Synthetic Dataset Generator

Nicolò Merzi
nicolo.merzi@studenti.unitn.it

Francesco Battista
francesco.battista@studenti.unitn.it

ABSTRACT

In this paper, we propose a method for generating a synthetic dataset of tree structures of complex objects where the elements that make up these objects appear with a certain frequency. This synthetic dataset provides an experimental tool to verify the accuracy and efficiency of data mining algorithms that are able to extract frequent patterns from complex structures.

Categories and Subject Descriptors
[**Information Systems**]: D*ata Management System, Database design and models, Relational Database Model.*

General Terms
Algorithms, Documentation.

Keywords
Synthetic dataset, Binary Tree, Big Data.

1. INTRODUCTION

Frequent Pattern Mining (FPM) refers to a significant class of explanatory mining tasks, namely those dealing with extracting patterns from massive databases representing complex interactions between entities. A large number of applications produce data records that are not independent from each other but are sequences where the appearance of one record depends on the appearance of another. Furthermore, each record can also consist of a different number of components (or characteristics). This is what we refer to when we talk about **complex objects**: **an object composed of several elements**.

The focus of this paper is creating a synthetic dataset that mimics the situation we just described: a dataset of complex objects organized in binary trees where the elements inside these objects appear with a certain frequency (in order to simulate the dependency between those elements).

2. STRUCTURE

The dataset has the following structure:

Record_id	Transaction_id	Action
1	1	[4,7,4] [5,9,8]
2	1	[5,9,8] [1,6,5]
3	1	[4,7,4] [5,5,9]
4	1	[5,5,9] [5,7,3]
5	1	[5,7,3] [3,7,6]
6	1	[5,7,3] [5,5,8]
7	1	[5,5,8] [0,2,0]

This represent a single tree with this form:

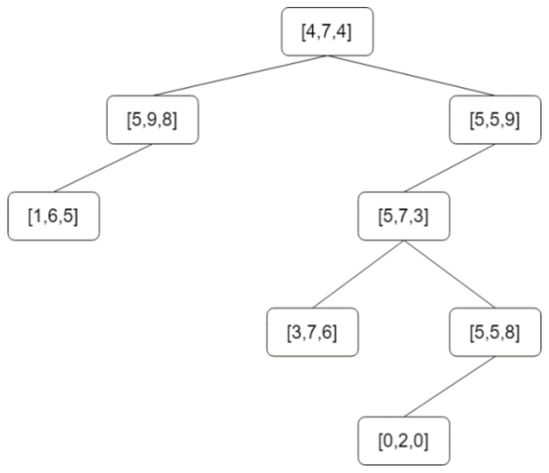

Figure 1. Graphical representation of the Tree.

- **Record_id**: is a sequential id that keeps count of how many records there are in the database.
- **Transaction_id**: keeps track of the tree structures (in the example above all six records belong to the same tree with id of 1).
- **Action**: tree structure in record format. Represent the connection between the complex objects (first object being the father and the second object being the child).

For simplicity and clarity, we make few assumptions. First, we define our complex objects as lists containing integers from 0 to 9, but the elements inside these lists could be of any type, since most data mining algorithm for finding frequent pattern will only count occurrences of these elements. Second, each node can have up to a maximum of two children, so we are dealing with binary trees (but they can also have only one child. It is also worth mentioning that the trees constructed using this method are not necessarily balanced, so one branch can be longer than the other.

3. ALGORITHM

Since we want to use this dataset to perform Frequent Pattern Mining, the complex objects cannot be randomly generated, but we need to force some sort of pattern during the creation. To achieve this, we create an arbitrary number of trees using the same sequence and then adding noise to those structures. We can then repeat the process using any other sequence. By doing this we reverse engineering the problem we want to solve: start with the frequent sequences and use those to create the dataset by adding noise to them.

The algorithm for generating the synthetic dataset is as follow:

1. Create a random sequence of this form (list containing a user specified number of lists composed by one or two integers from 0 to 9).
   [[9,2], [8], [6]]
2. Fill the previously generated sequence with noise by adding a user specified number of random integers.
   [[9,2,2], [8,2,4], [6,1,8]]
3. Use this sequence of complex objects to populate a binary tree starting with the first element as root.
4. Add noise to the tree by randomly adding new nodes either above or below.
5. Create random trees of complex objects with no predetermined sequence to make the dataset more realistic.
6. Repeat steps 2-3-4-5 using the same sequence to create different trees with that sequence.
7. Repeat all previous steps using different sequences in order to create multiple trees that follow multiple different sequences.

This result in an arbitrary number of trees constructed using an arbitrary number of different sequences (plus noise).

3.1 Example

The dataset can be generated with a single function call where it can be specified:

- The minimum number of trees to create for each sequence.
- The maximum number of trees to create for each sequence.
- The total number of sequences.
- The length of the sequences.
- Number of random objects to add to the trees (for noise).
- Number of elements inside each object (for noise).
- Number of random trees with no sequences (for noise).

The number of trees created using a sequence is a random number between the minimum and the maximum that the user specifies. This is to avoid creating the same number of trees for each sequence.

Table 1. Example of four trees created using sequence: [[9,2], [8], [6]]

Record_id	Transaction_id	Action
1	1	[8,5,4] [4,5,3]
2	1	[4,5,3] [9,2,2]
3	1	[9,2,2] [8,2,4]
4	1	[8,2,4] [6,1,8]
5	1	[9,2,2] [7,5,7]
6	1	[4,5,3] [7,5,3]
7	2	[8,5,1] [5,2,6]
8	2	[8,5,1] [7,2,9]
9	2	[7,2,9] [3,3,4]
10	2	[7,2,9] [9,2,4]
11	2	[9,2,4] [8,8,8]
12	2	[8,8,8] [6,6,3]
13	3	[4,2,8] [8,7,7]
14	3	[8,7,7] [4,1,9]
15	3	[8,7,7] [9,2,1]
16	3	[9,2,1] [2,4,7]
17	3	[9,2,1] [8,3,7]
18	3	[8,3,7] [6,1,4]
19	4	[6,6,3] [4,3,6]
20	4	[6,6,3] [9,5,5]
21	4	[9,5,5] [9,2,0]
22	4	[9,2,0] [8,1,3]
23	4	[8,1,3] [7,5,5]
24	4	[8,1,3] [6,3,0]

4. CONCLUSIONS

In this paper we have seen the growing relevancy of applications that produce records that are dependent from each other. We propose an algorithm for creating a realistic synthetic dataset of complex objects organized in binary trees, where these complex objects appear with a certain frequency, effectively simulating real-life applications that create dependencies between records. The method used to create the synthetic dataset is flexible as it allows to create longer trees, sequences of different length and complex objects with different element types inside. The incorporation of synthetic data, as well as real-world data, can help in better assessing the efficiency of data mining algorithms by creating a sort of baseline. Synthetic data can be effectively used to simulate real-world situations.