602.200.

In the end the number of different actors in this dataset is 1.868.056.

The size of *title.principal.tsv* is 1.6 GB.

*name.basics.tsv* is a tsv file with the subsequent structure:

- *nconst* (string) is an alphanumeric identifier of the cast/crew person.
- *primaryName* (string) the name of the person.
- *birthYear* (YYYY) year of birth.
- *deathYear* (YYYY) year of death.
- *primaryProfession* (array of strings) the top 3 profession of the person.
- *knownForTitles* (array of tconst) movies the person is known for (can be empty).

There are 9.711.022 rows in this file with only 3.625.895 people marked as actor/actress.

## 4 Preprocessing

The data used need to be preprocessed in order to make them suitable for the algorithms that will be presented.

The main dataset used is the *title.principals.tsv*, since it contains all the information needed: the main idea in this phase is to group all the rows by the *tconst* (movie id), and take all the *nconst* that refers to an actor (*category*).

Once this operation is performed, the resulting data are organized in sets (or baskets) of actors' IDs (one for each movie) so the only action left to perform is to change them, with an integer instead of a string: indeed, all the IDs are in the form of "nm" followed by a number, so it is rather easy to change them just cutting the first part.

In the end the completely transformed data are saved on mass memory.

## 5 Algorithms applied

In this section are presented the algorithms studied and implemented. The main idea behind them is to obtain the list of frequent itemsets (sets of actors in our case) exploiting the property of monotonicity in order to reduce the number of candidates frequent sets: the theoretical assumption is that if an itemset is frequent then all its subsets are frequent.

All the algorithms described are designed in order to work with a huge amount of data (big enough that can not fit in memory), so in all the sections will be given particular attention to the number of mass memory

access performed, which makes them slower.

## 5.1 A-priori

The first approach implemented is the basic A-priori algorithm. This algorithm can be outlined by two phases:

- Construction: in this phase all the sets of dimension  $k$  are built from the frequent ones of dimension  $k-1$ . For implementation purposes, it will be shown that this phase contains a sorting pre-phase in which all
- Filtering: in this phase all the new (candidate frequent) sets obtained are filtered by their frequency, so that if they do not reach a given threshold they are discarded.

The algorithm starts counting the number of occurrences of each item in the dataset, so in this phase the main memory is partially filled and the first scan of the data is performed. Once all the frequencies are found, the singletons are filtered using a predefined threshold.

Now, the algorithm scans again the dataset and for each basket builds a list of all the combinations of dimension 2 with items contained, discarding the ones that contains a non frequent singleton, then the memory is filled also with the pairs and their frequency. Again, the filtering step is performed.

From this point, the algorithm looks for all the tuples of dimension  $k \geq 2$  and it becomes important the sorting pre-phase mentioned before: all the frequent tuples of dimension  $k-1$  are "joined" if they have in common the first  $k-2$  elements, then the whole dataset is scanned and each basket is ordered and all the possible tuples of dimension  $k$  are built so that it is possible to check if a tuple is present or not in that basket.

These operations continue until it is not possible to find a single tuple of dimension  $k$  that is frequent.

## 5.2 PCY (single hash)

An improvement of this algorithm is the PCY (Park, Chen, Yu) that exploit the free space present in memory while counting the frequency of itemsets of size  $k$ , creating an hashmap in which counts, in the position satated by the hash of the itemset of size  $k+1$ , the number of times the latter occurs in the baskets. Note that it is possible to have collisions, but this approach aims to lower the number of itemsets that has to be saved in memory and for which is necessary to count the frequency, instead of directly delete each non frequent itemsets.

The hashmap is compressed into a bitmap for the purpose of saving space and permitting to create only itemsets marked as possible frequent on this map. To reach frequent itemsets of size  $k$  is necessary to pass  $k$  times over data, as the previous algorithm, but is possible reduce the candidates at every iteration.

## 5.3 SON

This algorithm is developed to parallelize the computation of frequent itemsets, working over chunks (partition) of the data. It's not a proper sampling algorithm because it will work on every chunk and, in the end, the totality of data will be elaborated.

It'a a way to use the previous implementations (A-priori/PCY) in a distributed form, to reduce the single workload over a chunk and parallelize the computation, and then put together the results. Is important en-



force appropriate adjustment to run the A-priori over every chunk, like the repositioning of the threshold that divide frequent itemsets from not frequent.

## 5.4 Toivonen

This algorithm, as the previously, exploits the A-priori or PCY implementation to obtain the frequent itemsets of a sample of the complete data (providing the adjusted threshold). This algorithm exploits the notion of **Negative Border** that once created will be used to check if the frequent itemsets in the sample are frequent even globally. The negative border is built extracting every itemsets that is not frequent but that have frequent immediate subsets. The algorithm, operating just over the sample can say if the candidate frequent itemsets are the right ones even for the totality of data, with a single passage over the complete dataset.

## 5.5 Implementation

These algorithms have been implemented with python using a sequential way for the PCY and the MapReduce paradigm to apply the PCY according the SON and Toivonen philosophy.

## 6 Scaling of proposed solutions

## 7 Experiments

### 7.1 Size of candidates with different algorithms

Over the whole dataset

Apriori, PCY, SON apriori, SON PCY

### 7.2 Size of candidates with different hash size PCY

### 7.3 Time at change of threshold

Apriori, PCY, SON apriori, SON PCY IMPORTANT In SON it's the worst time of a chunk and the time of aggregation and filtering

### 7.4 Size of negative border

With different size of sample With different threshold

## 8 Results

*I/We declare that this material, which I/We now submit for assessment, is entirely my/our 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/our work. I/We 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/us or any other person for assessment on this or any other course of study.*