Big Data Mining and Analytics

Carson Kai-Sang Leung
University of Manitoba, Canada

INTRODUCTION

Data mining and analytics aims to analyze valuable data—such as shopper market basket data—and extract implicit, previously unknown, and potentially useful information from the data. Due to advances in technology, high volumes of valuable data—such as streams of banking, financial, and marketing data—are generated in various real-life business applications in modern organizations and society. This leads us into the new era of Big Data (Madden, 2012; Mishne, Dalton, Li, Sharma, & Lin, 2013; Suchanek & Weikum, 2013). Intuitively, *Big Data* are interesting high-velocity, high-value, and/or high-variety data with volumes beyond the ability of commonly-used software to capture, manage, and process within a tolerable elapsed time. Hence, new forms of processing data are needed to enable enhanced decision making, insight, knowledge discovery, and process optimization. This drives and motivates research and practices in *business analytics and optimization*, which require techniques like *Big Data mining and analytics*, business process optimization, applied business statistics, as well as business intelligence solutions and information systems. Having developed systematic or quantitative processes to mine and analyze Big Data allows us to continuously or iteratively explore, investigate, and understand the past business performance so as to gain new insight and drive business planning. Over the past few years, several algorithms have been proposed that use the MapReduce model—which mines the search space with distributed or parallel computing—for different Big Data mining and analytics tasks (Luo, Ding, & Huang, 2012; Shi, 2012; Shim, 2012; Condie, Mineiro, Polyzotis, & Weimer, 2013; Kumar, Niu, & Ré, 2013). One such task is *frequent pattern mining*, which discovers interesting knowledge in the forms of frequently occurring sets of merchandise items or events. In this chapter, we focus mainly on *frequent pattern mining from Big Data with MapReduce*.

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. By extracting appropriate tree paths, projected databases containing relevant transactions are formed, from which frequent patterns can be discovered.

DOI: 10.4018/978-1-4666-5202-6.ch030

In many real-life applications, the available data are not *precise* data but *uncertain* data (Chen & Wang, 2011; Tong, Chen, Cheng, & Yu, 2012; Jiang & Leung, 2013; Leung, Cuzzocrea, & Jiang, 2013; Leung & Tanbeer, 2013). Examples include sensor data and privacy-preserving data. Over the past few years, several algorithms—such as the tree-based UF-growth algorithm (Leung, Mateo, & Brajczuk, 2008)—have been proposed to mine and analyze these uncertain data.

While 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 high-velocity, high-value, and/or high-variety data 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*.

MAIN FOCUS

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: "map" and "reduce". To solve a problem using MapReduce, the master node reads and divides input 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: $\langle \text{key}_1, \text{value}_1 \rangle \mapsto$ list of $\langle \text{key}_2, \text{value}_2 \rangle$,

where

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

Afterwards, pairs in the list of ⟨key, value⟩ pairs for this intermediate result are shuffled and sorted. Each worker node then executes the *reduce function* on (1) a single key from this intermediate result together with (2) the list of all values that appear with this key in the intermediate result. The reduce function "reduces"—by combining, aggregating, summarizing, filtering, and/or transforming—the list of values associated with a given key (for all *k* keys) in worker nodes and returns a list of ⟨key, value⟩ pairs, a list of values, or simply a single (aggregated or summarized) value:

Reduce: $\langle \text{key}_2$, list of $\text{value}_2 \rangle \mapsto$ list of $\langle \text{key}_3, \text{value}_3 \rangle$,

Reduce: $\langle \text{key}_2$, list of $\text{value}_2 \rangle \mapsto$ list of value_3, *or*

Reduce: $\langle \text{key}_2$, list of $\text{value}_2 \rangle \mapsto \text{value}_3$,

where

1. key_2 and key_3 are keys in some domains, and
2. value_2 and value_3 are the corresponding values in some domains.

By using the MapReduce model, users only need to focus on (and specify) the map and reduce functions, without worrying about implementation details for partitioning the input data, scheduling