Recommended Package Specification for Methods in openVA

Richard Li

March 18, 2016

1 Overview

This document provides a series of recommendations for package structures in order for easier integration into the **openVA** package. The example functions and data could be found in

```
# install.packages("openVA")
library(openVA)
```

2 Data input and main function

When implementing the model fitting function, we recommand the default data input to follow the rules below:

- The first column of data input is fixed to be ID.
- The symptoms are coded into three categories. "Y" for presense, "" for absence, and "." for missing. For methods not distinguishing missing and absence of symptom, we still highly recommand using this coding scheme for user input, but internally modify "." into "".
- For methods that require training data, it should be able to allow an additional column of cause-of-death in the input, and does not give an error or mistreat that column as a symptom.

For example, a model fitting function could look like the follows:

```
foo <- function(symps.train, symps.test, causes.train, causes.table = NULL,
                arg1, arg2, ...){

}
```

where the arguments are respectively:

- `symps.train` and `symps.test` are training and testing data respectively as specified before
- `causes.train` is either a character vector with the same length of the dimension of `symps.train`, or the column name of the cause-of-death column in training data.
- `causes.table`, is the list of causes-of-death to learn. If set to NULL, it will be internally calculated as all unique causes-of-death appeared in the training data. Notice, it could be only a subset of the causes-of-death in the training data too. In that case, it means the algorithm will throw away the causes not in the list and not consider them for the testing data.
- `arg1`, `arg2`, etc., are additional parameters required by the algorithm.
- `...` at the end is **required**, so as to be able to pass unused arguments from the wrapper to the function and not cause problems.

For examples of the dataset input, the `RandomVA3` dataset in **openVA** could be an example. As for a quick test, the following code should be able to run:

```r
library(openVA)
data(RandomVA3)
train <- RandomVA3[1:100, ]
test <- RandomVA3[101:200, ]

fit <- foo(symps.train   = train,
           symps.test    = test,
           causes.train  = "cause",
           causes.table  = NULL)
```

As an example of the description above, the following code snippet could also be useful as the first few lines of function:

```r
# if input cause is the column name
if(class(causes.train) == "character" && length(causes.train) == 1){
    colindex <- match(causes.train, colnames(symps.train))
    colindex2 <- match(causes.train, colnames(symps.test))

    if(is.na(colindex)){
      stop("Cannot find the cause-of-death column in training data")
    }

    # re-define causes.train as a vector
    causes.train <- symps.train[, colindex]
    # and remove it from the symptoms
    symps.train <- symps.train[, -colindex]

    # also remove this from testing data if it is provided
    if(!is.na(colindex2)){
        causes.test <- symps.test[, colindex2]
        symps.test <- symps.test[, -colindex2]
    }
}

# re-define causes.table is NULL
if(is.null(causes.table)){
  causes.table <- unique(causes.train)
}

# extract ID
id.train <- symps.train[, 1]
symps.train <- symps.train[, -1]
id.test <- symps.test[, 1]
symps.test <- symps.test[, -1]

# make sure train and test has the same columns
joint <- intersect(colnames(symps.train), colnames(symps.test))
symps.test <- symps.test[, joint]
symps.train <- symps.train[, joint]
```

3 Output and S3 methods

It will be best if the returned object from the function has a S3 class defined. For example,

```
foo <- function(symps.train, symps.test, causes.train, causes.table = NULL,
                arg1, arg2, ...){

  ...

  out <- list(...)
  class(out) <- "someName"
  return(out)
}
```

In terms of what to output, it will be best to output as much information as possible. For example, some output to be considered may include:

- `ID`, the IDs of the testing data should be part of the output no matter what, so that identification of each data point is not through row index only. But it could be part of the other output variables (for example, as row name of another output) as well.
- `prob` or `score`, basically this would be a N by C matrix specifying the likelihood or score of each cause-of-death for each death.
- `CSMF`, this could be also be omitted from output, but calculated from other functions.

4 Summary method

The summary method will consist of two parts. First is `summary.someName()` function, and also the `print.someName_summary()` function. For example,

```
summary.someName <- function(object, top = 5, id = NULL, ...){
  out <- NULL
  ...

  if(!is.null(id)){
    # calculate Ordered COD distribution for the given ID
  }else{
        # calculate Ordered CSMF distribution for the population
  }

  out$top <- top
  out$id <- id
        class(out) <- "someName_summary"
        return(out)
}


print.someName_summary <- function(x, ...){
  # message to print to console
}
```

- `summary.someName()` should be able to calculate ordered CSMF distribution of ordered COD distribution for a given ID. It should also save the relevent model fitting information, such as the model parameters arg1, arg2, etc., so that user could in principle retrieve all model options from the summary object.

- print.someName_summary() prints relevant information and top CSMFs to the console. For example of such function, see print.interVA_summary in **InterVA4** package.

5 Default Plot

The default plot will be plotting the top k CSMF. If no error bars are calculated, it could be written in a similar fashion as below:

```
plot.someName <- function (x, top.plot = 5, min.prob = 0, ...) {
    # calculate CSMF here
    dist.cod <- ...

    if (!is.null(top.plot)) {
        if (top.plot < length(dist.cod)) {
            thre <- sort(dist.cod, decreasing = TRUE)[top.plot]
            min.prob <- max(min.prob, thre)
        }
    }
    dist.cod.min <- dist.cod[dist.cod >= min.prob]
    dist.cod.min <- sort(dist.cod.min, decreasing = FALSE)
    par(las = 2)
    par(mar = c(5, 15, 4, 2))
    bar.color <- grey.colors(length(dist.cod.min))
    bar.color <- rev(bar.color)
    barplot(dist.cod.min, horiz = TRUE, names.arg = names(dist.cod.min),
        col = bar.color, cex.names = 0.8, xlab = "Probability",
        ...)
}
```

We recommend the default plot function be named plot.someName, so that it is specific to the class of model fitting, but this is just a minor issue. It should at least has two options: top.plot for how many causes to plot and main for the title of the plot. min.prob used above defines the secondary filter of the smallest causes to plot and could be optional. Additional variables are also allowed to control for the customization of the plot

6 Function to calculate CSMF and COD assignment

These two functions are very useful for **codeVA** package.

- csmf.someName() should take the fitted object as input, and return a vector of length C, or a data frame of C rows if error bars of the CSMFs are calculated. The names of vector should be specified as the causes-of-death (as in causes.table when fitting the model), so that it is straightforward to understand. It is best to return the unsorted results, so that it is easier to compare across datasets.

- topCOD.someName() should return a N by 2 data frame, where the first column is ID and the second column is the cause-of-death assignment for each death.

- Note these functions in the existing packages (i.e., InterVA4, Tariff, and InSilicoVA) are implemented quite differnetly. For example, an insilico object calculates the CSMF in the summary function, while an interVA object in the CSMF function. We are hoping as we include more packages this will be more consistent.

7 Additional plots and functions

Additional plots and functions are definitely allowed!

8 Additional comments

One other consideration of making the package is trying to avoid function names defined in **openVA** package. For example, `codeVA`, `plotVA`, `stackplotVA`, `getCSMF`, `getTopCOD`, etc. If certain function name has to be the same as the ones used in **openVA**, the usage of that function will need additional effort to avoid confusion.

For example, if `getCSMF` needs to be defined for something else in the package **somePkg**, and it is also used in another function `getCSMF2`. Then it should be called like the following:

```r
getCSMF <- function(x, y, z){...}

getCSMF2 <- function(x, y, z){
  some_results <- somePkg::getCSMF(x, y, z)
  ...
}
```

Notice in the above example, if `somePkg::getCSMF` is replaced with only `getCSMF`, it will create error when **openVA** is loaded.