Association Rule Mining for Earthquakes Prediction

Project for the course "Scalable and Cloud Programming", A.Y. 2021/22

Stefano Bucciarelli Gianluigi Carrozzo Melania Ghelli

Dipartimento di Informatica
Università di Bologna

Project objectives

- Obtain Association Rules from Earthquake dataset for predictive purposes
- Develop different versions of Apriori algorithm
- Compare the algorithms in order to verify if the MapReduce version has better performances

Table of Contents

The problem

- The purpose was to realize an application of Data Science for Sustainability
- Given a dataset of earthquake data the goal is to extract information for predictive purposes
- Association Rule Mining is one of the possible techniques one can use for this application
- In order to do so we followed the approach proposed in the reference paper [6] by exploiting the MapReduce paradigm

Data

- Data have been obtained from the U.S. Geological Survey (USGS) Earthquake Catalog
- The catalog can be accessed at https://earthquake.usgs.gov/earthquakes/search/

The technique

Association Rule

Given a database D of transactions where each transactions $T \in D$ is a set of items, an association rule $X \Rightarrow Y$ expresses that whenever a transaction T contains X probably T contains Y as well.

- The goal of Association Rule Mining technique is to find rules that express that "whenever X occur, Y will happen as well with a certain probability".
- This probability is called *confidence*, and it is defined as the percentage of transactions containing both X and Y with regard to the overall transactions containing X only.

Apriori algorithm

This algorithm decomposes the problem of discovering association rules into two subproblems:

1. **Find all itemsets.** An *itemset* is a set of items that has *support* above a treshold. The *support* is the number of transactions that contain the itemset.
2. **Use the largest itemset to generate the rules.** This is done by obtaining all non-empty subsets of the largest itemset I. Then for each subset s the outputted rule is $s \Rightarrow I - s$. Only rules that satisfy a minimum *confidence* are kept.

K-means algorithm

- When working with large dataset, looking for frequent itemsets means that only data that are often repeated will be taken into account
- In an Earthquake dataset the most significant events (i.e. seismic events with high magnitudo) will likely be present in a small percentage
- In order not to ingore those events, before generating frequent itemsets data are divided into clusters based on their magnitude
- For clustering we choose to use K-means algorithm from Spark MLlib: https://spark.apache.org/docs/latest/ml-clustering.html

KMeans algorithm

Here is how the algorithm works:

1. Given a number k of clusters, randomly choose k items that will serve as *centroids*
2. Repeat the following steps until centroids assignment is stable:
   1. **Assign** each data point to the closest centroid
   2. **Recalculate** centroids for each cluster

KMeans algorithm

Number of clusters k has been decided by using the elbow method:

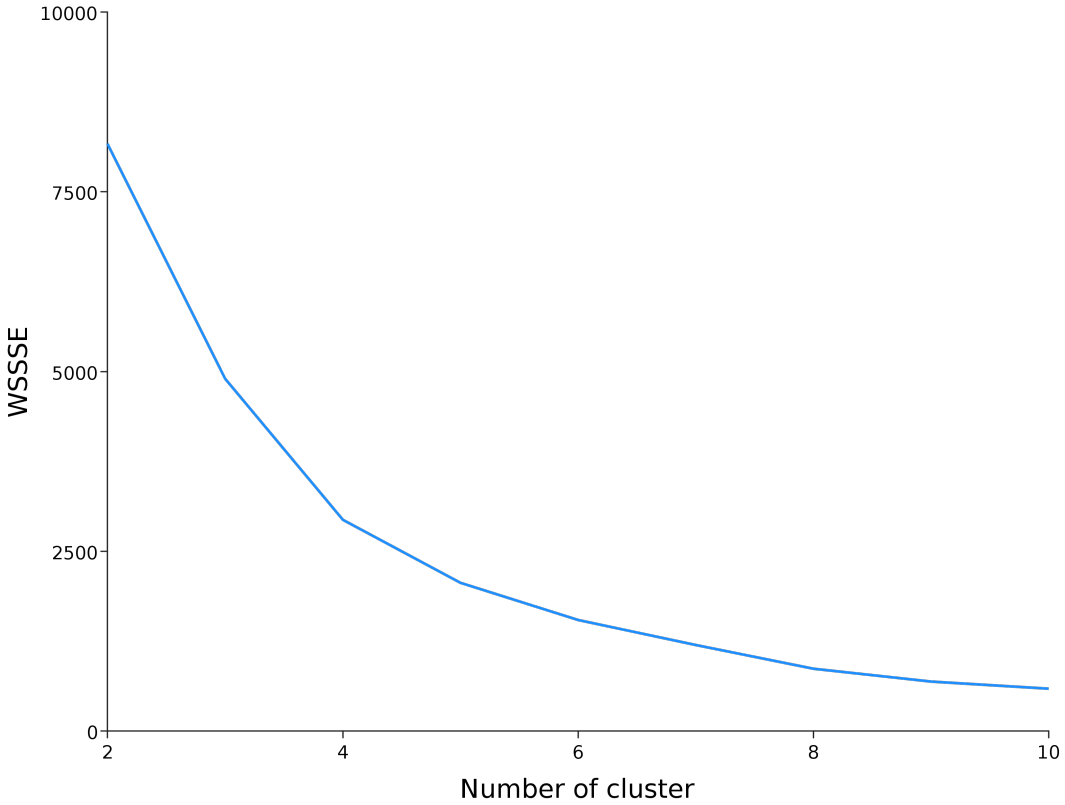


Table of Contents

Sequential Apriori

This algorithm uses Scala collections and their functions.

1. **Frequent Itemsets Research**
   - At each step i it takes i-dimensional subsets of item and save them in a `Set`
   - Maps them together with their occurrences and filter the ones that satisfy the minimum support
   - Performs a *prune* operation by filtering all the subsets of each itemset that satisfy the minimum support condition as well
   - Save the left itemsets (Frequent Itemsets) in a `Map` together with their support
2. **Association Rules Generation**
   - For each frequent itemset it generates all its subsets. Each subset will form a candidate rule $s \Rightarrow l - s$, where s is the found subset and l is the itemset
   - Rules are filtered: only the ones that satisfy a minimum confidence are kept

MapReduce Apriori (1/2)

Frequent Itemsets Research

1. **Phase 1**
   - For all transactions, map each item i in the pair $(Set(i), 1)$
   - Compute a reduce step to obtain $(Set(i), c)$ pairs, where c is the number of times i occurs in transactions
   - Filter the pair with c greater than min support value
2. **Phase 2**

   At each step k, perform frequent itemsets ($SetL_k$) with size k:
   - Starting from $SetL_{k-1}$ compute $SetC_k$, i.e. every possible itemset with size k
   - For each transaction T and each itemset S in $SetC_k$, if S is subset of T, map in $(S, 1)$
   - Compute a reduce step to obtain (S, c) pairs, where c is the number of the occurrences of i in transactions
   - Filter the pair with c greater than min support value

   The iteration stops when an empty $SetL_k$ or $k > 4$ is found

MapReduce Apriori (2/2)

Association Rules Generation

- Find all subsets of frequent itemsets.
- Count the number of occurrences of each subset (map and reduce steps).
- Generate association rules (S, I-S, C) where
  - I is a frequent itemset
  - S is a subset of I
  - C is the confidence, i.e. $\frac{support\ of\ I}{support\ of\ S}$
- Filter the rules with confidence greater than min confidence value

MapReduce Apriori with Tail Recursion (1/2)

Frequent Itemsets Research

1. **Phase 1**
   - Map each item i, in each transaction in $(Set(i), 1)$
   - Perform a reduce that calculates a sum on the second element of the pair, and evaluating $(Set(i), c)$ where c is the number of the occurrences of i in transactions that is greater than min support
2. **Phase 2**

   Evaluate $SetL_k$ with tail recursion at step k :
   - Evaluate $SetC_k$ taking all subsets of the previous step $SetL_{k-1}$ and perform all possible unions between any two of them and filter the ones with cardinality k
   - For each transaction, filter itemsets S in $SetC_k$ that are subset of transaction and map in $(S, 1)$
   - Perform a reduce that makes a sum on the second element of the pair, and evaluating (S, c) where c is the number of the occurrences of S in transactions that is greater than min support

   The recursion stops when it doesn't find an itemset that satisfies the requirements or when $k > 4$

MapReduce Apriori with Tail Recursion (2/2)

Association Rules Generation

- For each itemset get all the subset, and consider the subset as the antecedent of the AR and the remaining (the difference between the itemset and the subset) as the consequent
- The confidence is evaluated by dividing the support of the itemset by the support of the antecedent
- Filter the rules with confidence greater than minimun value

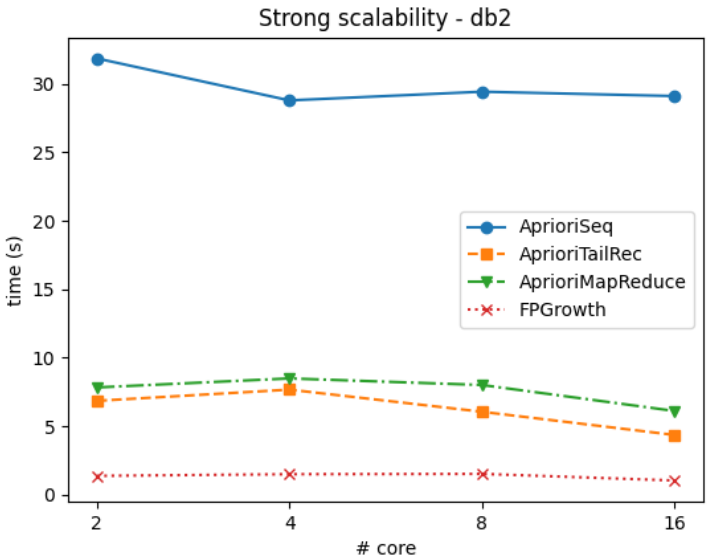
Table of Contents

Evaluation

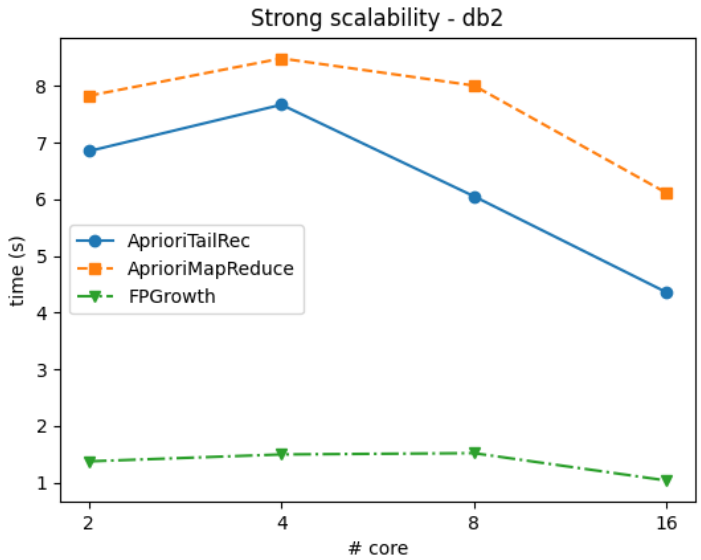
- In order to evaluate the implemented algorithms we compared the association rules we got with the ones outputted by Spark MLib FP-growth.
- Since FP-growth generates association rules by considering frequent itemsets of all dimension, in order to have a fair challenge we made our Apriori implementations do the same. Classic Apriori generates association rules for the biggest itemset only.
- Results shows that the rules outputted by FP-growth are outputted by the other three algorithms as well. Apriori algorithms also produce rules where the consequent has more than one element, thus being more informative than the other.

Strong Scalability

- **input data**: 200.000 samples of the dataset
- **cluster configurations**: 2 workers and 2 cores, 2 workers and 4 cores, 4 workers and 8 cores, 4 workers and 16 cores. Workers use either n1-standard-1, 2 or 4 machines, whilst master use a n1-standard-4 processor



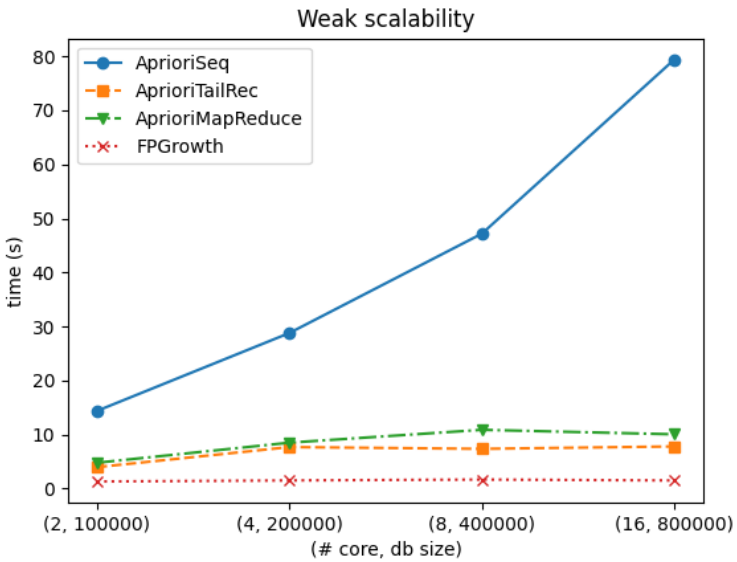
(a) Strong scalability tests



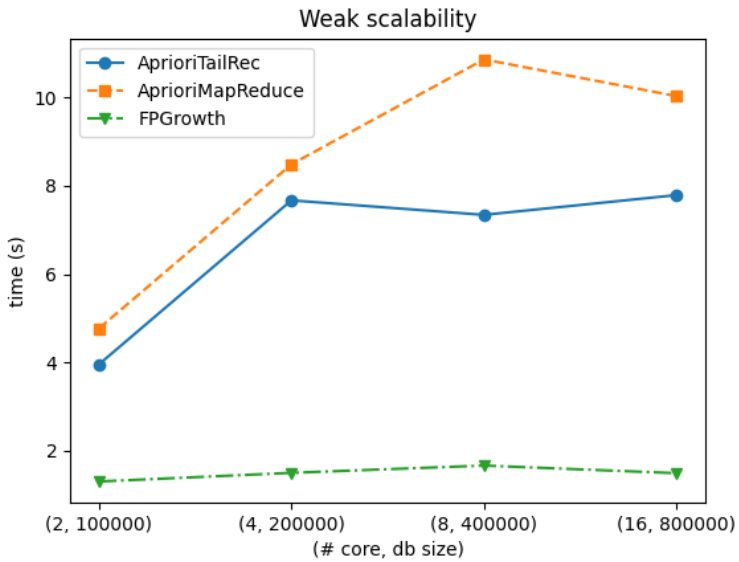
(b) Strong scalability tests - without Sequential Apriori

Weak Scalability

- **input data**: we used a growing number of samples, increasing them when number of available cores increased. Samples dimensions are: 100k, 200k, 400k and 800k
- **cluster configurations**: same as strong scalability



(a) Weak scalability tests



(b) Weak scalability tests - without Sequential Apriori

Conclusion

- The scalability tests show that the implemented algorithms scale well, following the same behaviour as the library algorithm.
- Map-Reduce algorithms allow a significant improvement in terms of performances with respect to the sequential one. The version that introduces Tail Recursion has slightly better performances than the Map-Reduce one.
- FP-growth still has better performances than Apriori, as expected.

References I

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. "Mining association rules between sets of items in large databases". In: *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*. 1993, pp. 207–216.

[2] Rakesh Agrawal and Ramakrishnan Srikant. "Fast Algorithms for Mining Association Rules". In: 1998. URL: https://api.semanticscholar.org/CorpusID:7736589.

[3] Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. "Algorithms for Association Rule Mining — a General Survey and Comparison". In: *SIGKDD Explor. Newsl.* 2.1 (June 2000), pp. 58–64. ISSN: 1931-0145. DOI: 10.1145/360402.360421. URL: https://doi.org/10.1145/360402.360421.

References II

[4] Ming-Yen Lin, Pei-Yu Lee, and Sue-Chen Hsueh. "Apriori-Based Frequent Itemset Mining Algorithms on MapReduce". In: *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*. ICUIMC '12. Kuala Lumpur, Malaysia: Association for Computing Machinery, 2012. ISBN: 9781450311724. DOI: 10.1145/2184751.2184842. URL: https://doi.org/10.1145/2184751.2184842.

[5] MS Mythili and AR Mohamed Shanavas. "Performance evaluation of apriori and fp-growth algorithms". In: *International Journal of Computer Applications* 79.10 (2013).

[6] U. Nivedhitha and Krishna Anand. "Development of a Predictive System for Anticipating Earthquakes using Data Mining Techniques". In: *Indian Journal of Science and Technology* 9 (Dec. 2016). DOI: 10.17485/ijst/2016/v9i48/107976.