Parallel ADR Detection Based on Spark and BCPNN

Li Sun, Shan Sun, Tianlei Wang, Jiyun Li*, and Jingsheng Lin

Abstract: Adverse Drug Reaction (ADR) is one of the major challenges to the evaluation of drug safety in the medical field. The Bayesian Confidence Propagation Neural Network (BCPNN) algorithm is the main algorithm used by the World Health Organization to monitor ADRs. Currently, ADR reports are collected through the spontaneous reporting system. However, with the continuous increase in ADR reports and possible use scenarios, the efficiency of the stand-alone ADR detection algorithm will encounter considerable challenges. Meanwhile, the BCPNN algorithm requires a certain number of disk I/O, which leads to considerable time consumption. In this study, we propose a Spark-based parallel BCPNN algorithm, which speeds up data processing and reduces the number of disk I/O in BCPNN, and two optimization strategies. Then, the ADR data collected from the FDA Adverse Event Reporting System are used to verify the performance of the proposed algorithm and its optimization strategies. Experiments show that the parallel BCPNN can significantly accelerate data processing and the optimized algorithm has a high acceleration rate and can effectively prevent memory overflow. Finally, we apply the proposed algorithm to a dataset provided by a real medical consortium. Experiments further prove the performance and practical value of the proposed algorithm.

Key words: Adverse Drug Reaction (ADR); Bayesian Confidence Propagation Neural Network (BCPNN); parallel; Spark

1 Introduction

Adverse Drug Reaction (ADR) is one of the major challenges to the evaluation of drug safety in the medical field[1]. ADRs have considerably increased the economic burden and have become a threat to human life and health, even leading to serious public health events and mortality. Therefore, how to effectively identify ADRs has become the problem that both the academic and health management departments need to pay close attention to.

At present, ADR data are mainly collected through the spontaneous reporting system around the world. The frequency and Bayesian methods are commonly used to detect ADR signals[2]. The frequency method is a classic signal detection method, which is based on the two-by-two contingency table and applies the method of ratio imbalance.

A disadvantage of the frequency method is that the four factors in the two-by-two contingency table cannot be zero, which means that each factor in the table must have corresponding reports. If the amount of data is small, then it is prone to false positive signals.

The most commonly used Bayesian method is the Bayesian Confidence Propagation Neural Network (BCPNN) algorithm[3]. The BCPNN algorithm applies
both the principle of ratio imbalance and the Bayesian approach, which makes the model a feedforward neural network. With the increase and update of database information, the model can execute regular autonomous learning and deduction and reevaluate the cumulative ADR report. This feature makes the model capable of detecting ADR signals in the early stage\(^4\). Moreover, the model can calculate relevant results despite missing data. Thus, the BCPNN algorithm has become one of the main algorithms used by the World Health Organization (WHO) to monitor daily ADRs. The traditional BCPNN algorithm exhibits a good performance in processing a certain amount of data. However, with the increase in spontaneous reporting data, the efficiency of data processing using the stand-alone algorithm will encounter considerable challenges. The BCPNN algorithm requires a certain number of disk I/O in the process, which will consume a considerable amount of computation time.

Spark is an open-source in-memory analytics cluster computing engine that supports operations in parallel steps\(^5\). Spark achieves distributed computation based on the idea of MapReduce and is a memory-based calculation method. Thus, Spark can save a considerable number of disk I/O and amount of storage time in the process of executing the program.

Thus, in this study, a parallel BCPNN algorithm based on Spark is proposed to speed up data processing, reduce the number of disk I/O, and maintain a good stability with the increase in the dataset. Then, an optimized parallel algorithm is proposed to further speed up data processing and prevent memory overflow because of the data tilt caused by the uneven data. Then, experiments are conducted using the ADR data provided by the Food and Drug Administration (FDA) Adverse Event Reporting System (FAERS). Results show that the parallel BCPNN algorithm can efficiently reduce the running time. Compared with the parallel BCPNN algorithm, the optimized parallel method has a better acceleration ratio and can more effectively reduce the risk of memory overflow. Thus, the optimized parallel method can better process massive data in ADR signal detection than the parallel BCPNN algorithm. Finally, we apply the algorithms to a real project to further prove the performance of the method proposed in this study.

This paper is organized as follows: Section 2 briefly introduces the related concepts, including the BCPNN algorithm and Spark distributed computing framework. Section 3 introduces the Spark-based parallel BCPNN algorithm and two optimization strategies in detail. In Section 4, experiments are conducted to verify the performance of the method proposed in Section 3. In Section 5, the algorithm and its optimization strategies are applied to a real project. The results further prove the performance and practical value of the theory proposed in this study.

2 Related Work

2.1 BCPNN algorithm

The BCPNN algorithm was proposed in 1998 by Bate et al.\(^6\) working on the Uppsala International Monitoring Center, a unit of the WHO. The BCPNN is a feedforward neural network that uses the principles of Bayes’ law to learn knowledge and evaluate an action or event.

For the work presented in this study, we use the BCPNN as a one-layer feedforward neural network. Drugs and ADR events are used as the input and output of the neural network, respectively. In our daily applications, drugs and ADR events are known; thus, we can construct the weights (generally called the Information Component, IC) between the input and output nodes\(^7\). These weights provide a measure of dependency between specific input and output nodes. The IC, which is used as the weight of the network, is derived using a calculation based on information theory\(^8\). The IC is defined, as follows:

\[
IC = \log_2 \frac{p(x,y)}{p(x)p(y)} \tag{1}
\]

where \(p(x)\) represents the probability of the occurrence of a particular drug \(x\) in an event report, \(p(y)\) represents the probability of the occurrence of a particular adverse event \(y\) in an event report, and \(p(x,y)\) represents the probability of the occurrence of the drug–ADR combination in an event report.

Then, the IC expectations can be expressed in Formula (2) as follows:

\[
E(IC) = \log_2 \frac{E(p(x,y))}{E(p(x))E(p(y))} \tag{2}
\]

Beta distribution is selected for the probability of the individual variables, and Dirichlet distribution is selected for the joint probability because they are conjugate priors. The formula for \(E(IC)\) can be expressed as follows:

\[
E(IC) = \log_2 \frac{(a + y_1)(N + a)(N + \beta)}{(N + y)(a + b + \alpha)(a + c + \beta)} \tag{3}
\]

Moreover, the variance of IC is defined as follows:
where $\alpha, \alpha_1, \beta, \beta_1$ are the parameters of the beta distribution, $\gamma$ and $\gamma_{11}$ are parameters of the Dirichlet distribution, and $N$ represents the number of all of the reports. The meaning of $a, b, c, d$ and $N$ are shown in Table 1.

For the parameters $\alpha, \alpha_1, \beta, \beta_1, \gamma,$ and $\gamma_{11}$, when no association exists between the drug and the ADR event, the default parameters can be defined as follows: $\alpha_1 = 1, \alpha = 2, \beta_1 = 1, \beta = 2,$ and $\gamma_{11} = 1$.

Notably, if we assume that no association exists between the drug and the ADR event, then the parameter can be defined as presented previously. However, if we conduct signal detection on a certain amount of ADR data and obtain specific statistical results, then, when a new batch of data is obtained, the parameter should be updated using the prior probabilities derived from the previous results, which reflect the principle of Bayesian statistics in the BCPNN algorithm.

The parameter $\gamma$ is defined as follows:

$$\gamma = \gamma_{11} \frac{((N + \alpha)(N + \beta))}{((b + \alpha_1)(c + \beta_1))}$$

Then, we can determine the weight of the neural network information using Formulas (3) and (4).

In general, the standard BCPNN is used to detect network information using Formulas (3) and (4).

$$V(\text{IC}) = \left(\frac{1}{\ln 2}\right)^2 \frac{(N - a + \gamma - \gamma_{11})}{(a + \gamma_{11})(1 + N + \gamma)} + \frac{(N - a - b + \alpha - \alpha_1)}{(a + b + \alpha_1)(1 + N + \alpha)} + \frac{(N - a - c + \beta - \beta_1)}{(a + c + \beta_1)(1 + N + \beta)}$$

(4)

Thus, with the increase in the data, the accuracy of the results will be improved, making it suitable for ADR detection in big data environments.

### 2.2 Spark introduction

Spark is a memory-based computing framework, which has better computing and fault tolerance capabilities and supports batch, iterative, and flow calculations. Spark is a general distributed computing framework, which is based on the Hadoop MapReduce algorithm. The core idea of the Hadoop MapReduce algorithm is to split a large task into small ones and execute them in parallel. The method of splitting a large task into small ones can efficiently avoid the problem that large data cannot be loaded into memory. However, intermediate results need to be stored in the Hadoop Distributed File System (HDFS) on the Hadoop platform, which will consume a considerable amount of time because of the large number of disk I/O. Thus, the Hadoop platform is unsuitable for iterative computation and is usually only suitable for one iteration or large data query. Hence, with the demand to solve the problem of data mining algorithm efficiency, Spark came into being.

Spark also has the advantages of Hadoop MapReduce. However, in contrast to that of MapReduce, the intermediate and output results of Spark jobs can be stored in memory. When the intermediate result needs to be used again, it can be immediately taken from memory, which reduces the time consumption of disk I/O and improves the efficiency of data computing. Spark provides a new method of data computing with fault tolerance, which effectively reduces the disk and network I/O overhead.

Spark provides a memory-based abstract object Resilient Distributed Dataset (RDD). RDD is the central module and class in the Spark platform, represents the abstraction of the dataset, and is a fault-tolerant collection of elements that can be operated in parallel. Spark provides and uses a number of RDD operators, such as map, reduce, union, filter, reduceByKey, partitionBy, join, and count, to complete the RDD conversion. RDD can be cached in the RAM, which
ensures rapid data access and reduces the number of disk operations.

Another important concept in Spark is the Directed Acyclic Graph (DAG). In Spark, the calculations occur when the action operation of RDD is triggered. For all of the transformation operations before the action operation, Spark only records the trajectory of the transformation operation of RDD without triggering a real calculation. The Spark kernel draws a DAG of the calculated path. When the action is triggered, the calculation starts from the starting point of the DAG[15, 16].

The Spark calculation model is shown in Fig. 1.

In Fig. 1, each of the large solid-line rectangles represents an RDD, and the small rectangles inside them are the data partitions in the RDDs.

3 Our Approach

As discussed in Section 1, with the increase and update of the ADR reports, the traditional BCPNN algorithm can execute regular autonomous learning and deduction and reevaluate the cumulative ADR report[17]. Thus, with the increase in the data, the accuracy of the results will be improved. The traditional BCPNN algorithm is suitable for ADR detection in big data environments.

However, with the increase in the reports collected through the spontaneous reporting system, the use of the BCPNN method alone will considerably limit the data processing speed. Moreover, the traditional BCPNN algorithm requires a certain number of disk I/O, which not only consumes time but also takes up the system CPU. Thus, a method that can accelerate data processing, ensure the stability of the system, and reduce the number of disk I/O needs to be proposed.

In this study, a Spark-based parallel BCPNN algorithm is proposed to solve the problems that ADR signal detection encounters in big data environments. Two optimization strategies are also proposed.

In this section, we first describe the algorithm flow and the implementation method of the Spark-based parallel BCPNN algorithm. Then, aiming at the problems that the parallel BCPNN algorithm encounters when applied to ADR detection, we propose two optimization strategies. The principles and implementation algorithms of these two optimization strategies are described in detail.

3.1 Parallel BCPNN algorithm based on Spark

We can divide the traditional stand-alone BCPNN algorithm into four stages. In the first stage, when new data are inputted into the database, the parameters $\alpha, \alpha_1, \beta, \beta_1$, and $\gamma_{11}$ are set as default values if it is the first time to detect the ADR signal on the dataset or are updated according to the prior probability if ADR signal detection is conducted on a certain amount of ADR data and statistical results are obtained. In the second stage, the values of $a, b, c, \text{ and } N$ in the two-by-two contingency table are updated and the value of the last parameter $\gamma$ is calculated using Formula (5) and the parameters $\alpha, \alpha_1, \beta, \beta_1$, and $\gamma_{11}$ and $a, b, c, \text{ and } N$ are updated using the previously presented steps. In the third stage, IC (in this study, IC refers to $E(\text{IC})$) and SD are calculated using Formulas (2) and (3) and the formula $SD = \sqrt{\text{VAR}(\text{IC})}$. In the fourth stage, the suspected drug–ADR combination is detected according to the determination criteria.

The BCPNN algorithm based on Spark can also be calculated according to the four stages of the traditional BCPNN algorithm. The concrete implementation flowchart of the Spark-based parallel BCPNN algorithm is shown in Fig. 2, and the four stages of the Spark-based parallel algorithm will be described in detail.

In the first stage of the Spark-based parallel algorithm, when new data are inputted into the dataset, all of the data in the dataset will be stored in the HDFS, an RDD will be created for the dataset, and the RDD will be cached in the memory[18]. If it is the first time to detect the ADR signal on the dataset, then the parameters $\alpha, \alpha_1, \beta, \beta_1$, and $\gamma_{11}$ need not be calculated. Thus, only the default values, that is, $\alpha_1 = 1, \alpha = 2, \beta_1 = 1, \beta = 2, \text{ and } \gamma_{11} = 1$ are considered. However, if specific statistical results are obtained from a certain amount of ADR data, then
the parameters $\alpha, \alpha_1, \beta, \beta_1,$ and $\gamma_1$ should be updated according to the prior probability and the new data using the map operation.

In the second stage, drug and ADR data in the RDD should be recounted to determine the values of $a, b,$ and $c$ in the two-by-two contingency table. The method used to compute $a, b,$ and $c$ adopts the idea of MapReduce.

We use the map operation to set key-value pairs, where the field that needs to obtain the statistics is set as the key and the value is set as 1. Then, we count the key using the reduceByKey operation. The value of $N$ is obtained using the function count(). In order to facilitate the computation in the third stage, we use the join operation to convert the data of drug, ADR, $a$, intAB, and intAC into a vector and the map operation to convert the value of $N$ into a vector. The transformation operations of RDD in the second stage are shown in Fig. 3.

Then, we convert the results of the first and second stages into a vector string (Drug, ADR, $a$, intAB, intAC, $N$, $\alpha, \alpha_1, \beta, \beta_1, \gamma_1$) using the join operation. We call the RDD storing the vectors as RDDM. Then, we calculate the value of the last parameter $\gamma$ using the map operation and Formula (5).

In the third stage, we use the map operation to calculate the values of $E(IC)$ and SD, and the join operation to combine them into a vector. Then, we
calculate the value of $ic_{\text{min}}$ using the map operation. The RDD transformation process in the third stage is shown in Fig. 4. RDD_M in the figure is the result of the second stage.

We need to store the data in the two-by-two contingency table to facilitate the updating of the parameters after new data are obtained. Thus, we use the join operation to convert the useful data into one vector string $(\text{Drug}, \text{ADR}, a, b, c, N, ic_{\text{min}})$.

In the fourth stage, we use the map transformation, if–else statements, and value of $ic_{\text{min}}$ to detect the suspected drug–ADR signals; If $0 < ic_{\text{min}} \leq 1.5$; then it indicates a weak signal; if $1.5 < ic_{\text{min}} \leq 3.0$; then it indicates a medium intensity signal; if $ic_{\text{min}} > 3.0$; then it indicates a strong signal\[19\].

As the parallel algorithm runs on a distributed cluster, the speed of the algorithm is considerably improved. Moreover, the Spark-based parallel BCPNN algorithm reduces the number of disk I/O, which speeds up data processing further\[20\].

3.2 Improved strategy: Pre-hash partition before join

As discussed in the previous section, in the Spark-based BCPNN parallel algorithm, the join operation is used many times, which can lead to high communication costs. By default, the join operation will calculate the hash values for all keys in the two databases, transfer the records that have the same hash value to one machine over the network, and combine the records that have the same hash value on that machine. The join operation will lead to high communication costs between partitions, which not only waste a considerable amount of time but also increase the network burden.

The join operation in the third stage of the Spark-based parallel BCPNN algorithm is taken as an example. The communication among partitions is shown in Fig. 5. When the operation RDDSD join RDDIC is called, Spark initially computes the hash value for each key–value pair in each partition and subsequently sends the key–value pairs that have the same hash value as the corresponding intermediate partition. The partitions in RDDIC are processed in the same manner. Finally, the key–value pairs are joined in the intermediate partitions.

As shown in Fig. 5, every partition in RDDSD and RDDIC needs to communicate with every partition in the intermediate RDD, which requires high communication costs, leading to considerable time consumption and increase in the network burden.

We propose a method of reassigning the partition data by hashing before the join operation to reduce the communication cost caused by the join operation. For the operation RDDSD join RDDIC in the Spark-based parallel BCPNN algorithm, we use the operation partitionBy() to reassign the partition data in the RDDSD and cache them in the memory before the join operation. When the operation RDDSD join RDDIC is called, Spark will only shuffle the data in RDDIC and send each record in RDDIC to the machine that stores the records of RDDSD that has the same hash value. The communication among partitions in the two RDDs is shown in Fig. 6. Notably, every partition in RDDSD only needs to communicate with one partition. In this manner, the communication between partitions
is reduced. Thus, the running speed of the program is significantly improved and the network burden is reduced.

Algorithm 1 shows the method of pre-hash partition before join operation. First, we obtain RDDIC and RDDSD using the map operation. Then, we use the partitionBy() transformation operation to deal with the partition data. We send the Spark.HashPartitioner object to the partitionBy operation to transform the partitions in RDDSD to hash partitions. The parameter $n$ of the HashPartition refers to the number of partitions, controls the parallel number when the join operation is called, and is usually at least the same as the number of clusters. Then, the RDD should be cached in the memory. Finally, the join operation is called. This method applies to all join operations in the parallel BCPNN algorithm based on Spark.

3.3 Improved strategy: Improving shuffling parallelism

From the drug information of the dataset provided by the FAERS, we determined that, among the 21,593 kinds of drugs collected from January to June in 2016, 18,520 kinds of drugs have only less than 10 data, whereas 1 kind of drug called HUMIRA has up to 33,048 data. Most keys correspond to 10 data, whereas a few keys correspond to tens of thousands of data.

When data are shuffled, records that have the same key in all nodes will be pulled to a task in a unique node and dealt with. The task assigned only 10 data may run over in 1 s, whereas the task assigned tens of thousands of data may need one or two hours to run over. To our knowledge, the running time is decided by the task that has the longest running time. In fact, Spark becomes slow because of the data tilt. With the increase in the data, the degree of data tilt will increase, which will considerably limit the running speed and may even lead to memory overflow. Figure 7 shows the data tilt situation. We propose an optimization strategy, that is, improving the shuffling parallelism, to solve the previously mentioned problems. By increasing the shuffle task number, the keys assigned to one task can be assigned to multiple tasks. Then, each task needs to deal with less data than before. The optimized task allocation is shown in Fig. 8.

Algorithm 1 Pre-hash partition before join

Require: row RDD row data RDD(line)

Ensure: resultRDD: join result RDD

1. RDDIC → RDD.map(funIC())
2. RDDSD → RDD.map(funSD())
3. RDDSD_h → RDDSD.partitionBy(newHashPartition($n$)).cache()
4. RDDSD_hIC → (RDDSD_h).join(RDDIC)

Ensure: resultRDD: join result RDD
Algorithm 2 shows the method used to improve the shuffling parallelism to speed up data processing and prevent memory overflow because of the data tilt. After the RDD is transformed into the RDD(key, value) format using the map operation, we use the reduceByKey operation to implement the aggregation function for the RDD(key, value) format. The parameter func(value) of the reduceByKey operation refers to the aggregation function, and the parameter numofTask refers to the number of parallel tasks. Then, the RDD can be cached in the memory.

4 Experimental Results and Analysis

4.1 Experimental environment and dataset

Dataset: The experimental data are the ADR data that were collected by the FAERS from January to December in 2016[21]. The FDA began to analyze data from their own reporting system in 1993. Subsequently, in 1998, they made the data available to researchers.

Hardware and software: The experimental environment is a Spark cluster consisting of six PCs. In addition, we installed CentOS6.5, Spark1.5.1, Hadoop2.6.0, and jdk7 in the six PCs. The available memory of each PC is 4 GB. The experimental platform uses IntelliJ IDEA Community Edition 15.0.1 as a cluster development tool. For the programming language, the program is written by Scala.

4.2 Parallel algorithm performance experiment

We conducted experiments to compare the stand-alone algorithm and the Spark-based parallel algorithm. First, we applied the Spark-based BCPNN on the cluster with only one work node and the traditional BCPNN to the original dataset to compare the time consumption. The experiment shows that the parallel algorithm can significantly reduce the data processing time. Then, we extend the original dataset to 1, 2, 3, and 4 times and conduct experiments on the extended datasets to verify the performance of the parallel algorithm in dealing with large amounts of data. We called the original and extended datasets D1, D2, D3, and D4 to facilitate easy description.

Figure 9 shows that, compared with the stand-alone BCPNN algorithm, the Spark-based algorithm can save a considerable amount of time. With the increase in the dataset, the growth rate slows down, which indicates that the Spark-based algorithm is suitable for dealing with large amounts of data.

4.3 Optimized parallel algorithm performance experiment

We conducted three groups of experiments on the extended datasets D1, D2, D3, and D4 to verify the performance of the optimization strategies. The three groups of experiments are the Spark-based BCPNN, the Spark-based optimized parallel BCPNN algorithm that uses the strategy of pre-hash partition before join, and the Spark-based optimized parallel BCPNN algorithm that uses both strategies to compare the performance before and after the optimization strategies. Experiments were conducted on the cluster with only one work node.

Figure 10 shows that both strategies can efficiently reduce the time consumption. After applying the second strategy, the growth rate obviously slows down with the increase in the data. Thus, improving the shuffling parallelism is a good solution to the problem of time wasted because of the data tilt.

Then, we run the parallel BCPNN algorithm before and after optimization in the cluster with node numbers 1, 2, 3, 4, 5, and 6 on the platform to compare the acceleration ratio. We extend the original dataset to 10 times to make the experimental results more obvious.

Algorithm 2 Improving shuffling parallelism

Require: row RDD: raw data RDD (line)
1. transform RDD (line) to RDD(key, value) format
2. RDD → rowRDD.map(line ⇒ (key, value))
3. RDDSD → RDD.map(funcSD())
4. RDDintAB → RDD.reduceByKey(func(value), numofTask).cache()
5. return (result RDD)

Ensure: resultRDD: statistical result RDD.
The acceleration ratio refers to the performance improvement achieved by reducing the running time by parallel computing and is an important measure of parallel computing performance. The formula of the acceleration ratio is $T_s / T_d$, where $T_s$ represents the time spent on a single node and $T_d$ represents the time it takes to calculate the parallel algorithm\cite{22}. The higher the acceleration ratio is, the shorter the relative time for parallel computing and the higher the parallel efficiency and performance.

Figure 11 shows that the acceleration rate of the optimized algorithm is higher than that of the unoptimized algorithm. Thus, the proposed optimization strategies can effectively improve the data efficiency and computational performance.

Then, we run the Spark-based parallel BCPNN and Spark-based optimized parallel BCPNN algorithm with the available memory of 1 GB, 2 GB, and 4 GB in the original dataset to complete the performance of the second strategy in preventing memory overflow.

Table 2 shows a comparison of whether memory overflows or not before and after improving the shuffling parallelism. Table 2 also shows that the algorithm that increases the shuffling parallelism is less prone to memory overflow. Thus, the method of improving the shuffling parallelism proposed in this study can effectively prevent memory from overflowing.

In this section, we conduct experiments to compare the stand-alone BCPNN algorithm, the Spark-based parallel algorithm, and the optimized parallel algorithm. First, we apply the traditional BCPNN and the Spark-based BCPNN on the cluster with only one work node to the original dataset to compare the time consumption. We extend the original dataset to 1, 2, 3, and 4 times and conduct experiments on the extended datasets to verify the performance of parallel algorithms in dealing with large amounts of data. Then, we run the parallel BCPNN algorithm before and after optimization in the cluster with node numbers 1, 2, 3, 4, 5, and 6 on the platform to compare the acceleration ratio. The experiments show that the Spark-based parallel algorithm can significantly reduce the data processing time and has a good performance in dealing with large amounts of data. Then, we conduct three groups of experiments to verify the performance of the optimization strategies. The experiments show that both optimization strategies can efficiently reduce the time consumption. Finally, we conduct experiments to verify whether the second strategy can efficiently prevent memory overflow.

### 5 Application to a Real Project

We apply the parallel algorithm and the improved parallel algorithm proposed in this study to a real project to further verify the performance of the proposed method in ADR monitoring.

#### 5.1 Experimental environment and experimental data

**Dataset:** The dataset of the project is the real electronic medical records information from May to November in 2016 provided by a real medical consortium in China. Every record in the dataset contains the drug
and symptom information described in the unstructured text in Chinese.

**Hardware and software:** The experimental environment is the same as that described in Section 4. For the programming language, the program is written in Python in the data preprocessing phase and the program is written in Scala in the ADR detection phase.

### 5.2 Preprocessing of dataset

Before applying the parallel BCPNN algorithm, we need to conduct data preprocessing. In normal circumstances, the electronic medical records provided by hospitals or medical consortia are described in the original language, but the data required for the algorithm are the drug and symptom words. In this section, we describe a data preprocessing method for the Chinese electronic medical records to extract the symptom words from the electronic medical records before conducting the algorithmic experiments to solve the previously mentioned problem.

For extracting the symptom words of suspected ADR from the original language, our idea is to identify a standard symptom thesaurus or a standard ADR symptom thesaurus and extract symptom words from the original language by matching them with the words in the symptom thesaurus.

The symptom thesaurus that we used includes all of the symptom words in the ADR data collected by the FAERS from January to December in 2016. However, the ADR symptom words in the FAERS are written in English, and the electronic medical records that we need to extract the symptom words are written in Chinese. Thus, we need to translate the symptom thesaurus from English to Chinese. The tool that we used to translate the symptom thesaurus into Chinese is the translation API provided by Google. We only need to obtain an API developer certificate from the Google Cloud Platform Console and call the API in the program to execute the translation job.

After translating the symptom thesaurus into Chinese, we extract symptom words from the original language by matching them with the words in the symptom thesaurus. The method that we used to extract symptom words from the original language is the Aho–Corasick (AC) automata\(^{23,24}\). The AC automata was developed in the Bell Labs in 1975 and is one of the well-known multimode matching algorithms. The AC automata is based on the Trie dictionary tree and KMP pattern matching algorithm and is a multimode string matching algorithm. Its function is to derive a substring from the input string that matches the pattern string in the dictionary\(^{24}\).

The steps to extract symptom words using the AC automata are as follows:

- First, we build a tree using the dictionary string set. Each node in the tree represents a string that consists of the characters from the root to this node in order. Each time we insert a string, the original tree will be expanded. If there is no node that points to the character, then we need to create a new node in the tree.

- After all of the strings are added to the tree, the tree is completed.
- Then, we use the BFS method to calculate the Fail pointer for each node. The Fail pointer refers to the longest suffix of the current string that matches the node. Thus, the AC automata is successfully created. This effect has been widely investigated.
- After the AC automata is created, we can use it to match the existing strings. For each string, we conduct verbatim match. If the next node can match the next character, then we continue. Otherwise, we go back to the previous step until a node meets the matching conditions. If a word in the dictionary appears in the original string, then we mark that word and extract it.

By using the AC automata, we can extract the symptom words from the electronic medical records provided by hospitals or medical consortia.

### 5.3 Results of the ADR detection experiments

After data preprocessing, we obtain the dataset consisting of the drugs and corresponding symptom words. Then, we apply the stand-alone BCPNN algorithm and the Spark-based unoptimized and optimized parallel algorithms to the dataset. We use only one working node in the cluster to compare with the stand-alone BCPNN. The time consumption of each algorithm is shown in Fig. 12.

Figure 12 shows that the Spark-based parallel algorithm saves a considerable amount of time than the stand-alone algorithm. Moreover, the optimized parallel algorithm is less time consuming than the unoptimized parallel algorithm.

The data show the time spent by the algorithms on the dataset with only one work node in the cluster, and it takes 28 min to complete the data processing using the optimized parallel BCPNN algorithm. To speed up data processing, we apply the optimized parallel BCPNN
algorithm to the dataset with six nodes in the cluster, and it only takes 8 min.

Thus, the application to a real project further proves the performance and practical value of the proposed algorithm in ADR detection.

6 Conclusion

In this study, we propose a new algorithm for ADR detection based on Spark and BCPNN, which overcomes the problem of signal detection on large amounts of ADR data. Then, we propose two optimization strategies for the Spark-based parallel BCPNN algorithm, namely, pre-hash partition before join, which solves the problem of the slow running speed because of the join operation, and improving the shuffling parallelism to speed up data processing and prevent memory overflow caused by the data tilt. The experiments show that the method can effectively increase the detection speed, improve the acceleration ratio, and prevent memory overflow. Then, we apply the algorithms to a real project. We introduce a data preprocessing method for electronic medical records and conduct experiments to apply the algorithms to the preprocessed dataset. The experiments further prove the performance and practical value of the proposed algorithm.

Acknowledgment

This work was supported in part by the Science and Technology Innovation Action Plan Project of Science and Technology Commission of Shanghai Municipality (No. 18511102703) and the Scientific Research Plan Project of Science and Technology Commission of Shanghai Municipality (No. 16JC1400803).

References

[1] A. Bate, M. Lindquist, I. R. Edwards, and R. Orre, A data mining approach for signal detection and analysis, *Drug Saf.*, vol. 25, no. 6, pp. 393–397, 2002.
[2] A. Bate, The use of a Bayesian confidence propagation neural network in pharmacovigilance, PhD dissertation, Umeå University, Sweden, 2003.
[3] S. Karimi, C. Wang, A. Metke-Jimenez, R. Gaire, and C. Paris, Text and data mining techniques in adverse drug reaction detection, *ACM Computing Surveys*, vol. 47, no. 4, p. 56, 2015.
[4] W. G. Chen and J. X. Deng, A study on signal detection and automatic warning algorithm for adverse drug reaction, in *Proc. 2008 International Conference on Computer Science and Software Engineering*, 2008.
[5] K. V. Swetha, S. Sathyadevan, and P. Bilna, Network data analysis using spark, in *Software Engineering in Intelligent Systems*, R. Silhavy, R. Senkerik, Z. Oplatkova, Z. Prokopova, and P. Silhavy, eds. Springer, 2015, pp. 253–259.
[6] A. Bate, M. Lindquist, I. R. Edwards, S. Olsson, R. Orre, A. Lansner, and R. M. De Freitas, A Bayesian neural network method for adverse drug reaction signal generation, *Eur. J. Clin. Pharmacol.*, vol. 54, no. 4, pp. 315–321, 1998.
[7] N. Farahini, A. Hemani, A. Lansner, F. Clermidy, and C. Svensson, A scalable custom simulation machine for the Bayesian Confidence Propagation Neural Network model of the brain, in *Proc. 2014 19th Asia and South Pacific Design Automation Conf.*, Singapore, 2014, pp. 578–585.
[8] B. Honigman, J. Lee, J. Rothschild, P. Light, R. M. Pulling, T. Yu, and D. W. Bates, Using computerized data to identify adverse drug events in outpatients, *J. Am. Med. Inform. Assoc.*, vol. 8, no. 3, pp. 254–266, 2001.
[9] L. Duan, M. Khoshneshin, W. N. Street, and M. Liu, Adverse drug effect detection, *IEEE Journal of Biomedical & Health Informatics*, vol. 17, no. 2, pp. 305–311, 2013.
[10] D. W. Bates, R. S. Evans, H. Murff, P. D. Stetson, L. Pizziferri, and G. Hripcsak, Detecting adverse events using information technology, *Journal of the American Medical Informatics Association*, vol. 10, no. 2, pp. 115–128, 2003.
[11] Z. J. Han and Y. J. Zhang, Spark: A big data processing platform based on memory computing, in *Proc. 7th Int. Symposium on Parallel Architectures, Algorithms and Programming*, Nanjing, China, 2015, pp. 172–176.
[12] A. Solovyev, M. Mikheev, L. M. Zhou, J. Dutta-Moscato, C. Ziraldo, G. An, Y. Vodovotz, and Q. Mi, SPARK: A framework for multi-scale agent-based biomedical modeling, *International Journal of Agent Technologies & Systems*, vol. 2, no. 3, pp. 18–30, 2010.
[13] P. Cao, Optimization and implementation of clustering algorithm based on Spark platform, (in Chinese), master degree dissertation, Beijing Jiaotong University, Beijing, China, 2016.
[14] A. Zhang, *Insider of Spark Technology*, (in Chinese). Beijing, China: Mechanical Industry Press, 2015.
[15] J. Fu, J. W. Sun, and K. Y. Wang, SPARK—A big data...
processing platform for machine learning, in Proc. 2016 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration, Wuhan, China, 2017, pp. 48–51.

[16] T. Brewer and G. A. Colditz, Postmarketing surveillance and adverse drug reactions: Current perspectives and future needs, JAMA, vol. 218, no. 9, pp. 824–829, 1999.

[17] A. Lansner and Ö. Ekeberg, A one-layer feedback artificial neural network with a Bayesian learning rule, International Journal of Neural Systems, vol. 1, no. 1, pp. 77–87, 1989.

[18] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, Learning Spark: Lightning-Fast Data Analysis, (in Chinese). Beijing, China: People’s Posts and Telecommunications Press, 2015.

[19] W. Y. Li, Research on apache spark for big data processing, (in Chinese), Modern Computer, no. 8, pp. 55–60, 2015.

[20] S. L. Xie, Research and application of distributed ETL based on spark, (in Chinese), master degree dissertation, Donghua University, Shanghai, China, 2017.

[21] FDA adverse event reporting system (FAERS): Latest quarterly data files, https://www.fda.gov/Drugs/GuidanceComplianceRegulatoryInformation/Surveillance/AdverseDrugEffects/ucm082193.htm, 2017.

[22] Z. F. Wu, T. Zhang, and Y. Xiao, Improvement and parallel implementation of K-means clustering algorithm based on the Spark platform, (in Chinese), China Internet, no. 1, pp. 44–50, 2016.

[23] P. F. Wang and L. Li, Research on multi-pattern matching algorithms based on Aho-Corasick algorithm, (in Chinese), Application Research of Computers, vol. 28, no. 4, pp. 1251–1253&1259, 2011.

[24] A. V. Aho and M. J. Corasick, Efficient string matching: An aid to bibliographic search, Communications of the ACM, vol. 18, no. 6, pp. 333–340, 1975.

Li Sun is an associate professor at School of Computer Science and Technology, Donghua University, Shanghai, China. She received the MS degree in computer science from Donghua University in 1986. Her main research interests include database, data mining, and artificial intelligence technology.

Shan Sun received the BEng degree from Shanghai Ocean University of Comuper Science and Technology, in 2015. She is currently a master student in Donghua University, China. Her main research interests include data mining, data warehouse, etc.

Tianlei Wang is now a master student in University of Southern California. He received the bachelor degree from Donghua University in 2018. He was also a silver medalist of ACM-ICPC Asian regional contest. His main research interests include data mining, text analysis, etc.

Jiyun Li received the MS and PhD degrees from Donghua University in 1996 and 2003, respectively. Currently she is a professor of computer science and technology in Donghua University. Her research interests include data engineering, machine learning, etc.

Jingsheng Lin received the BEng degree from East China University of Science and Technology in 2009, and the Engineering Master degree from Donghua University in 2011. Now he is working at Ruijin Hospital Affiliated to Shanghai Jiao Tong University. His main research interests include deep learning, artificial intelligence, intelligent medical, etc.