Big Data Analysis and Mining

Carson K. Leung
University of Manitoba, Canada

INTRODUCTION

Advancements in the field of information science and technology enables users to collect or generate high volumes of valuable data of different levels of veracities —such as streams of banking, financial, and shopper market basket data—at high velocities from wide varieties of data source in various real-life business, engineering, scientific applications in modern organizations and society. Embedded in these *big data* (Madden, 2012; Leung, 2015) is implicit, previously unknown, and potentially useful information and knowledge. However, these big data come with volumes beyond the ability of commonly-used software to capture, manage, and process within a tolerable elapsed time. Hence, new forms of information science and technology—such as *big data analysis and mining*—are needed to process and analyze these big data so to as enable enhanced decision making, insight, knowledge discovery, and process optimization. Over the past few years, algorithms have been proposed for various big data analysis and mining tasks, including clustering (which groups similar data together), classification (which categorizes groups of similar data), outlier detection (which identifies anomalies), and frequent pattern mining (which discovers interesting knowledge in the forms of frequently occurring sets of merchandise items or events). Most of these algorithms use the *MapReduce* model—which mines the search space with distributed or parallel computing (Shim, 2012). Among different big data analysis and mining tasks, this chapter focuses on applying the MapReduce model to big data for the discovery of frequent patterns.

BACKGROUND

Since the introduction of the research problem of *frequent pattern mining* (Agrawal, Imieliński, & Swami, 1993), numerous algorithms have been proposed (Hipp, Güntzer, & Nakhaeizadeh, 2000; Ullman, 2000; Ceglar & Roddick, 2006). Notable ones include the classical Apriori algorithm (Agrawal & Srikant, 1994) and its variants such as the Partition algorithm (Savasere, Omiecinski, & Navathe, 1995). The Apriori algorithm uses a level-wise breadth-first bottom-up approach with a candidate generate-and-test paradigm to mine frequent patterns from transactional databases of precise data. The Partition algorithm divides the databases into several partitions and applies the Apriori algorithm to each partition to obtain patterns that are locally frequent in the partition. As being locally frequent is a necessary condition for a pattern to be globally frequent, these locally frequent patterns are tested to see if they are globally frequent in the databases. To avoid the candidate generate-and-test paradigm, the tree-based FP-growth algorithm (Han, Pei, & Yin, 2000) was proposed. It uses a depth-first pattern-growth (i.e., divide-and-conquer) approach to mine frequent patterns using a tree structure that captures the contents of the databases. Specifically, the algorithm recursively extracts appropriate tree paths to form projected databases containing relevant transactions and to discover frequent patterns from these projected databases.

In various real-life business, engineering, scientific applications in modern organizations and society, the available data are not *precise* data but *uncertain* data (Tong et al., 2012; Leung,

DOI: 10.4018/978-1-5225-2255-3.ch030

Cuzzocrea, & Jiang, 2013; Leung, MacKinnon & Tanbeer, 2014; Jiang & Leung, 2015; Ahmed et al., 2016). Examples include sensor data and privacy-preserving data. Over the past few years, several algorithms have been proposed to mine and analyze these uncertain data. The tree-based UF-growth algorithm (Leung, Mateo, & Brajczuk, 2008) is an example.

With high volumes of big data, it is not unusual for users to have some phenomenon in mind. For example, a store manager is interested in some promotional items. Hence, it would be more desirable if data mining algorithms return only those patterns containing the promotional items rather than returning all frequent patterns, out of which many may be uninteresting to the store manager. It leads to *constrained mining*, in which users can express their interests by specifying constraints and the mining algorithm can reduce the computational effort by focusing on mining those patterns that are interesting to the users.

Besides the aforementioned algorithms discover frequent patterns *in serial*, there are also *parallel and distributed* frequent pattern mining algorithms (Zaki, 1999). For example, the Count Distribution algorithm (Agrawal & Shafer, 1996) is a parallelization of the Apriori algorithm. It divides transactional databases of precise data and assigns them to parallel processors. Each processor counts the frequency of patterns assigned to it and exchanges this frequency information with other processors. This counting and information exchange process is repeated for each pass/database scan.

As we are moving into the new era of big data, more efficient mining algorithms are needed because these data are wide varieties of valuable data of different veracities with volumes beyond the ability of commonly-used algorithms for mining and analyzing within a tolerable elapsed time. To handle big data, researchers proposed the use of the *MapReduce programming model*.

BIG DATA ANALYSIS AND MINING FOR FREQUENT PATTERNS

B

The MapReduce Programming Model

MapReduce (Dean & Ghemawat, 2004; Dean & Ghemawat, 2010) is a high-level programming model for processing high volumes of data. It uses parallel and distributed computing on large clusters or grids of nodes (i.e., commodity machines), which consist of a master node and multiple worker nodes. As implied by its name, MapReduce involves two key functions:

1. The "map" function, and
2. The "reduce" function.

To solve a problem using MapReduce, the master node reads and divides input big data into several partitions (sub-problems), and then assigns them to different worker nodes. Each worker node executes the *map function* on each partition (sub-problem). The map function takes a pair of ⟨key, value⟩ and returns a list of ⟨key, value⟩ pairs as an intermediate result:

- **Map:** $\langle key_1, value_1 \rangle \mapsto$ list of $\langle key_2, value_2 \rangle$,

where:

1. key_1 & key_2 are keys in the same or different domains, and
2. $value_1$ & $value_2$ are the corresponding values in some domains.

The pairs in the list of ⟨key, value⟩ pairs for this intermediate result are then shuffled and sorted. Each worker node then executes the *reduce function* on:

1. A single key from this intermediate result, and
2. The list of all values that appear with this key in the intermediate result.