2004－ICS－138　(15)
2004／12／4

Preprocessing Planning for Data Mining

Atsushi Shiro,† Masayuki Numao†† and Cholwich Nattee††

Due to accumulating data in computer networks, data mining is recently in the spotlight as an effective data processing technology. However, AI technology is hardly used except for the data analysis. The automation of preprocessing using AI research is expected. Planning system has three inputs: descriptions of the world , the agent's goal and the possible actions. The planner's output is a sequence of actions which achieve the goal. We define the preprocessing process using metadata where the characteristics of the data are extracted, and propose how to carry out automatic preprocessing by using planning system and metadata which is used for a description of the world and the goal.

1. Introduction

Recent development of computer and network helped decrease the cost of data gathering and enabled automatic access to huge amount of new data. However the data are collected rather incidentally than intentionally for specific purposes and not necessarily in a form suitable for analysis algorithm, which makes analysis difficult as it is. There rises a need for preprocessing such as data cleaning, re-editing, and data preparation. This preprocessing has a critical influence on data analysis results and is considered as one of the most important steps, occupying 60% of the whole processing time. Despite many active researches on data analysis, not many have been made comprehensively on preprocessing except minor researches on individual data cleaning steps such as data-merging and missing value filling.

We developed TransX system utilizing XML in order to automatize preprocessing by AI methods. In this system, having XML as input, users can preprocess by filtering as they watch simplified XML tree structures called "unit tree". We find out what kind of processing was performed for similar data structure in the past by reviewing the history and specify a possible filter to be applied next. But in reality, history was presented only when they had the same data structure, failing to function effectively.

So we first formulated preprocessing and applied the formula to extract data characteristics. Then we formulated preprocessing utilizing the extracted metadata. Planning system automatically generates action called "plan" when given initial state, goal state and operator set. Taking advantage of this nature, we propose a method for automatic preprocessing using metadata where we define metadata of existing dataset as "initial state", metadata of the aimed data as "goal state" and operation for metadata as "operator set". We implemented this method as actual system with suggestions for solving problems expected to arise. Then we applied this method to real-world data as case study to verify the effectiveness.

2. Preprocessing Formulation

We formulate preprocessing using metadata with the idea: "preprocessing is a combination of many small data processing operation". First we define processing for file or a set of file

Definition 2.1 (Command)
Processing for file f or set of file F is called command, described as c. When command c is applied to F and the obtained result is F', it is shown as

$$F' = c(F)$$

One command c is a unit of operation no matter how big it is; it can be a single query by SQL or a comparatively large processing using special tools. Using this command, the following command sequence can be defined.

Definition 2.2 (Command sequence)
When commands are executed in sequence for set of file F and F' is generated, it is shown as

† Department of Computer Science, Tokyo Institute of Technology
†† The Institute of Scientific and Industrial Research, Osaka University

$$F' = c_n(c_{n-1}(c_{n-2}(...c_1(F)...))) = \bigodot_{k=1}^{n} c_k(F)$$

where $\bigodot_{k=1}^{n} c_k$ *is called command sequence.*

All above leads us to define preprocessing as follows:

Definition 2.3 (Preprocessing)
Preprocessing is to find and implement command sequence $\bigodot_{k=1}^{n} c_k$ *that satisfies*

$$F' = \bigodot_{k=1}^{n} c_k(F)$$

where desired preprocessed file is $f_t \in F'$.

3. Preprocessing Using Metadata

Here is how we perform preprocessing using metadata. Data mining handles enormous and complex data, which makes comprehensive understanding very difficult. This naturally leads to processing using metadata. For example, we simply perform data sampling, define processing sequence for the sample data, and apply the obtained sequence to actual data. This can be considered as preprocess planning using sample data as metadata. What separates this from simple data visualization is to process metadata instead of actual data and then to process actual data only after processing sequence is obtained.

3.1 Formulation of Preprocessing Using Metadata

We define terms and formulate preprocessing using definition described in section 2.

Definition 3.1 (Metadata)
Data describing files are called metadata and shown as $d \in D$ *while function to extract metadata from files is* $d = meta(f)$ *and the one for a set of file is* $D = meta(F)$.

Definition 3.2 (Operator)
Operation for metadata is called operator and shown as o.

Accordingly operation to obtain D' using metadata set D can be described as

$$D' = o(D)$$

Then we have to consider relation between operation for metadata o and operation for actual data c. To reflect operation o on operation c, o has to be convertible to c.

The entity for transformation is defined as follows:

Definition 3.3 (Entity)
When $D = meta(F)$, $D' = meta(F')$, $D' = o(D)$, *entity is a function to generate* c *that satisfies*

$$F' = c(F)$$

and $c = entity(o)$

All these conclude as follows[☆]:

Theorem 3.4
When $D = meta(F)$, $D_n = meta(F_n)$ *and both*

$$D_n = o_n(o_{n-1}(...(o_1(D))...)) = \bigodot_{k=1}^{n} o_k(D)$$

and

$$\forall ic_i = entity(o_i)$$

are satisfied. Then,

$$F_n = c_n(c_{n-1}(...(c_1(F))...)) = \bigodot_{k=1}^{n} c_k(F)$$

is also satisfied.

The theory 3.4 shows that when the existing dataset is F, the metadata is $D = meta(F)$, the desired data is $f_t \in F_t$ and the metadata is $d_t \in D_t$, by finding $\bigodot_{k=1}^{n} o_k$ that satisfies

$$D_t = \bigodot_{k=1}^{n} o_k(D)$$

We can find $\bigodot_{k=1}^{n} c_k$ that satisfies

$$F'_t = \bigodot_{k=1}^{n} c_k(F) \wedge meta(F'_t) = D_t$$

We used F'_t to distinguish from desired F_t because when metadata processing is the same o and o', same metadata may sometimes be generated from different actual data. However when metadata extracted from data features are the same, the actual data are likely to be identical, which assures high possibility to generate command sequence for desired data by

[☆] We omit proving steps but it can be explained with mathematical induction for n.

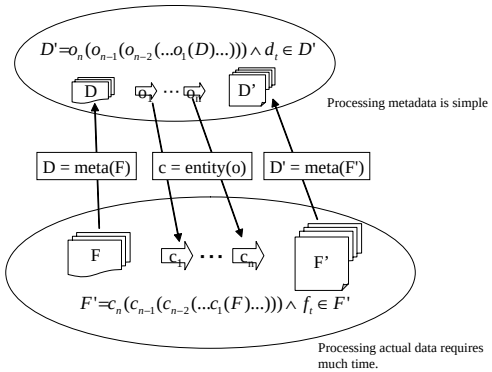

Fig. 1 Preprocessing Using Metadata

manipulating $\bigodot_{k=1}^{n} c_k$

4. Preprocessing Planning

4.1 Planning System

Planning system, one of the fields researched from the start of AI, automatically generates plan to transform initial state to goal state using the following three as input and output:

- Initial state : existing environmental model
- Goal state : aimed environmental model
- Operator : rule to convert environmental model

Algorithms developed and improved to generate various plans called "planner" have been actively done and many tools were released. The input language consisting of very simple structures with limited expression was not considered feasible for actual application. But development of expressive language using data type and quantifier recently opened the door for new application attempts.

4.2 Idea for Automatization

We try to realize automatic preprocessing by implementing planning system when metadata D is initial state, D' is goal state and $o \in O$ is operator set. If functioned as expected, planning system enables users to perform automatic planning by giving D and D', and to obtain $\bigodot_{k=1}^{n} o_k$ as a plan and $\bigodot_{k=1}^{n} c_k$ as a suggestion for actual data processing according to theorem 3.4.

However, this is just an idea with arising problems in implementation. We will explain how we tried to solve them.

4.3 Realization

4.3.1 Metadata

We investigate problems to solve in designing metadata. Planning system requires description of goal state. Accordingly metadata should include simple and obvious information as goal even before preprocessing. One of necessary information is header describing attribute values. Header can show necessary attributes or the candidates for analysis and possibly describe goal state.

Then we see how much data change can be observed using header. If we change attribute names whenever attribute values change, we can comprehend the state according to header change. But preparing metadata to observe record-related operation requires analysis in great detail. Metadata candidates are average and distribution for numerical attribute, or potential attribute value and the ratio for other attributes. The latter can be described somehow but it is impossible to describe numerical data before they are prepared, needless to say the description of goal state. So we leave record-related operation to operator and use information as metadata by which attribute is sorted.

4.3.2 Operator

If we can prepare all necessary operators applicable for all operations, simply describing initial state and goal state can generate plans, which is not possible nor realistic with countless operations used for preprocessing. Operator preparation is a bottleneck in realizing not only preprocessing but also planning system. Moreover, preprocessing may have to reflect users' intension. For example, system can not tell which existing attribute to use for new attribute. Or oversize operator set enlarges search space taking up too much time for planning and fails to prepare plan in the worst case. So we try to solve these problems as follows:

Dividing Operator Set

First operator set for automatically processing is prepared as O_{basic}. This operator set with user-defined operator set O_{user} are used as operator set for planning system as:

$$O_{planner} = O_{basic} \cup O_{user}$$

Details for O_{basic} and O_{user} is as follow:

O_{basic} includes processing like: simply join attributes from other file using main key, delete attribute and sort record. Operator schema can realize these operators. Preparation of these operators in advance enables system to automatically join or delete at-

tributes even when input file has extra attributes or when files are divided, and user's load can be reduced.

O_{user} is operator that users describe. The following *metaop* is prepared and users describe actual command, from which necessary information for operator is extracted.

Definition 4.1 (metaop)
Function metaop to extract operator o for actual data from command c is defined as follows:

$$(o) = metaop(c)$$

With this, actual command can be obtained as follows:

$$metaop^{-1}(o) = entity(o) = c$$

Additional Information for Operator

Simple metadata like the one we use this time may generate many problems. To avoid these problems, we add the following information to operator.

Ignore When some operators have the same operation for metadata, they all satisfy aim. So system may choose the wrong operator for planning. In such case, we can add Ignore information to the operator for system to re-plan.

ToDo System ignores operator like record selection, which does not change metadata. Each operator adds literal (done operatorID) to goal state and notify the system that processing must be performed.

Precondition When users want system to perform record selection first and then tabulation after that while system can not distinguish the priority as they have the same results regardless of order. By assigning record selection operator ID in tabulation operator precondition, system is forced to perform record selection first.

5. Case Study

Having no system at hand for comparison, we applied the proposed method to real-world data in order to confirm the effectiveness or shortcoming.

5.1 Implementation

Based on GraphPlan as a planner, we used IPP[5] for type information and quantifier and MUSASHI[4] as command for final output, naturally describing MUSASHI command for input.

MID	testdate	testname	testresult
1	19810219	GOT	55H
1	19810219	GPT	65H
1	19810219	TTT	1.7
1	19810320	GOT	57H
1	19810320	GPT	64H
1	19810320	TTT	1.6

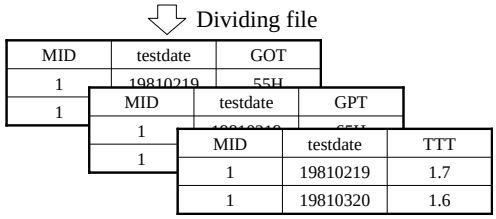

Fig. 2 Dividing file

5.2 Used Data

We used medical data (common data) for hepatitis after transforming the Japanese attribute name into English for implementation convenience. Cross tabulation, which is likely to be employed, requires shell script as well as MUSASHI command but can not be performed with the system☆. So we first divide file as shown in Fig. 2 and kept base-data file with MID, testdate and number of test in labobase.xt.

5.3 Case 1

We join test results : GOT, GPT, TTT, ZTT in base file labo-base.xt for data-cleaning First operator set $O_{cleaning}$ for cleaning was prepared as in Table 1.

`xtsed` is a command that eliminates unnecessary letters at the end of attribute values. Then other operator set O_{step1} necessary for data was prepared as in Table 2

`xtagg` tabulates attributes assigned by `-f` using attribute assigned by `-k` as key, and `xtdelnul` deletes all the lines whose attribute is Null. We put `yes` to ToDo and `delnul1` to Precondition so that `xtagg` will be performed after `xtdelnul`.

We performed planning using conditions as in Table 3, with results as in Fig. 3.

This sequence can be used in the most cases. Even if reference file does not have data, com-

☆ possible if we prepare macro command as one operator but we do not this time

Table 1 Operator Set for Cleaning

ID	Command	Precondition	ToDo
sed1	`xtsed -f GPT:GPTclean -c 'H#$\|H$' -v ''`		
sed2	`xtsed -f GOT:GOTclean -c 'H#$\|H$' -v ''`		
sed3	`xtsed -f TTT:TTTclean -c 'H$' -v ''`		
sed4	`xtsed -f ZTT:ZTTclean -c 'H$\|L$' -v ''`		

Table 2 Operator Set of Case 1

ID	Command	Precondition	ToDo
delnul1	`xtdelnul -f GOTclean,GPTclean,TTTclean,ZTTclean`		yes
agg1	`xtagg -k MID,testdate,testnumber -f GOTclean,GPTclean,TTTclean,ZTTclean -c sum`	delnul1	yes

Table 3 Input for Case 1

Input file	labo-base.xt
Related file	GOT.xt,GPT.xt,TTT.xt, ZTT.xt
Output file	case1.xt
Output attribute	MID,testdate,GOTclean, GPTclean,TTTclean,ZTTclean
Operator set	$O_{user} = O_{cleaning} \cup O_{step1}$

```
xtjoin -k MID,testdate,testnumber
       -f TTT -m ./attribute/TTT.xt
       -i labo-base.xt |
xtjoin -k MID,testdate,testnumber
       -f ZTT -m ./attribute/ZTT.xt |
xtjoin -k MID,testdate,testnumber
       -f GPT -m ./attribute/GPT.xt |
xtjoin -k MID,testdate,testnumber
       -f GOT -m ./attribute/GOT.xt |
xtsed -f GOT:GOTclean -c 'H#$|H$' -v '' |
xtsed -f GPT:GPTclean -c 'H#$|H$' -v '' |
xtsed -f ZTT:ZTTclean -c 'H$|L$' -v '' |
xtsed -f TTT:TTTclean -c 'H$' -v '' |
xtdelnul -f GOTclean,GPTclean,ZTTclean,\
                TTTclean -F |
xtagg -k MID,testdate,testnumber
      -f GOTclean,GPTclean,TTTclean,\
      ZTTclean -c sum |
xtcut -r -f testnumber -o case1.xt
```

Fig. 3 Output Command Sequence (Case 1)

mand sequence will be completed when we manually add `-n` option which does not delete base file record. This is because system can not judge options since `xtjoin` was automatically generated by system.

5.4 Case 2

Suppose data crossed.xt was given whose attributes were already joined using other input data. Crossed.xt includes unnecessary attributes like ALP, I-BIL, UA. Then we performed another planning with different input and reference file as in Table 4 to see how system performs planning according to structure change with results as in Fig. 4.

Table 4 Input for Case 2

Input file	crossed.xt
Related file	none
Output file	case2.xt
Output attribute	MID,testdate,GOTclean, GPTclean,TTTclean,ZTTclean
Operator set	$O_{user} = O_{cleaning} \cup O_{step1}$

```
xtsed -f GOT:GOTclean -c 'H#$|H$'
      -v '' -i clossed.xt |
xtsed -f GPT:GPTclean -c 'H#$|H$' -v '' |
xtsed -f ZTT:ZTTclean -c 'H$|L$' -v '' |
xtsed -f TTT:TTTclean -c 'H$' -v '' |
xtcut -r -f UA |
xtcut -r -f I-BIL |
xtcut -r -f ALP |
xtdelnul -f GOTclean,GPTclean,ZTTclean,\
         TTTclean -F |
xtagg -k MID,testdate,testnumber
      -f GOTclean,GPTclean,ZTTclean,\
      TTTclean -c sum |
xtcut -r -f testnumber -o case2.xt
```

Fig. 4 Output Command Sequence (Case 2)

The results show that the system was automatically adjusted with the input data change by excluding unnecessary `xtjoin` command and instead including `xtcut -f` command that deletes unnecessary attribute.

5.5 Case 3

When we prepare operator set $O_{cleaning}$ used for cleaning or operator set to discretize data, both data and medical knowledge are required beforehand. So when these data are prepared by users with the background and others can easily reuse the data, this means the knowledge can be reused. We considered the following case to see data reusability.

We prepared data having MID, testdate and GPTclean as attributes to observe GPT change by making a graph, which requires GPT cleaning and tabulation.

$O_{gptagg} = \{$`xtagg -k MID,testdate,testnumber`

-f GPTclean -c sum, ToDo=yes}

We have not done tabulation up to now so we use the following for input as in Table 5 with output command as in Fig. 5

Table 5 Input for Case 3

Input file	GPT.xt
Related file	None
Output file	case3.xt
Output attribute	MID,testdate,GPTcode
Operator set	$O_{user} = O_{cleaning} \cup O_{gptagg}$

```
xtsed -f GPT:GPTclean -c 'H#$|H$' -v ''
      -i GPT.xt |
xtagg -k MID,testdate,testnumber
      -f GPTclean -c sum |
xtcut -r -f testnumber -o case3.xt
```

Fig. 5 Output Command Sequence (Case 3)

The results show that only the necessary data were selected and applied in adequate order even though $O_{cleaning}$ had some unnecessary operators, which proved this system's capability to generate plans. Yet in case some operators for similar operation are included, it sometimes generate wrong plan. We can then perform replanning by adding Ignore information to wrong operators.

For more effective reuse of these operators, we need some schemes like dividing operator sets according to function, or realizing automatic preparation of operator set from history.

6. Conclusion

We explained an idea to automatize preprocessing using planning system techniques. Then we made research and experiment to realize the idea. Command sequence was generated semiautomatically with some room for improvement. The results confirmed the system's capability for effective adjustment to input data change and for knowledge reuse.

Acknowledgments

The authors thank Mayumi Miki for reviewing the manuscript for English. This research was supported by the Active Mining Project (Grant-in-Aid for Scientific Research on Priority Areas, No.759).

References

1) Peter Cabena and Pablo Hadjinian. "Discovering Data Mining", Prentice Hall PTR,1998.
2) Kempei Igarashi, Yoshihiro Ohta, Shigeki Yokoyama and Masayuki Numao. "Structural Data Transformation in Datamining using XML (in Japanese)", The 15th Annual Conference of Japanese Society for Artificial Intelligence, 2001.
3) Yukichi Yamada, Ryutaro Ichise, Masayuki Numao. "An Effective Pre-processing Model Using Layered Structure", JSAI Technical Report, SIG-A2-KBS60/FAI52-J, pp. 75–80, 2003.
4) MUSASHI project. "MUSASHI:Mining Utilities and System Architecture for Scalable processing of HIstorical data", http://musashi.sourceforge.jp/
5) Michael Brenner, Jana Koehler and Joerg Hoffmann. "IPP", http://www.informatik.uni-freiburg.de/~koehler/ipp.html