Representing Knowledge in the Statistical System Jasp

Ikunori Kobayashi[1] , Junji Nakano[2] , Yoshikazu Yamamoto[1] and Takeshi Fujiwara[3]

[1] Tokushima Bunri University, 1314-1 Shido, Kagawa 769-2193, JAPAN
[2] The Institute of Statistical Mathematics, 4-6-7 Minami-Azabu, Tokyo 106-8569, JAPAN
[3] The Graduate University for Advanced Studies, 4-6-7 Minami-Azabu, Tokyo 106-8569, JAPAN

Summary

In this article, we describe a framework to assist the process of analyzing data with knowledge in the statistical system Jasp. Although newly developed statistical systems have many statistical methods are sophisticated enough, most of users have more or less difficulties to master them and are in danger of swallowing results from a system without thinking deeply. The mechanism to assist users in Jasp has been implemented for preventing these problems. Knowledge used in this mechanism is represented as rules of "condition - action" form in the class definition. Jasp with built-in knowledge for statistical analysis can give advices to users, or can notify problems when they appear. Such abilities are useful for users, especially for students and novices of statistics or Jasp. We designed the form for representing knowledge so simple that users also can use Jasp as an environment to implement new statistical knowledge by themselves.

Keywords: Jasp, Object-Oriented Language, Rule, Statistical knowledge,

1 Introduction

Many statistical systems have been developed from the dawn of the computer, and have continued to adopt the new computer technology of each age. Famous examples are, to name a few, SAS (SAS Institute Inc. 2001), S (Chambers 1998) and XploRe (Härdle, Klinke & Müller 1999). Recently, the dissemination of cheap and powerful personal computers and the Internet offer new possibilities to data analysis environments. For example, we are able to browse many kinds of data easily on the Web, to stock huge data and to handle them even on our laptop computers. In order to use such modern technologies in statistical analyses effectively, several statistical systems are newly designed using recent technologies such as the Java language and the distributed computing. Our statistical system named Jasp (JAva based Statistical Processor) (Nakano, Fujiwara, Yamamoto & Kobayashi 2000) is one of them.

It is indisputable that statistical systems are useful for analyzing data. However, even if we have a very efficient statistical system, it does not ensure that we have meaningful results form the statistical analysis using it automatically. Here, we consider two main reasons for this fact. First, systems are not able to evaluate the calculation results such as many kinds of statistics, figures and tables. This work should be done by users considering the object of statistical analysis. Second, if we implement many functions in a system, the system becomes complex and difficult to operate. This is clearly shown by the fact that most textbooks of statistical systems are very thick. As too many procedures are available for particular data, it is not easy to choose suitable procedures from candidates for people with less experience of statistical analysis, even if he or she studied statistical analysis from some textbooks.

We need to solve these problems for making statistical systems more useful. As evaluation of the obtained results depends on the object of the data analysis, it is almost impossible to realize functions for it in a general purpose statistical system, even if we use techniques of new AI (Russell & Norvig 1995) research results. However, it may be possible to implement a function to inform basic mistakes of using statistical procedures, and suggest next possible procedures to try to solve them. Second, it is preferable to show a list of suitable instructions or commands to users. It is necessary to select them carefully according to the state and the history of the analysis, because the list of all applicable functions is too long in recent general purpose statistical systems. For example, when a user obtains a regression model, a list of instructions to calculate statistics for diagnosing a regression model is better

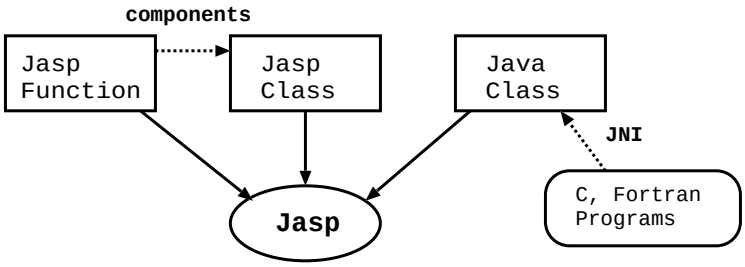

Figure 1: Jasp programming

than a much longer list for all the regression procedures.

Considering these requirements, we try to implement an ability to assist users using statistical knowledge stored in Jasp. Knowledge representation in the system should be simple, because users want to understand easily what kinds of knowledge have been built in the system and how they are used in analysis. We decide to describe statistical knowledges in the similar form as procedures or functions. In Jasp, a knowledge is expressed as a rule which is composed of a condition part and an action part. The rule is embedded in a class definition, and the scope is limited in it. A rule can have a priority to avoid conflict when several conditions of rules in the same class become true simultaneously.

2 Implementing Knowledge in Jasp

Statistical systems are required to be able to express various computation procedures easily and clearly, to draw graphics flexibly, and to customize functions for routine tasks. For realizing these purposes, most statistical systems use their programming languages as interfaces between systems and users. Jasp also has a programming language, i.e., the Jasp language. The Jasp language is based on the Pnuts language (Tomatsu 2000), which is a script language written in and for the Java language. We modified Pnuts for statistical users to be able to describe matrix handling and basic statistical computations simply, and to get graphical results easily. Jasp has three ways of programming, which are shown in Figure 1, and their purposes are explained in Table 1 (Kobayashi, Fujiwara, Yamamoto & Nakano 2001).

We implement rules in the Jasp class. The Jasp class is a module in which related Jasp functions are stored to use them at the same time. This implementation helps us to avoid needless conflicts with several rules.

Table 1: Purposes of Jasp programming

concept	purpose
Jasp Function	Description of a procedure of calculations
Jasp Class	Bundling related Jasp functions according to a meaningful statistical technique
Java Class	System extension

We have many kinds of rules for statistical objects. Conditions for data range are simple but useful rules; values of human height must be between 0 and 3 meters. We also have rules for timing of using methods; diagnostic procedures must follow the model building procedures. Significance levels are important rules for evaluating statistics; 5% is used as a significance level in biological sciences, but is too large in production engineering. Some of them are widely available for many data sets, but others are useful in limited field. We think most of such rules can be described in a simple general form. If the form is complex, users have difficulties to express their knowledge in the format.

A Jasp class is defined by the form

```
jaspclass NAME( SUPER ) BODY
```

where `NAME` is the name of the class, `SUPER` is the name of the super class. If there is no super class, `(SUPER)` can be omitted. `BODY` consists of constructors, methods, private functions and rules. Private functions are Jasp functions which are effective only in the class, and methods are used as interfaces of private functions to the outside of the class. Constructors are methods of the same name as its class name, and generate instances of the class.

Rules are defined in a Jasp class on the similar position as methods in the form

```
rule NAME( PRIORITY ) { CONDITION }{ ACTION }
```

where `NAME` is a name of the rule, `PRIORITY` is a real number which lies between 0 and 1. `CONDITION` shows the condition for the rule, and `ACTION` is the procedure which should be executed when the condition becomes true. The condition part and the action part are expressed by Jasp functions. In the condition part, a Jasp function which is called with no argument, returns a logical value (true or false) and does not change any field value is allowed. The action part can contain Jasp functions which are called with no argument, returns no value and can change some values of fields.

For checking rules, we send "`checkRules()`" message to an instance of the Jasp class. We assume that there is the Jasp class named `LinearRegression`

in Jasp, and the class has some rules about linear regression. We have to make an instance of the class by invoking the constructor first, and send "`checkRules()`".

```
> lr_test = LinearRegression("test.dat")
> lr_test.checkRules()
  [check_t_value(0.7), check_multicolinearity(0.5)]
```

Jasp returns names of rules with number of the local priority in the form of a list, according to priorities. Each condition of the rule in this list is evaluated as true.

To invoke the action part of the rule of the first element in the list, "`invokeFirst()`" message can be used.

```
> lr_test.invokeFirst()
```

Then, the method to calculate the t-statistics will be called. Each rule can be invoked by using "`invokeRule()`" message with the name of a rule:

```
> lr_test.invokeRule("check_t_value")
```

The "`invokeRule()`" message also can accept integer as the argument:

```
> lr_test.invokeRule(2)
```

To execute all rules in the list, "`invokeAll()`" message is available.

The inheritance mechanism in object oriented programming is applied to the rules. Then, a derived class can refer the rules in the upper class as well as methods and fields. We can mount particular rules with stability of Jasp by using this mechanism.

3 Examples

Figure 2 shows a part of the program for linear regression analysis in the Jasp class named `LinearRegression`. This program focuses on calculating the VIF (Variance Inflation Factor) and to check the multicolinearity by it. There are two method, a private function and a rule in this class. "`...`" means omission. The methods behave constructors, because their names are the same as this class. The constructor is invoked when the instance is created. The variables prefixed "`this.`" are treated as fields in this class. The private function named `vif` is for calculating the VIF with three arguments (a dependent variable, independent variables and a total sum of squares).

The rule named `check_multicolinearity` is for checking the multicolinearity using the VIF if it has never been checked. The priority of this rule is set to 0.5. In this action part, the VIF of each independent variable is compared to 10. If this is larger than 10, independent variables have multicolinearity (Ryan 1997), and Jasp notifies it to the user.

4 Conclusion

If knowledge of statisticians is stored and can be used effectively, statistical systems become much more convenient for naive users. Therefore, we propose how to represent and use statistical knowledge in the general purpose statistical system Jasp. Knowledge in Jasp is described by a simple "condition - action" form as a rule, and is stored in Jasp classes. Jasp shows applicable procedures when users require some assistance. This ability will be useful for users of Jasp, when they are at a loss what to do next. Jasp assists user's analysis by proposing next procedures. However, Jasp can not have any responsibility for the analysis, because objects of analysis are different for each user and are not stored in Jasp as general knowledge. Such knowledge can be implemented by users for their particular purposes.

At present, this ability to assist user's analysis is available only on the CUI (Character User Interface) window. We plan to improve Jasp to use rules on the GUI (Graphical User Interface) windows.

References

Chambers, J. M. (1998), *Programming with data: a guide to the S language*, Springer.

Härdle, W., Klinke, S. & Müller, M. (1999), *XploRe – Learning Guide*, Springer. (`http://www.xplore-stat.de/`)

Kobayashi, I., Fujiwara, T., Yamamoto, Y. & Nakano, J. (2001), The Language and the Extendibility of the Statistical System Jasp. In: *Proc. in ISM Symposium - Statistical software in the Internet age* -, 65–73.

Kobayashi, I., Fujiwara, T., Yamamoto, Y. & Nakano, J. (2001), Extendibilities of a Java based statistical system. In: *Proc. in International Conference on New Trends in Computational Statistics with Biomedical Applications*, 109–115.

Nakano, J., Fujiwara, T., Yamamoto, Y. & Kobayashi, I. (2000), A statistical package based on Pnuts. In: *COMPSTAT2000 Proceedings in Computational Statistics*, 361–366. Heidelberg: Physica-Verlag.

```
jaspclass LinearRegression {
  method LinearRegression( file ){
    ...
    this.LinearRegression( y, x )
  }
  method LinearRegression( y, x ){
    this.depVar = y ; this.indepVar = x
    this.coef = coef( y, x ) // coefficient
    this.y_hat = forecast( x, this.coef )
    this.st = cal_st( y )   // the total sum of squares
    ...
    this.vif = vif( y, x, this.st )
    this.checked_vif = false  // flag
  }
  function vif( y, x, st ){
    row = x.nr    // number of row
    col = x.nc    // number of column
    ans = Vector(col-1)
    for(i=2; i<=col; i++){
      new_x = x[,!i]     // delete i-th column
      coe = coef( y, new_x )
      tmp = y - new_x*coe
      tmp = tmp.trans * tmp
      mm = sqrt((st - tmp) / st)
      ans[i-1,1] = mm
    }
    ans.setColLabel(["VIF"])
    return ans
  }
  rule check_multicolinearity(0.5) {
    this.checked_vif == false
  }{
    str = ""
    size = this.vif.nr
    var = this.indepVar.getColLabel()
    for(i=1; i<=size; i++){
      if(this.vif[i,1] >= 10){
        str = str + var[i] +" "
      }
    }
    this.checked_vif = true
    str = str + " multicolinearity.\n"
    str = str + "You should check your model!\n"
    str
  }
  ...
}
```

Figure 2: A program for checking multicolinearity in a Jasp class

Ryan, T. P. (1997), *Modern Regression Methods*, John Wiley & Sons.

Russell, S. & Norvig, P.: (1995), *Artificial Intelligence*, Prentice Hall.

SAS Institute Inc. (2001), Statistical Analysis System,
`http://www.sas.com/`

Tomatsu, T. (2001), Pnuts, `http://javacenter.sun.co.jp/pnuts/`