MiSeRe-Hadoop: A Large-Scale Robust Sequential Classification Rules Mining Framework

Elias Egho[1], Dominique Gay[2(✉)], Romain Trinquart[1], Marc Boullé[1], Nicolas Voisine[1], and Fabrice Clérot[1]

[1] Orange Labs, 2, avenue Pierre Marzin, 22307 Lannion Cédex, France
{elias.egho,romain.trinquart,marc.boulle,
nicolas.voisine,fabrice.clerot}@orange.com
[2] Université de La Réunion, 2, rue Joseph Wetzell, 97490 Sainte Clotilde, France
dominique.gay@univ-reunion.fr

Abstract. Sequence classification has become a fundamental problem in data mining and machine learning. Feature based classification is one of the techniques that has been used widely for sequence classification. Mining sequential classification rules plays an important role in feature based classification. Despite the abundant literature in this area, mining sequential classification rules is still a challenge; few of the available methods are sufficiently scalable to handle large-scale datasets. MapReduce is an ideal framework to support distributed computing on large data sets on clusters of computers. In this paper, we propose a distributed version of *MiSeRe* algorithm on MapReduce, called *MiSeRe-Hadoop*. *MiSeRe-Hadoop* holds the same valuable properties as *MiSeRe*, i.e., it is: *(i)* robust and user parameter-free anytime algorithm and *(ii)* it employs an instance-based randomized strategy to promote diversity mining. We have applied our method on two real-world large datasets: a marketing dataset and a text dataset. Our results confirm that our method is scalable for large scale sequential data analysis.

1 Introduction

Sequential data is widely present in several domain such as biology [7,19], medical [10] and web usage logs [20]. Consequently, sequence classification [24] has become a fundamental problem in data mining and machine learning. The sequence classification task is defined as learning a sequence classifier to map a new sequence to a class label [24]. The literature in sequence classification is split into three main paradigms: (1) feature based classification; (2) distance based classification and (3) model based classification.

The feature based classification approach aims at extracting sequential classification rules of the form $\pi : s \rightarrow c_i$ where s is the body of the rule and c_i is the value of a class attribute. Then, these rules are used as an input of a classification method to build a classifier. Due to its potential of interpretability, many approaches have been developed for mining sequential classification rules, such

L. Bellatreche and S. Chakravarthy (Eds.): DaWaK 2017, LNCS 10440, pp. 105–119, 2017.
DOI: 10.1007/978-3-319-64283-3_8

a BayesFM [16], CBS [21], DeFFeD [13], SCII [27], and so on. All these methods need parameters to prune the enumeration space. Unfortunately, setting these parameters is not an easy task – each application data could require a specific setting. Egho et al. [8,9] introduce a user parameter-free approach, *MiSeRe*, for mining robust sequential classification rules. This algorithm does not require any parameter tuning and employs an instance-based randomized strategy that promotes diversity mining. *MiSeRe* works well in practice on typical datasets, but it can not provide scalability, in terms of the data size and the performance, for big data.

In modern life sciences, the sequential data can be very large; e.g., considering a document collection with millions of documents or a web site with millions of user web logs [1]. Mining massive sequential data on single computer suffers from the problems of limited memory and computing power. To solve this problem, parallel programming is an essential solution [23]. Parallel programming can be divided into two categories: shared memory system; in which processes share a single memory address space, and distributed memory system; where processes only have access to a local private memory address space [2].

A number of efficient and scalable parallel algorithms have been developed for mining sequential patterns [14]. Zaki et al. [26] present how a serial sequential approach SPADE [25] can be parallelized by using a shared memory system. Parallelizing an algorithm by using shared system architecture is easy to implement, but it does not provide enough scalability due to high synchronization among processors and memory overheads [2]. On the other hand, Guralnik et al. [12] propose a distributed memory parallel algorithm for a tree-projection based sequential pattern mining algorithm. Cong et al. [5] present Par-CSP a distributed memory parallel algorithm for mining closed sequential pattern. Par-CSP is implemented by using the Message Passing Interface (MPI) in which low level language is used for programming [11].

A recent framework for distributed memory system, Hadoop-Yahoo MapReduce, has been proposed by Google [6]. MapReduce is a scalable and fault-tolerant data processing model that allows programmers to develop efficient parallel algorithms at a higher level of abstraction [6]. Many parallel algorithms has been developed by using MapReduce framework for mining big sequential data such as: BIDE-MR [22], PLUTE [17], SPAMC [4], MG-FSM [3] and so on. However, all these algorithms need parameters to prune the enumeration space (frequency threshold, maximum length and a gap constraint) and they focus solely on mining sequential pattern.

Although a significant amount of research results have been reported on parallel implementations of sequential pattern mining, to the best of our knowledge, there is no parameter-free parallel algorithm that targets the problem of mining sequential classification rules. In this paper, we propose *MiSeRe-Hadoop*, the first scalable parameter-free algorithm for mining sequential classification rules. *MiSeRe-Hadoop* is the parallel implementation of a serial algorithm *MiSeRe* [8,9] on MapReduce framework. *MiSeRe-Hadoop* has the same features as *MiSeRe* which are: it is user parameter-free and it promotes diversity mining.

To validate our contributions, we perform an experimental evaluation on two real large datasets. The first one is marketing dataset from the French Telecom company Orange containing sequential information about the behavior of 473, 898 customers. Our second dataset includes natural language text from New York Times corpus (NYT) which consists of over 53 million sentences.

The remainder of this paper is organized as follows. Section 2 briefly reviews the preliminaries needed in our development as well as a running example. Section 3 describes the serial algorithm *MiSeRe*, while the design and implementation issues for *MiSeRe-Hadoop* are presented in Sect. 4. Section 5 presents experimental results before concluding.

2 Preliminaries

Let $\mathcal{I} = \{e_1, e_2, \cdots, e_m\}$ be a finite set of m distinct items. A **sequence** s over $\mathcal{I}$ is an ordered list $s = \langle s_1, \cdots, s_{\ell_s} \rangle$, where $s_i \in \mathcal{I}$; $(1 \leq i \leq \ell_s, \ell_s \in \mathbb{N})$. An atomic sequence is a sequence with length 1. A sequence $s' = \langle s'_1 \cdots s'_{\ell_{s'}} \rangle$ is a **subsequence** of $s = \langle s_1 \ldots s_{\ell_s} \rangle$, denoted by $s' \preceq s$, if there exist indices $1 \leq i_1 < i_2 < \cdots < i_{\ell_{s'}} \leq \ell_s$ such that $s'_z = s_{i_z}$ for all $z = 1 \ldots \ell_{s'}$ and $\ell_{s'} \leq \ell_s$. s is said to be a **supersequence** of s'. $\mathbb{T}(\mathcal{I})$ will denote the (infinite) set of all possible sequences over $\mathcal{I}$. Let $\mathcal{C} = \{c_1, \cdots, c_j\}$ be a finite set of j distinct classes. A **labeled sequential data set** $\mathcal{D}$ over $\mathcal{I}$ is a finite set of triples (sid, s, c) with sid is a sequence identifier, s is a sequence $(s \in \mathbb{T}(\mathcal{I}))$ and c is a class value $(c \in \mathcal{C})$. The set $\mathcal{D}_{c_i} \subseteq \mathcal{D}$ contains all sequences that have the same class label c_i (i.e., $\mathcal{D} = \cup_{i=1}^{j} \mathcal{D}_{c_i}$). The following notations will be used in the rest of the paper: $\boldsymbol{m}$ is the number of items in $\mathcal{I}$, $\boldsymbol{j}$ is the number of classes in $\mathcal{C}$, n is the number of triples (sid, s, c) in $\mathcal{D}$, n_c is the number of triples (sid, s, c) in $\mathcal{D}_c$, ℓ_s is the number of items in the sequence s, k_s is the number of distinct items in the sequence s, $(k_s \leq \ell_s)$ and ℓ_{max} is the number of items in the longest sequence of $\mathcal{D}$.

Table 1. $\mathcal{D}$: a tiny labeled sequential data set as an example.

sid	Sequence	Class
1	$\langle baadg \rangle$	c_1
2	$\langle agbe \rangle$	c_1
3	$\langle badgb \rangle$	c_2
4	$\langle eefgbg \rangle$	c_2

Definition 1. *(Support of a sequence) Let $\mathcal{D}$ be a* labeled sequential data set *and let s be a sequence. The* ***support*** *of s in $\mathcal{D}$, denoted $f(s)$, is defined as:*

$$f(s) = |\{(sid', s', c') \in \mathcal{D} | s \preceq s'\}|$$

The value of $n - f(s)$ can be written as $\overline{f}(s)$. The support of s in $\mathcal{D}_c$ is noted $f_c(s)$ and $\overline{f}_c(s)$ stands for $n_c - f_c(s)$.

Given a positive integer σ as a minimal support threshold and a labeled sequential data set $\mathcal{D}$, a sequence s is frequent in $\mathcal{D}$ if its support $f(s)$ in $\mathcal{D}$ exceeds the minimal support threshold σ. A frequent sequence is called a *"sequential pattern"*.

Definition 2 (Sequential classification rule). *Let $\mathcal{D}$ be a* labeled sequential data set *with j classes. A sequential classification rule π is an expression of the form:*

$$\pi : s \rightarrow f_{c_1}(s), f_{c_2}(s), \cdots, f_{c_j}(s)$$

where s is a sequence, called body of the rule, and $f_{c_i}(s)$ is the support of s in each $\mathcal{D}_{c_i}$, $i = 1 \cdots j$.

This definition of classification rule is slightly different from the usual definition where the consequent is a class value (i.e., $s \rightarrow c_i$). It refers to the notion of distribution rule [15] and allows us to access the whole frequency information within the contingency table of a rule π – which is needed for the development of our framework.

Example 1. We use the sequence database $\mathcal{D}$ in Table 1 as an example. It contains four data sequences (i.e., $n = 4$) over the set of items $\mathcal{I} = \{a, b, d, e, f, g\}$ (i.e., $m = 6$). $\mathcal{C} = \{c_1, c_2\}$ is the set with two classes (i.e., j=2). The longest sequence of $\mathcal{D}$ is $s = \langle eefgbg \rangle$ (i.e., $\ell_s = \ell_{max}$), $\ell_{max} = 6$ while $k_s = 4$. Sequence $\langle aad \rangle$ is a subsequence of $\langle baadg \rangle$. The sequence $\langle a \rangle$ is an atomic sequence. Given the sequence $s = \langle ab \rangle$, we have $f(s) = 2$, $\overline{f}(s) = 2$, $f_{c_1}(s) = 1$, $\overline{f}_{c_1}(s) = 1$, $f_{c_2}(s) = 1$ and $\overline{f}_{c_2}(s) = 1$. $\pi : \langle ab \rangle \rightarrow f_{c_1}(\langle ab \rangle) = 1, f_{c_2}(\langle ab \rangle) = 1$ is a sequential classification rule.

Given a labeled sequential data set $\mathcal{D}$ and a sequential classification rule $\pi : s \rightarrow f_{c_1}(s), f_{c_2}(s), \cdots, f_{c_j}(s)$, a Bayesian criterion *level* is defined by Egho et al. [8,9] for evaluating the interestingness of sequential classification rule. This criterion is based on the a posteriori probability of a rule given the data and does not require any wise threshold setting. The *level* criterion is defined as follows:

$$level(\pi) = 1 - \frac{cost(\pi)}{cost(\pi_\emptyset)}$$

where $cost(\pi)$ is defined as the negative logarithm of the a posteriori probability of a rule given the data:

$$cost(\pi) = -log(P(\pi \mid \mathcal{D})) \propto -log(P(\pi) \times P(D \mid \pi))$$

Considering a hierarchical prior distribution on the rule models, Egho et al. [8,9] obtained an exact analytical expression of the *cost* of a rule:

$$\begin{aligned} cost(\pi) = \log(m+1) + log(\ell_{max}+1) + \log(\frac{m^{k_s}}{k_s!}) + \log({k_s}^{\ell_s}) \\ + \log\binom{f(s)+j-1}{j-1} + \log\binom{\overline{f}(s)+j-1}{j-1} \\ + \log(f(s)!) - \sum_{i=1}^{j} \log(f_{c_i}(s)!) + \log(\overline{f}(s)!) - \sum_{i=1}^{j} \log(\overline{f}_{c_i}(s)!) \end{aligned}$$

$cost(\pi_\emptyset)$ is the cost of the default rule with empty sequence body. The cost of the default rule $\pi_\emptyset$ is formally:

$$cost(\pi_\emptyset) = log(m+1) + log(\ell_{max}+1) + log\binom{n+j-1}{j-1} + log(n!) - \sum_{i=1}^{j} log(n_{c_i}!)$$

The *level* naturally highlights the border between the interesting rules and the irrelevant ones. Indeed, rules π such that $level(\pi) \leq 0$, are less probable than the default rule $\pi_\emptyset$. Then using them to explain the data by characterizing classes of sequence objects is more costly than using $\pi_\emptyset$; such rules are considered spurious. *"rule such that* $0 < level(\pi) \leq 1$ *is an* ***interesting rule****"* (see [8,9] for more details about the interpretation of these two formulas).

3 MiSeRe Algorithm

In this section we describe MiSeRe [8,9], an algorithm for mining sequential classification rules, which forms the basis for our distributed algorithm. MiSeRe is an anytime algorithm: the more time the user grants to the task, the more it learns. MiSeRe employs an instance-based randomized strategy that promotes diversity mining. MiSeRe is based on the following two-step process:

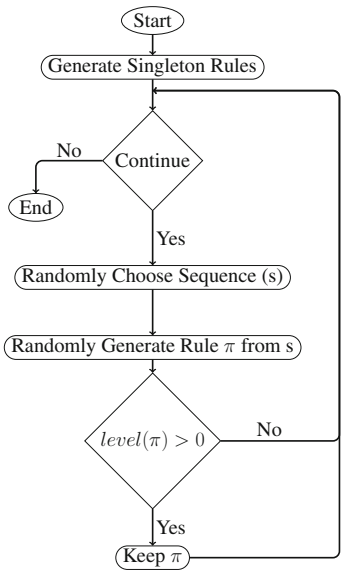


Fig. 1. Flow chart for the main procedure of the *MiSeRe* algorithm

Step 1: MiSeRe firstly performs a single data scan to gather basic statistics about the data: the number of items in $\mathcal{I}$, the number of classes in $\mathcal{C}$, the number of sequences in $\mathcal{D}$, the number of sequences in $\mathcal{D}_{c_i}$ and the length of the longest sequence of $\mathcal{D}$. In this step, MiSeRe also generates all sequential classification rules whose body is an atomic sequence, such rules with positive *level* values are chosen. These rules are selected for two reasons: first, these rules are easy to mine. Second, the short sequences are more probable a priori and preferable as the cost of the rule $c(\pi)$ is smaller for lower ℓ_s and k_s values, meeting the consensus: ***"Simpler and shorter rules are more probable and preferable"***.

Step 2: In this step, MiSeRe randomly chooses one sequence **s** from the labeled sequential database $\mathcal{D}$. Then, it randomly generates a subsequence from the chosen sequence **s**. This generation is done by randomly removing z items from **s** where z is between 1 and $\ell_s - 2$. Then, the rule π is built based on the generated subsequence s'. Finally, the rule π is added to the rule set if its level value is

positive and it is not already in $\mathcal{R}$. MiSeRe repeats the Step 2 until the algorithm is stopped by the user manually at some point in time. Figure 1 shows a flow chart (simplified for exposition, see [8,9] for full details) for the main procedure of the *MiSeRe* algorithm.

4 MiSeRe Hadoop Algorithm

In this section, we present *MiSeRe-Haddoop*, a distributed version of *MiSeRe*. *MiSeRe-Hadoop* has the same features as *MiSeRe* which are: (i) it is user parameter-free and (ii) anytime algorithm and (iii) it employs an instance-based randomized strategy. *MiSeRe-Hadoop* is divided into two steps as *MiSeRe* where each step is completely parallelized (Fig. 2).

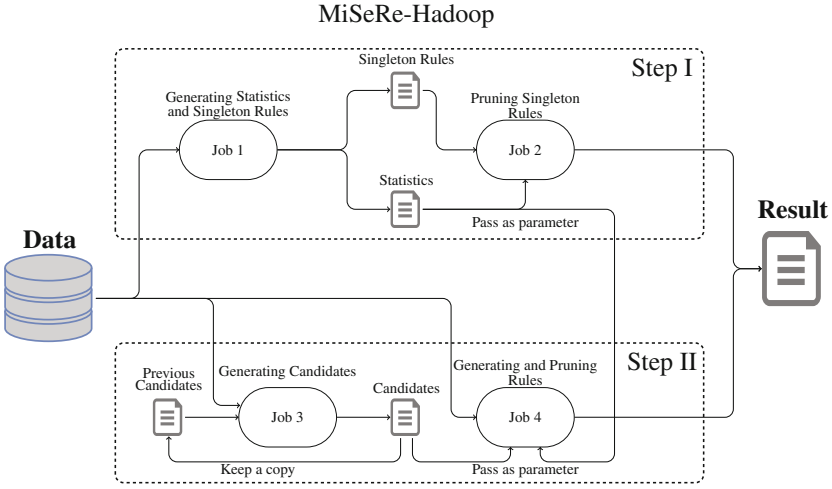


Fig. 2. MiSeRe-Hadoop

4.1 Step I:

In the first step, *MiSeRe-Hadoop* gathers basic statistics about the data and mines all sequential classification rules having only an atomic sequence in the body. This can be done efficiently in two MapReduce jobs.

The job *"**Generating Statistics and Singleton Rules**"* generates the singleton rules and the statistics about the data consisting of: the number of sequences in $\mathcal{D}_{c_i}$ (i.e., n_{c_i}), the number of sequences in $\mathcal{D}$ (i.e., n) and the length of the longest sequence of $\mathcal{D}$ (i.e., ℓ_{max}). During this job, the data is distributed to available mappers. Each mapper takes a pair $(s, c) \in \mathcal{D}$ as input and tokenizes it into distinct items. Then, the mapper emits the class label for each item. In order to compute the statistic values about data n_{c_i}, n and ℓ_{max}, the mappers output the pair (key,value) as follow: the mapper emits also 1 for the class label c (i.e., $(class.c, 1)$), 1 for the term n (i.e., $(notation.n, 1)$) and the number of items in the sequence for the term ℓ_{max} (i.e., $(notation.\ell_{max}, \ell_s)$).

Example 2. Given the pair $(\langle baadg\rangle, c_1)$ from our toy data set (Table 1). The mapper splits the sequence $\langle baadg\rangle$ into distinct items b, a, d and g. Then, the mapper forms the key value pairs: (b, c_1), (a, c_1), (d, c_1), (g, c_1), $(class.c_1, 1)$, $(notation.n, 1)$ and $(notation.\ell_{max}, 5)$.

The reducer forms an aggregation function and outputs its results into two files: *Singleton Rules* and *Statistics* depending on the key as follows:

- In case the key is a class label $class.c_i$ or the term $notation.n$, the reducer sums up the values associated with this key. Then, it outputs them to the *Statistics* file.
- In case the key is the term $notation.\ell_{max}$, the reducer selects the maximum values associated with the key $notation.\ell_{max}$. Then, it outputs them to the *Statistics* file.
- In case the key is an item e; $e \in \mathcal{I}$, the reducer computes the frequency of each class label appearing in its value. Then, it outputs the sequence and its support in each $\mathcal{D}_{c_i}$ to the *Singleton Rules* file.

Example 3. If the reducer receives the following (key,value) pairs from the mapper: (f, c_2), (g, c_1), (g, c_2), (g, c_2), $(notation.\ell_{max}, 5)$, $(notation.\ell_{max}, 4)$, $(notation.\ell_{max}, 5)$, $(notation.\ell_{max}, 6)$, $(notation.n, 1)$, $(notation.n, 1)$, $(notation.n, 1)$, $(notation.n, 1)$, $(notation.c_2, 1)$ and $(notation.c_2, 1)$. Then, it results $(\langle f\rangle,\ f_{c_2}(\langle f\rangle) = 1)$, $(\langle g\rangle,\ f_{c_1}(\langle g\rangle) = 1\ f_{c_2}(\langle g\rangle) = 2)$, $(class.c_2, 2)$, $(notation.n, 4)$ and $(notation.\ell_{max}, 6)$.

Given the output of the job ***"Generating Statistics and Singleton Rules"***, we compute the rest of the statistics about the data consisting of: the number of classes in $\mathcal{C}$ (i.e., j) and the number of items in $\mathcal{I}$ (i.e., m).

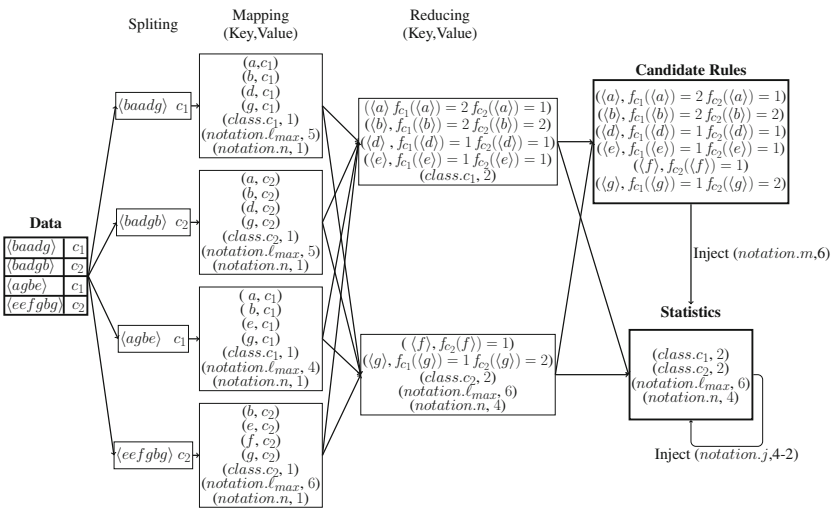


Fig. 3. Generating statistics and singleton rules job

The term j is computed by counting the number of lines in *Statistics* file then subtracting 2 from the result which represents the two lines of the terms $notation.\ell_{max}$ and $notation.n$. Then, we inject the value of j into the *Statistics* file. The term m is computed by counting the number of lines in *Singleton Rules* file and we inject it into the *Statistics* file. Figure 3 details the job *"**Generating Statistics and Singleton Rules**"*.

The job "**Pruning Singleton Rules**" prunes the candidate rules which are generated after applying the job *"**Generating Statistics and Singleton Rules**"*. This job consists only of mappers. The rules are distributed to available mappers. The basic statistics of our data (i.e., *Statistics* file) are passed as a parameter to each mapper by using the distributed cache. Then, the mapper computes the level of each rule and emits the rules having a positive level, i.e., *"interesting singleton rules"*, as a key and its level as a value.

4.2 Step II:

In the second step, *MiSeRe-Hadoop* mines all sequential classification rules having a sequence with more than one item in the body. *MiSeRe-Hadoop* iteratively repeats this step until the algorithm is stopped by the user manually at some point in time. The *Step II* can be efficiently achieved in two MapReduce jobs.

The job *"**Generating Candidates**"* employs an instance-based randomized strategy to generate candidate sequences from the data. This strategy can generate exactly the same candidate sequence several times. To avoid this redundancy problem, we define two kinds of mappers as follows:

- The set of first mappers take data as an input and generates new candidate sequences as an output. When the data is distributed to these mappers, each mapper randomly chooses one sequence **s** from each subset of data. From this chosen sequence **s**, the mapper randomly removes some items to generate a new subsequence. Finally, the mapper outputs the generated subsequence as a key and the term *"New"* as a value. This task is repeated by each mapper until a fixed number of candidate sequences is generated from each subset.
- The second set of mappers take the sequences from *Previous Candidates*[1] file which is generated from this step iteratively. The sequences in *Previous Candidates* file are distributed to these mappers. Then, the mapper outputs the sequence as a key and the term *"Old"* as a value.

The reducer filters the sequences based on its value i.e., it only outputs those sequences which do not include the term *"Old"* in their value. Figure 4 details the job *"**Generating Candidates**"*. The candidate sequences generated from this job are then copied to the *Previous Candidates* file to avoid generating the same candidate sequences from the data in the next iteration.

The job *"**Generating and Pruning Rules**"* generates the sequential rules from the data based on the candidate sequences generated from the job

[1] This file keeps a copy of all the candidate sequences generated from the job *"**Generating Candidates**"* in each iteration.

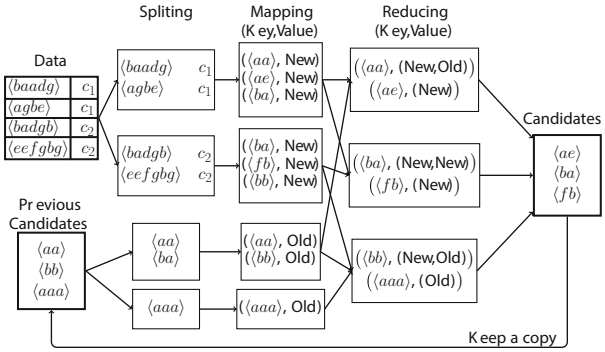


Fig. 4. Generating candidates job

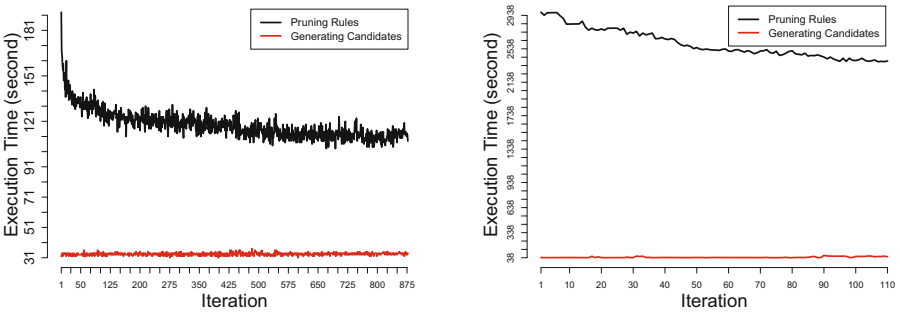
*"**Generating Candidates**"*. The data is distributed to available mappers while the candidates are passed as a parameter to each mapper by using the distributed cache. Each mapper takes a pair $(s, c) \in \mathcal{D}$ as input. Then, each candidate sequence is checked against the input sequence, if it is a sub-sequence of the input sequence, a key value pair is emitted where the key is the candidate sequence and the value is the class label. The reducer then generates the sequential classification rules by aggregating the frequency of the sequence in each class. Finally, the reducer computes the level of the rule and emits only the rules having a positive level, i.e., *"interesting rules"*, as a key and its level as a value.

5 Experiments

In this section, we empirically evaluate our approach on two real big datasets. *MiSeRe-Hadoop* is implemented in JAVA. The experiments are performed on a cluster of 6 computers, each with 64GB memory. One machine actes as the Hadoop master node, while the other five machines acte as worker nodes. The experiments are designed to discuss the following two points: the scalability of *MiSeRe-Hadoop* and the diversity of the mined rules with *MiSeRe-Hadoop*.

Datasets. We use two real-world datasets for our experiments. The first dataset is a large marketing database from the French Telecom company Orange containing sequential information about the behavior of **473 898 customers**. We use this dataset for predicting their propensity to churn. Each sequence represents a time-ordered set of actions (or events) categorized into two parts: (1) history of interaction between the customer and the LiveBox[2], e.g., changing the channel, rebooting the router, etc., (2) state of the box such as sending the temperature of box etc. These customers are classified into **2** classes. The first class includes **159 229** customers who terminated their contract with the company.

[2] Orange Livebox is an ADSL wireless router available to customers of Orange's Broadband services in several countries.



(a) Orange (b) New York Times

Fig. 5. The scalability results for each iteration in Step 2.

The second class consists of **314 669** customers who still have the contract. This data set contains **433** distinct actions, the longest sequence is a customer having **23 759** actions while the median length of sequences is **192** actions. Our second dataset is the New York Times corpus (NYT) [18] which consists of **53 267 584** sentences from **1.8 million** articles published between 1987 and 2007. We treat each sentence as an input sequence with each word (token) as an item. All the sentences were stemmed and lemmatized using WordNet lemmatizer. These sentences are classified into **34** classes such as News, Sports, Health, etc. This data set contains **965 782** distinct words, the longest sequence is a sentence having **2 381** words while the median length of sequences is **10** words. The difference between Orange and NYT dataset is that the former does not contain a large number of sequences however each sequence contains too many number of actions creating longer sequences. While the latter consists of huge number of sequences with lesser number of words in the sequences.

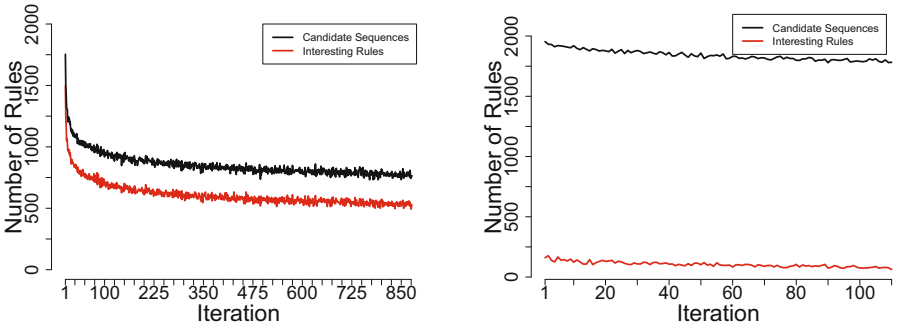
Scalability. In this experiment, we explore the scalability of each job in *MiSeRe-Hadoop*. We run *MiSeRe-Hadoop* for **48 h** over *Orange* dataset. *MiSeRe-Hadoop* firstly runs the job *"generating statistics and singleton rules"* which takes **43 s** and generates **433 candidate singleton rules**. Then, *MiSeRe-Hadoop* filters these rules over the second job *"pruning singleton rules"* which takes **40 s** and returns **268 singleton interesting rules**. Then, *MiSeRe-Hadoop* passes to the second step where it iteratively repeats the two jobs *"generating candidates"* and *"generating and pruning rules"* until it is stopped after **48 h**. In each iteration, *MiSeRe-Hadoop* generates at most **2000** distinct candidate sequences then prunes them. *MiSeRe-Hadoop* repeats this step **877 times** during **48 h**. Figure 5 (a) shows the execution times of the two jobs *"generating candidates"* and *"generating and pruning rules"* over *Orange* dataset. The average execution time of the job *"generating candidates"* is **33.4 s** while for the job *"generating and pruning rules"* it is **118.51 s**.

We also run *MiSeRe-Hadoop* for **192 h** over *NYT* dataset. The job *"generating statistics and singleton rules"* takes **183 s** and generates **965 782 candidate**

singleton rules. Then, the job *"pruning singleton rules"* filters these rules which takes **47 s** and returns **7 092 singleton interesting rules**. Then, *MiSeRe-Hadoop* iteratively repeats **110 times** the two jobs *"generating candidates"* and *"generating and pruning rules"* during **192 h**. In each iteration, *MiSeRe-Hadoop* generates at most **2000** distinct candidate sequences then prunes them. Figure 5(b) shows the execution times of the two jobs *"generating candidates"* and *"generating and pruning rules"* over *NYT* dataset. The average execution time of the job *"generating candidates"* is **43.67 s** while for the job *"generating and pruning rules"* it is **2 528.4 s**.

From Fig. 5, it can be noticed that the job *"generating and pruning rules"* is the only job which takes the most time in the pipeline of *MiSeRe-Hadoop*. The performance of this job is based on two criteria. The first one is the number of candidate sequences which are generated form the job *"generating candidates"*. For this reason, the execution time of the job *"generating and pruning rules"* for NYT data set is more stable and fixed around **2 528.4 s** as the number of candidates are more stable and fixed around 2000 new candidates sequences (see Fig. 6). On the other hand, for *Orange* dataset, the number of candidate sequences is lesser at each iteration, for this reason the execution time of the job *"generating and pruning rules"* is decreasing at each iteration. The second criteria which effects on the performance of the job *"generating and pruning rules"* is the size of the dataset. For this reason, the execution time of this job over *NYT* dataset takes 20 times more than the execution time of the same job over *Orange* dataset because the size of *NYT* data set is larger than the *Orange* data set.

Diversity. In this section, we study the diversity of the mined rules by *MiSeRe-Hadoop*. The maximum number of candidate sequences to be generated by the job *"generating candidates"* at each iteration over the two datasets was set to 2000. The main goal of these experiments are as follows: (1) how many candidate sequences and interesting rules are generated by *MiSeRe-Hadoop* at each itera-



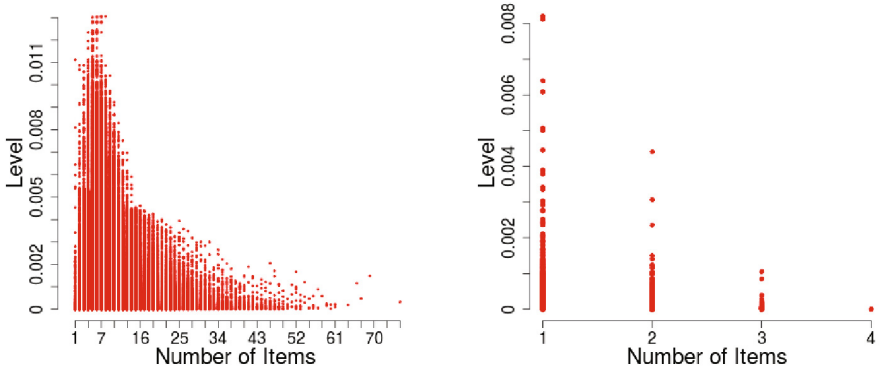
(a) Orange (b) New York Times

Fig. 6. Number of candidate sequences and rules generated over each iteration in Step 2.

tion, (2) the number of candidate sequences and interesting rules generated by *MiSeRe-Hadoop* having n-items and (3) what is the relation between the number of items in the body of the rules and its value for the level criterion.

Figure 6 shows the number of candidate sequences generated and the interesting rules found over each iteration. From Fig. 6(a), it can be observed that for *Orange* dataset, at each iteration around **70%** of the candidate sequences are interesting rules which implies that these rules were easier to generate. On the contrary, for *NYT* dataset, we have **text data** where *MiSeRe-Hadoop* has difficulty in finding interesting rules at each iteration as the percentage of finding rules from candidate sequence is very low, which is around **5%** (see Fig. 6(b)). The total number of interesting rules generated from *Orange* dataset during **877 iterations** is **534 460** rules, while the total number of rules generated from *NYT* dataset during **110 iterations** is **11 473** rules.

In Fig. 7, we study the relation between the length and the value of level of interesting rules. We plot the length of the interesting rule against its level. For *Orange* dataset, we can extract interesting rules with up to **72 items** while for *NYT*, *MiSeRe-Hadoop* extracts interesting rules with at most **4 items (words)**. From Fig. 7, it can be concluded that the level value of the shorter rules is larger than the longer ones, meeting the consensus: ***"Simpler and shorter rules are more probable and preferable"***.



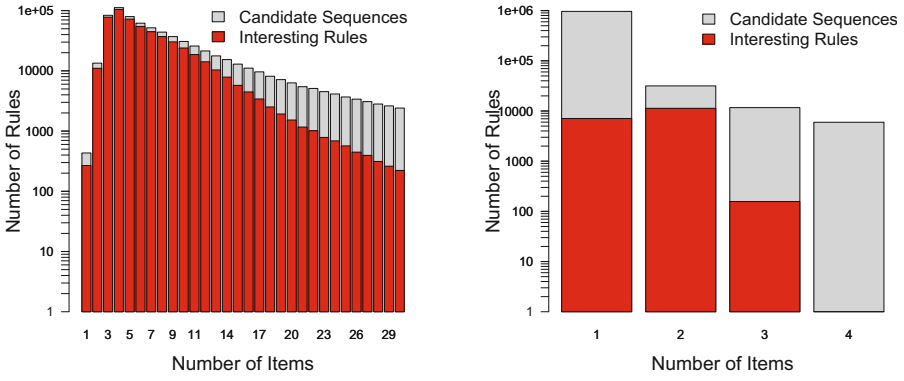
(a) Orange (b) New York Times

Fig. 7. Length of rules against its value of level criterion.

We also study the relation between the length of candidate sequences and mined rules. Figure 8(a) shows the number in logscale of candidate sequences and rules mined over the step 2 of *MiSeRe-Hadoop* from *Orange* dataset. For this plot, we limit the visualization up to **30 items** as **99.75%** of mined rules have a body with maximum **30 items**. In *MiSeRe-Hadoop*, the job *"generating candidates"*

generates more candidate sequences with lesser number of items as compared to the longer ones because short rules are more probable and preferable. For this reason, the candidate sequences having items less than 8 represent 85% of all the generated candidate sequences. For *NYT* data set, *MiSeRe-Hadoop* generates most of the candidate sequences with 2-items because it is a text data set and has 965 782 distinct words. Thus, it can generate upto $(965782)^2$ distinct candidate sequences with 2-items. From Fig. 8(b), It can be observed that generating sequential classification rules form text data is not easy task. For example, *MiSeRe-Hadoop* generates **11 673 candidate sequences** with three words and finally finds just **157** interesting rules over these candidates.

Figures 6, 7 and 8 highlight that the randomized strategy employed in *MiSeRe-Hadoop* allows us to mine interesting rules with diversity which is highly dependent on the data.



(a) Orange (b) New York Times

Fig. 8. Length of candidate sequences and rules against the number of rules.

6 Conclusion and Future Work

This paper focuses on the important problem of mining sequential rule patterns from large scale sequential data. We propose *MiSeRe-Hadoop*, a scalable algorithm for mining sequential rules in MapReduce. *MiSeRe-Hadoop* is a parameter-free algorithm that efficiently mines interesting rules. The empirical experiments show that: (1) our method is scalable for large scale sequential data analysis and (2) the randomized strategy employed in *MiSeRe-Hadoop* allows us to mine interesting rules with diversity. As future work, we plan to extend our approach for multi label sequential data set. On the other hand, we are also planning on using the mined sequential classification rules by *MiSeRe-Hadoop* as an input of the classification method to build a classifier.

References

1. Anastasiu, D.C., Iverson, J., Smith, S., Karypis, G.: Big data frequent pattern mining. In: Aggarwal, C.C., Han, J. (eds.) Frequent Pattern Mining, pp. 225–259. Springer, Cham (2014). doi:10.1007/978-3-319-07821-2_10
2. Andrews, G.R.: Foundations of Multithreaded, Parallel, and Distributed Programming. University of Arizona, Wesley (2000)
3. Beedkar, K., Berberich, K., Gemulla, R., Miliaraki, I.: Closing the gap: sequence mining at scale. ACM Trans. Database Syst. **40**(2), 8:1–8:44 (2015)
4. Chen, C.C., Tseng, C.Y., Chen, M.S.: Highly scalable sequential pattern mining based on mapreduce model on the cloud. In: 2013 IEEE International Congress on Big Data, pp. 310–317 (2013)
5. Cong, S., Han, J., Padua, D.: Parallel mining of closed sequential patterns. In: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, pp. 562–567. ACM (2005)
6. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
7. Deshpande, M., Karypis, G.: Evaluation of techniques for classifying biological sequences. In: Chen, M.-S., Yu, P.S., Liu, B. (eds.) PAKDD 2002. LNCS, vol. 2336, pp. 417–431. Springer, Heidelberg (2002). doi:10.1007/3-540-47887-6_41
8. Egho, E., Gay, D., Boullé, M., Voisine, N., Clérot, F.: A parameter-free approach for mining robust sequential classification rules. In: 2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14–17, 2015, pp. 745–750 (2015)
9. Egho, E., Gay, D., Boullé, M., Voisine, N., Clérot, F.: A user parameter-free approach for mining robust sequential classification rules. Knowl. Inform. Syst. **52**, 1–29 (2016)
10. Egho, E., Jay, N., Raïssi, C., Nuemi, G., Quantin, C., Napoli, A.: An approach for mining care trajectories for chronic diseases. In: Peek, N., Marín Morales, R., Peleg, M. (eds.) AIME 2013. LNCS, vol. 7885, pp. 258–267. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38326-7_37
11. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard. Parallel Comput. **22**(6), 789–828 (1996)
12. Guralnik, V., Karypis, G.: Parallel tree-projection-based sequence mining algorithms. Parallel Comput. **30**(4), 443–472 (2004)
13. Holat, P., Plantevit, M., Raïssi, C., Tomeh, N., Charnois, T., Crémilleux, B.: Sequence classification based on delta-free sequential patterns. In: ICDM 2014, pp. 170–179 (2014)
14. Itkar, S., Kulkarni, U.: Distributed sequential pattern mining: a survey and future scope. Int. J. Comput. Appl. **94**(18), 28–35 (2014)
15. Jorge, A.M., Azevedo, P.J., Pereira, F.: Distribution rules with numeric attributes of interest. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) PKDD 2006. LNCS, vol. 4213, pp. 247–258. Springer, Heidelberg (2006). doi:10.1007/11871637_26
16. Lesh, N., Zaki, M.J., Ogihara, M.: Mining features for sequence classification. In: ACM SIGKDD 1999, pp. 342–346 (1999)
17. Qiao, S., Li, T., Peng, J., Qiu, J.: Parallel sequential pattern mining of massive trajectory data. Int. J. Comput. Intell. Syst. **3**(3), 343–356 (2010)

18. Sandhaus, E.: The New York Times Annotated Corpus. Linguistic Data Consortium, Philadelphia (2008)
19. She, R., Chen, F., Wang, K., Ester, M., Gardy, J.L., Brinkman, F.S.L.: Frequent-subsequence-based prediction of outer membrane proteins. In: ACM SIGKDD 2003, pp. 436–445 (2003)
20. Tan, P., Kumar, V.: Discovery of web robot sessions based on their navigational patterns. Data Min. Knowl. Discov. **6**(1), 9–35 (2002)
21. Tseng, V.S., Lee, C.: CBS: a new classification method by using sequential patterns. In: SDM 2005, pp. 596–600 (2005)
22. Wang, J., Han, J.: BIDE: efficient mining of frequent closed sequences. In: ICDE 2004, pp. 79–90 (2004)
23. Wu, X., Zhu, X., Wu, G.Q., Ding, W.: Data mining with big data. IEEE Trans. Knowl. Data Eng. **26**(1), 97–107 (2014)
24. Xing, Z., Pei, J., Keogh, E.J.: A brief survey on sequence classification. SIGKDD Explor. **12**(1), 40–48 (2010)
25. Zaki, M.: Sequence mining in categorical domains: incorporating constraints, pp. 422–429 (2000)
26. Zaki, M.J.: Parallel sequence mining on shared-memory machines. J. Parallel Distrib. Comput. **61**(3), 401–426 (2001)
27. Zhou, C., Cule, B., Goethals, B.: Itemset based sequence classification. In: ECML/PKDD 2013, pp. 353–368 (2013)