A Hybrid Distributed Frequent Itemset Mining Method with Its Application in Medical Diagnosis

Mingxue Zhang and Fuyuan Xiao(✉)

School of Computer and Information Science, Southwest University, No. 2 Tiansheng Road, Beibei District, Chongqing 400715, China
doctorxiaofy@hotmail.com, xiaofuyuan@swu.edu.cn

Abstract. The world has entered an era of globalization, which represents the explosion of information. The application and development of big data technology make the scale of medical data geometric growth. People cannot intuitively see the correlation and the implicit relationship between complex medical data, which leads to the situation of more data and less knowledge. By using Spark and a variety of data pre-processing techniques and machine learning related algorithms, we implemented a platform which could help patients recommend more accurate treatment plans, help doctors analyze the relationship between diseases, and provide more natural results through a visual interface. Besides, we proposed a distributed frequent itemset mining algorithm (DSDFIM) based on the adjacency list and Spark. After evaluating, the proposed algorithm could reduce data transportation once between main memory and secondary storage, and improved the speed of data processing through distributed computing, compared with the classic algorithm. Meanwhile, it could solve the problem of merging frequent itemset of the same item under different independent paths.

Keywords: Data mining · Medical data · Frequent item mining

1 Introduction

With the development of big data, especially the popularity of wearable intelligent devices, making the scale of medical data increases geometrically. Due to the complexity of medical data, people cannot get the formal and professional medical services. Besides, through the search engine on the Internet, the search results of illness and symptoms often mislead patients, which would increase their worries. Applying data mining in the medical field to actual systems can serve both doctor and patients [1]. In order to find the relationship between data, many association algorithms, such as Apriori and FP-Growth [2], and analysis system have been built.

This research is supported by the National Natural Science Foundation of China (No. 62003280), Research Project of Education and Teaching Reform in Southwest University (No. 2019JY053), Fundamental Research Funds for the Central Universities (No. XDJK2019C085), and Chongqing Overseas Scholars Innovation Program (No. cx2018077).

Qiu proposed the YAFIM algorithm, which accelerated the speed of mining through parallel computing [3]. Shi used Spark's engine and distributed mining further to expedite the processing speed of YAFIM [4]. Miao launched a distributed FP-Growth algorithm based on the Spark [5]. Zhang used unique weights and constrained subtrees to avoid scanning to obtain conditional pattern bases and establish FP-Trees [6]. Li calculated the weights for each item separately and constructs FP-Tree [7]. Yin used adjacency lists to reduce I/O once [8]. Li improved the shortcomings of weighted frequent itemset for mining and storage by using the weighted support of records in the two-dimensional table. This method saved the process of searching the first conditional pattern library by traversing the weighted FP tree [9]. Xiao integrated Dempster-Shafer theory with belief entropy, the new method could help to decrease the uncertainty caused by subjective human cognition to improve decision making [10]. Xiao proposed a generalized intelligent quality-based approach for fusing multi-source information, which could fuse the multi-complex-valued information while maintaining a high quality of the fused result [11]. Gao improved the accuracy of uncertainty measuring by optimizing Tsallis entropy [12]. Liu improved the result of detecting the number of unknown targets by proposing a new method based on Elbow method [13]. Li proposed a new method based on the similarity measure. The aggregation operator is presented according to Pythagorean fuzzy information, instead of being provided in advance by decision makers, which can reduce human subjectivity [14].

Our goal is to use Spark engine and machine learning algorithms to analyze medical data, which could provide accurate treatments and analysis results for patients and doctors. In this research process, the following issue will be studied and solved:

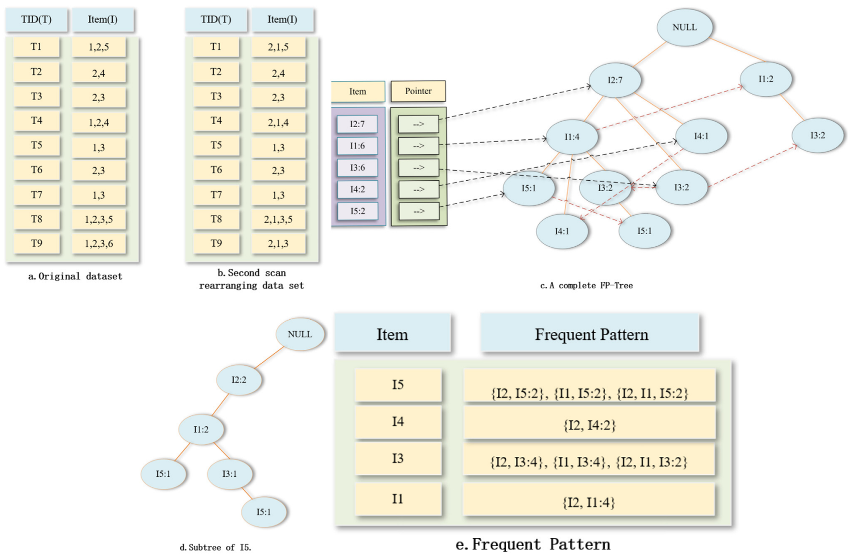
1) Medical data processing: Because medical records have characters of colloquialism and inconsistent, this study used word segmentation tools, synonyms tools, unique hot codes, and regularization methods to characterize the data set. Besides, we used Spark's productive machine learning library to complete the recommendation of the optimal treatment method based on the features provided by the user.
2) Frequent item mining: The traditional FP-Growth algorithm does not make corresponding optimization on the medical data set. And by using the adjacency list as the storage structure can reduce the I/O times between the main memory and the secondary storage. Still, it has an inaccurate problem in meagering frequent itemset with the same item stored in multiple independent paths. We synthesized several frequent itemset mining methods [4, 5, 8] and proposed a hybrid method called DSDFIM, which used adjacency list and matrix as structure and Spark for distributed computing and divided the data into disease and symptom by tags to solve the problem.

2 FP-Growth

2.1 Scan Database and Building FP-Tree

First, the algorithm scans the data set stored in the database on the hard disk (Fig. 1a) to obtain all frequent items and stores them in the header table in the descending order of minimum support. We assume the minsupport = 2.

The algorithm secondly scans each transaction in the database, deletes the items in the original item set that are less than the minimum support. The remaining items are arranged in descending order of their frequency (Fig. 1b). After obtaining the rear-ranged data set, it reads the transactions in sequence. First, T1 = {2, 1, 5} is read. The algorithm creates a path of $\emptyset \rightarrow$ I2 $\rightarrow$ I1 $\rightarrow$ I5, and sets each degree counts as 1 because the path is independent. Secondly, the algorithm continues reading in sequence, increases item's degree by 1 when the path it had with the previous overlapping, and accessed from the empty set if there is no coincident path. The algorithm continues to read transactions according to this idea until the end, and gets a complete FP-Tree.



TID(T)	Item(I)
T1	1,2,5
T2	2,4
T3	2,3
T4	1,2,4
T5	1,3
T6	2,3
T7	1,3
T8	1,2,3,5
T9	1,2,3,6

a. Original dataset

TID(T)	Item(I)
T1	2,1,5
T2	2,4
T3	2,3
T4	2,1,4
T5	1,3
T6	2,3
T7	1,3
T8	2,1,3,5
T9	2,1,3

b. Second scan rearranging data set

Item	Pointer
I2:7	-->
I1:6	-->
I3:6	-->
I4:2	-->
I5:2	-->

Item	Frequent Pattern
I5	{I2, I5:2}, {I1, I5:2}, {I2, I1, I5:2}
I4	{I2, I4:2}
I3	{I2, I3:4}, {I1, I3:4}, {I2, I1, I3:2}
I1	{I2, I1:4}

e. Frequent Pattern

Fig. 1. The flow of FP-Growth

2.2 Generating Conditional Pattern Base and Frequent Itemset

After building, the algorithm digs up from the item with the minimum support degree in the item header table until the item with the second max support degree. The conditional pattern base is a subset consisting of the prefix path with the lowest node (suffix) in the FP tree. After obtaining the FP subtree. The algorithm assigns the value of the leaf node to each node above it, and deletes the node whose count is lower than the minimum support. From this conditional pattern base, frequent itemset can be obtained by recursive mining. Taking *I5* as an example (Fig. 1d). First, it uses *I5* as the leaf node to get the subtree. Because *I5* has two nodes, when the common ancestor nodes on the path of the two nodes are consistent, its value becomes the sum of the leaf nodes. The conditional pattern base is used to create conditional frequent item tree. The algorithm merges the same item on the same side and deletes the item that does not meet the minimum support. Finally, a frequent itemset is produced through a conditional frequent items tree. The items in the tree are merged to create the frequent items. Taking item *I3* as an example, although *I2* and *I3* both have a support level of 4, when the combination is {*I2, I1*}, the support for *I3* cannot be added together because they come from two different paths. When calculating this, we should use 2 as degree. Thus, the result was shown in Fig. 1e.

3 DSDFIM Algorithm

3.1 Principle of DSDFIM Algorithm

Due to the FP-Growth algorithm have to scan the database twice, the operating efficiency is not very ideal. We propose a Disease and Symptom Distributed Frequent Itemset Mining Algorithm Based on Adjacency List and Spark (DSDFIM). Because most medical records only recorded one disease, and the number of documented complications only accounted for 23% of the entire data set. The items in each transaction in the data set are labeled based on the medical dictionary. After that, the data is distributed and calculated to obtain the adjacency list and frequent itemset, and then combined to get the complete frequent item set. The entire process is shown in Fig. 2.

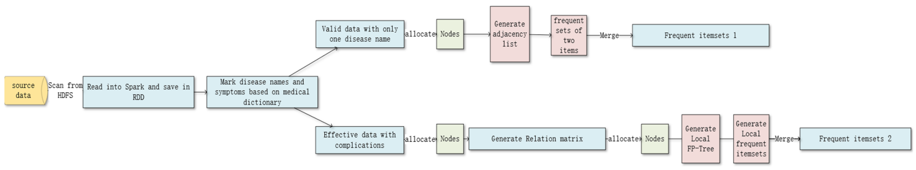
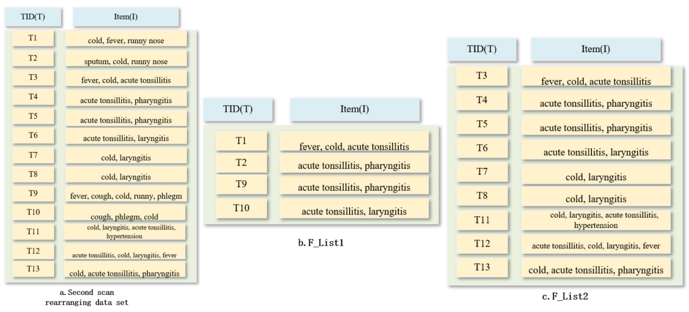

source data
Scan from HDFS
Read into Spark and save in RDD
Mark disease names and symptoms based on medical dictionary
Valid data with only one disease name
allocate
Nodes
Generate adjacency list
frequent sets of two items
Merge
Frequent itemsets 1
Effective data with complications
allocate
Nodes
Generate Relation matrix
allocate
Nodes
Generate Local FP-Tree
Generate Local frequent itemsets
Merge
Frequent itemsets 2

Fig. 2. The flow of DSDFIM Algorithm

For Valid Data with Only One Disease

Step 1. The algorithm obtains data from HDFS, and reads it into Spark and store it in RDD. Then, it scans the data, marks the disease name, and replaces the first element in the thing and store it in *F_List1*. As shown in Fig. 3.



TID(T)	Item(I)
T1	cold, fever, runny nose
T2	sputum, cold, runny nose
T3	fever, cold, acute tonsillitis
T4	acute tonsillitis, pharyngitis
T5	acute tonsillitis, pharyngitis
T6	acute tonsillitis, laryngitis
T7	cold, laryngitis
T8	cold, laryngitis
T9	fever, cough, cold, runny, phlegm
T10	cough, phlegm, cold
T11	cold, laryngitis, acute tonsillitis, hypertension
T12	acute tonsillitis, cold, laryngitis, fever
T13	cold, acute tonsillitis, pharyngitis

a. Second scan rearranging data set

TID(T)	Item(I)
T1	fever, cold, acute tonsillitis
T2	acute tonsillitis, pharyngitis
T9	acute tonsillitis, pharyngitis
T10	acute tonsillitis, laryngitis

b. F_List1

TID(T)	Item(I)
T3	fever, cold, acute tonsillitis
T4	acute tonsillitis, pharyngitis
T5	acute tonsillitis, pharyngitis
T6	acute tonsillitis, laryngitis
T7	cold, laryngitis
T8	cold, laryngitis
T11	cold, laryngitis, acute tonsillitis, hypertension
T12	acute tonsillitis, cold, laryngitis, fever
T13	cold, acute tonsillitis, pharyngitis

c. F_List2

Fig. 3. The result of splitting the dataset

Step 2. The algorithm divides *F_List1* into *G* groups, named *g1, g2 … gk* (*k* < nodes.num ()) to the computing nodes. After the split, the Spark engine can be used to calculate each group, and the adjacency list of each group can be obtained very quickly. The items that do not meet the requirements are deleted according to the minimum support. As shown in Fig. 4.

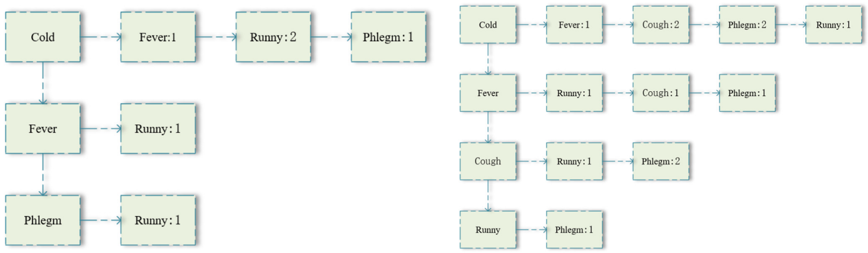
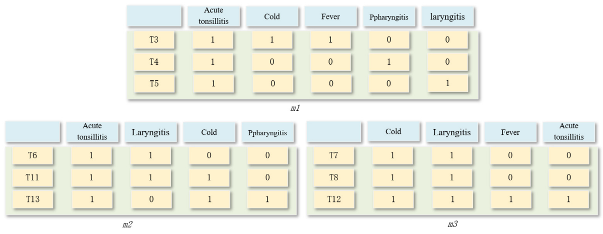


Fig. 4. The adjacency List of *g1* and *g2*

Step 3. The algorithm obtains binary frequent itemset from the two adjacency lists of *g1* and *g2*, which are {*cold, fever: 1*}, {*cold*, *runny: 2*}, {*cold*, *phlegm: 1*}, {*flow spit*, *fever*: 1}, {*runny*, *phlegm: 1*}; {*cold*, *fever: 1*}, {*cold*, *cough: 2*}, {*cold*, *phlegm: 2*}, {*cold*, *runny: 1*}, {*fever*, *runny: 1*}, {*fever*, *cough: 1*}, {*fever*, *phlegm: 1*}, {*cough*, *phlegm: 2*}, {*cough*, *runny: 1*}, {*runny*, *sputum: 1*}.
Step 4. The algorithm integrates binary frequent item sets in different nodes. According to the Apriori algorithm, it can be known that all its supersets are infrequent item sets when an item set is an infrequent item set. The minimum support for this combination can be combined with binary frequent item sets to obtain higher element frequent items sets as {*sputum, runny, cold:* 2}, {*cough*, *sputum*, *cold:* 2}, {*fever*, *runny*, *cold:* 2}. So the final result is: {*cold, fever:* 2}, {*cold, runny: 3*}, {*cold, sputum: 3*}, {*cold, cough:* 2}, {*fever, cough:* 2}, {*cough, Sputum:* 2}, {*sputum*, *runny*: 2}, {*sputum*, *runny, cold:* 2}, {*cough*, *sputum*, *cold:* 2}, {*fever*, *runny*, *cold:* 2}.

For Valid Data with Multiple Diseases

Step 1. The algorithm scans the data. If there are multiple disease names in the transaction, the data would be stored in *F_List2.*
Step 2. The algorithm splits *F_List2* into *m1, m2 … mn* (*n* < nodes.num ()) to form a matrix, and sorts according to the frequency. As shown in the Fig. 5.
Step 3. The algorithm combines m1, m2, and m3 to obtain a complete matrix and a Header Table based on this matrix. As shown in Fig. 6.
Step 4. The algorithm splits the data set and assigns it to each node for the calculation to generate a local frequency tree. This part uses the method implemented in DFPS to ensure that each item could be computed accurately. By using the DFPS algorithm, frequent itemset can be mined in parallel without delivering messages. Finally, the algorithm generates local frequency itemset.



	Acute tonsillitis	Cold	Fever	Ppharyngitis	laryngitis
T3	1	1	1	0	0
T4	1	0	0	1	0
T5	1	0	0	0	1

m1

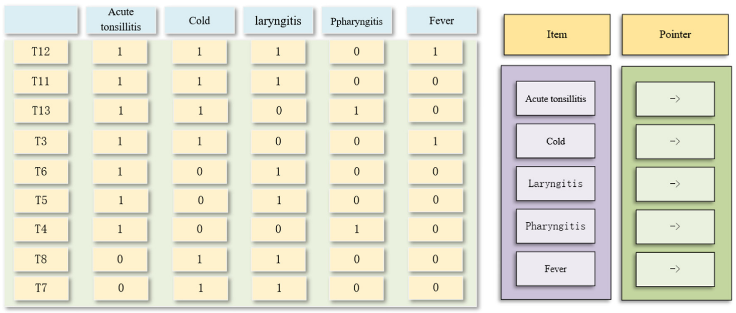
	Acute tonsillitis	Laryngitis	Cold	Ppharyngitis
T6	1	1	0	0
T11	1	1	1	0
T13	1	0	1	1

m2

	Cold	Laryngitis	Fever	Acute tonsillitis
T7	1	1	0	0
T8	1	1	0	0
T12	1	1	1	1

m3

Fig. 5. The matrix of *m1*, *m2* and *m3*



	Acute tonsillitis	Cold	laryngitis	Ppharyngitis	Fever
T12	1	1	1	0	1
T11	1	1	1	0	0
T13	1	1	0	1	0
T3	1	1	0	0	1
T6	1	0	1	0	0
T5	1	0	1	0	0
T4	1	0	0	1	0
T8	0	1	1	0	0
T7	0	1	1	0	0

Item	Pointer
Acute tonsillitis	->
Cold	->
Laryngitis	->
Pharyngitis	->
Fever	->

Fig. 6. Header table

Algorithm1: Generate frequent item for the data with only one disease
Input: dataset, minsupport=2
Output: Frequent itemset

```
1)Read file
2)Set label according to the Medical Dictionary as F_List1RDD
3)Distributed F_List1RDD into different Groups: Gk
4)Map<Node: String, Map<Node: String, edge: Int >>//Store the adjacency list
5)foreach transaction t in Gk
6)    foreach item Ii in t
7)      if Ii in Map
8)          if Ij in list where j≠i
9)              lj.edge++
10)         else
11)             add(lj, 1)
12)     else
13)         add li to a new list in Map
14)   end
15)end
16)foreach item Ii in list
17)   foreach lj in item
18)       if li.edge <minSupport
19)           del(lj)
20)       else if (Ii, Ij) ∉ frequent set
21)           add((Ii, Ij) into frequent set
22)           if (lj, li) in Map
23)               del(lj, li)
24)     end
25)end
26)aggregating local FP of each Node
27)foreach ItemSet (Ii,lj) in frequent set //loop until generate all frequent set
28)     if (Ij,li) in frequent set
29)           (Ii,lj).degree++
30)           del((Ij,li))
31)     foreach ItemSet(li, lk) in frequent set
32)           if (lj, lk) or (lk, lj) ∉frequent set
33)                (li, lj, lk).weight=min((Ii,lj).degree, (Ii,lk).degree, (lj, lk).degree)
34)     end
35)end
```

Algorithm2: Generate frequent item for the data with only one disease
Input: dataset, minsupport=2
Output: Frequent itemset

```
1)Read file
2)Set label according to the Medical Dictionary as F_List2RDD
3)Distributed F_List2RDD into different Groups: Gk
4)matrix :Int // Storage matrix information
5)foreach transaction t in Gk
6)   foreach item Ii in t
7)       j=0;
8)       do{
9)            if map(I, j).name==li
10)               map(I, j)==1
11)           else
12)               j++
13)       }while(j==t.length)
14)   end
15) end
16)aggregating local matrix of each Node
17)Map<item: String, freq: Int> // Build FP-Tree
18)foreach j in item.value
19)   Set map< matrix (0,j).name, 0>
20)   foreach i in transaction.value
21)     if matrix (j,i)==0
22)        freq++
23)   end
24)end
25)Generate Header Table
26)Call for DPFS to generate frequent itemset
```

Fig. 7. DSDFIM algorithm

Step 5. The DSDFIM algorithm is shown in Fig. 7. The algorithm combines local frequent itemset to get the final result: {*acute tonsillitis, fever: 2*}, {*cold, fever: 2*}, {*acute tonsillitis, pharyngitis: 2*}, {*acute tonsillitis, Laryngitis: 4*}, {*cold, laryngitis: 4*}, {*acute tonsillitis, pharyngitis: 4*}, {*acute tonsillitis, cold, laryngitis: 2*}, {*Acute tonsillitis, Cold, fever: 2*}.

4 Experimental Results Analysis

4.1 Algorithm Performance

Accuracy. The proposed algorithm could solve the problem that the frequent combination of frequent itemset based on adjacency list or matrix in multiple independent paths by distinguishing disease names and disease characteristics and using different processing methods in different situations. For example, in the data set of Fig. 1e, the method proposed in [8] will get the frequent itemset, like {*I2, I1, I3: 4*}. However, in the correct FIM algorithm, the data set {*I1, I3*} comes from different independent path, and the frequency cannot be directly added during the combination. Thus, the correct result is {*I2, I1, I3: 2*}.

When processing this kind of data, we divide them into the second category according to the assumption and use traditional algorithms to calculate, which can avoid this problem. Through the DSDFIM algorithm, the results are shown in Fig. 8 and 9, which can see the relationship compared with the original dataset, it indicates that cold and acute tonsillitis have a very close relationship and often appear as complication. Thus, the DSDFIM algorithm could make the relationship between the data easier to explain, and could help doctors and patients find a better treatment, the association of complications and the invisible connection between the data.

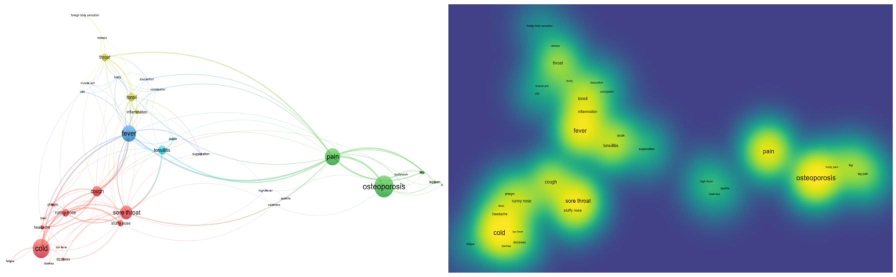


Fig. 8. Relationship of the data set before DSDFIM

Running Time. The adjacency list used in this article has a complexity of O (1) when adding nodes and edges. Although the complexity is O (N + E) at the time of construction compared to the storage structure of the tree and the time consumption is not significantly optimized, even under partial support compared with the PFP algorithm calculated in parallel, the calculation time is significantly improved compared to the traditional FP-Growth and Apriori algorithms due to the distributed computing and reducing I/O between main memory and secondary storage once.

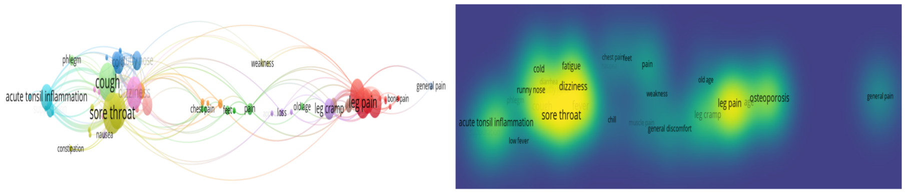


Fig. 9. Relationship of the data set after DSDFIM

Memory Usage. The algorithm used adjacency lists and matrices as the storage structure. By using the characteristics of RDD in Spark. The algorithm scanned the data set only once. Besides, by using Spark to distributed compute on multiple nodes, it could significantly ease the pressure of main memory.

4.2 Medical Data Analysis

Through the DSDFIM algorithm proposed in this article, it can be intuitively found that the association of complications and the invisible connection between the data. Diseases such as osteoporosis have a very low probability of being related to the first two diseases in characteristics, and these are consistent with our cognition from life.

Besides, according to statistical analysis of the previously processed data, we used one-hot codes to encode data, and the platform could analyze and transform feature vectors from various dimensions to the same dimension. For example: {upper respiratory tract infection, low fever} The corresponding one-hot codes are:

{*100000000000000000000000000000000000000000000000000000000000000000000, 10000000000000000000000000000000000000000000000000000*}

We converted two numbers into decimal base:

{*295150156979166511104, 4503599627370496*}

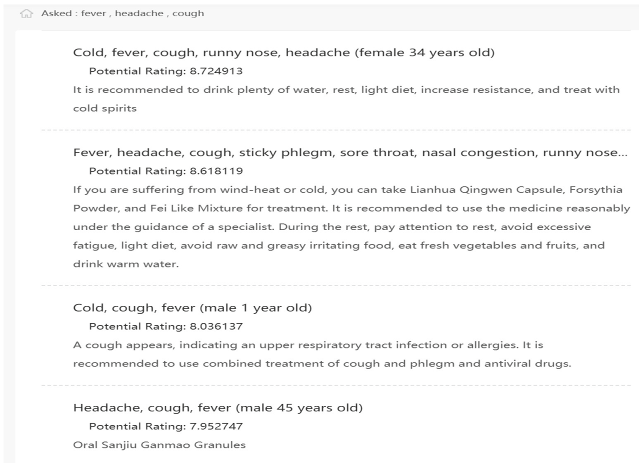
At the same time, we complemented the features by using frequent itemset obtained by DSDFIM. It is used to shrink the feature dimensions to 4. For example, the original input feature of the user is {sneeze, runny nose, stuffy nose}. Then the available prediction from frequent itemset is {sore throat, cold}. Therefore, the first feature in the prediction is added to get {sneezing, runny nose, stuffy nose, sore throat}. Currently, the decimal value corresponding to the feature vector is listed below:

{*604462909807314587353088, 302231454903657293676544, 151115727451828646838272, 590295810358705651712*}

For the features in the dataset that are higher than four dimensions, we used the PCA algorithm for data dimensionality reduction. Finally, the value of the previously converted decimal feature matrix is too large, which is not conducive for data analysis. Therefore, the z-score standardized method is used to convert the data to obtain the characteristic effect. The result is shown below:

{*1.52180576, 0.16850216, −0.50814964, −1.18215827*}

After the feature vector dimension is unified and normalized, the features are trained by K-Means algorithm. We specified the number of clusters is 3, and set 20 as the iteration round. According to the characteristics previously input by the user and the result of clustering, we could get a user group which is similar with the user. At the same time, because the user in the medical forum will get multiple answers, each data is weighted according to the order as a benchmark, and this was defined as the patient's preference for different treatments. Then we used the ALS algorithm to perform matrix decomposition and collaborative filtering and recommend their treatment plan to the user, as well as getting the user's potential for the treatment plan and score. As shown in the Fig. 10.


Asked : fever , headache , cough

Cold, fever, cough, runny nose, headache (female 34 years old)
Potential Rating: 8.724913
It is recommended to drink plenty of water, rest, light diet, increase resistance, and treat with cold spirits

Fever, headache, cough, sticky phlegm, sore throat, nasal congestion, runny nose...
Potential Rating: 8.618119
If you are suffering from wind-heat or cold, you can take Lianhua Qingwen Capsule, Forsythia Powder, and Fei Like Mixture for treatment. It is recommended to use the medicine reasonably under the guidance of a specialist. During the rest, pay attention to rest, avoid excessive fatigue, light diet, avoid raw and greasy irritating food, eat fresh vegetables and fruits, and drink warm water.

Cold, cough, fever (male 1 year old)
Potential Rating: 8.036137
A cough appears, indicating an upper respiratory tract infection or allergies. It is recommended to use combined treatment of cough and phlegm and antiviral drugs.

Headache, cough, fever (male 45 years old)
Potential Rating: 7.952747
Oral Sanjiu Ganmao Granules

Fig. 10. Recommend result

5 Conclusions

This research used the Spark engine to analyze medical data, and we proposed a hybrid algorithm, called DSDFIM algorithm, which split and calculated the data through a distributed engine. Meanwhile, the disease names and disease features in the data set were labeled and the data set was split according to different situations. The adjacency list and matrix were used to replace the original prefix tree storage structures. Through verification, this method optimized problem of meagering frequent item sets with the same item under multiple independent paths and reduced the I/O of disk and memory once and had a high processing efficiency compared with the traditional FP-growth comparison. Besides, the data platform integrated the methods in Spark's ML library and combined them to realize the function of data analysis. There is still room for the optimization of the proposed algorithm in this paper, such as reducing the number of iterations when mining frequent itemset, load balancing strategies in the distributed system, and scalability that means expanding the classic data set, such as mushroom and chess.

References

1. Ding, Z., Liu, Y., Jing, S., Zhang, X.: A review of medical data mining. Smart Health **2**(10), 54–56 (2016)
2. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: ACM SIGMOD International Conference on Management of Data, pp. 1–12. ACM Press, New York (2000)
3. Qiu, H., Gu, R., Yuan, C., Huang, Y.: YAFIM: a parallel frequent itemset mining algorithm with spark. In: 2014 IEEE International Parallel & Distributed Processing Symposium Workshops, Phoenix, AZ, USA, pp. 1664–1671. IEEE Computer Society (2014)
4. Shi, X., Chen, S., Yang, H.: DFPS: distributed FP-growth algorithm based on Spark. In: 2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Chongqing, China, pp. 1725–1731. IEEE (2017)
5. Miao, Y., Lin, J., Xu, N.: An improved parallel FP-growth algorithm based on Spark and its application. In: 2019 Chinese Control Conference (CCC), Guangzhou, China, pp. 3793–3797. IEEE (2019)
6. Zhang, W., Luo, K.: A parallel FP-growth mining algorithm based on Spark framework. Comput. Eng. Sci. **39**(08), 1403–1409 (2017)
7. Li, Y., Yin, S.: Mining algorithm for weighted FP-growth frequent item sets based on ordered FP-tree. Int. J. Eng. Manag. Res. **9**(5), 154–158 (2019)
8. Yin, M., Wang, W., Liu, Y., Jiang, D.: An improvement of FP-Growth association rule mining algorithm based on adjacency table. MATEC Web Conf. **189**, 10012 (2018)
9. Li, Y., Yin, S.: Mining algorithm for weighted FP-tree frequent item sets based on two-dimensional table. J. Phys. Conf. Ser. **1453**, 012002 (2020)
10. Xiao, F.: EFMCDM: evidential fuzzy multicriteria decision making based on belief entropy. IEEE Trans. Fuzzy Syst. **28**, 1477–1491 (2020)
11. Xiao, F.: GIQ: a generalized intelligent quality-based approach for fusing multi-source information. IEEE Trans. Fuzzy Syst. (2020). https://doi.org/10.1109/TFUZZ.2020.2991296
12. Gao, X., Liu, F., Pan, L., Deng, Y., Tsai, S.-B.: Uncertainty measure based on Tsallis entropy in evidence theory. Int. J. Intell. Syst. **34**(11), 3105–3120 (2019)
13. Fan, L., Deng, Y.: Determine the number of unknown targets in Open World based on Elbow method. IEEE Trans. Fuzzy Syst. (2020). https://doi.org/10.1109/TFUZZ.2020.2966182
14. Fei, L., Deng, Y.: Multi-criteria decision making in Pythagorean fuzzy environment. Appl. Intell. **50**, 537–561 (2019). https://doi.org/10.1007/s10489-019-01532-2