InSilicoVA package

Richard Li, Tyler McCormick, Sam Clark
Contact email: lizehang@uw.edu

November 18, 2015

Usage guidelines

This document contains instructions for installing and using the InSilicoVA R package. This package is experimental and we welcome your feedback. Please note that our model uses a Bayesian computation framework that may require tuning to fit specific datasets. The results will not be valid if the model is not fit correctly. Please contact us if you are not familiar with Bayesian modeling and, specifically, MCMC diagnostics. We'll be happy to help! Finally, please do not distribute this software. We are actively working on a fully documented, open-source, release. Until then, if you know someone else that would like to experiment with our model, please email us and we'll gladly distribute the latest version to them.

1 Install pre-requisites (R, Java, and rJava)

The InSilicoVA package requires the user's machine to have Java installed. The configuration steps could be tedious depending on the operating systems (Windows, OSX, or Linux). Here is a quick checklist for setting up the required environment for the InSilicoVA package. If the solutions provided here are not sufficient, please contact the author at the email above.

1. **Check R environment** (or install from `http://cran.r-project.org/` if R is not installed, or R version lower than 3.0). Open R and on the welcome message, there is a line starting with "Platform" and ending with "(32-bit)" or "(64-bit)". It is very important which version (32-bit or 64-bit) of R you use, since the Java JDK should have the same version. Note that sometimes multiple versions of R could be installed on the same machine, so you should check the version you wish to use for data analysis. For example, if you prefer using RStudio to run the codes, you should check the default R version of RStudio by reading the welcome message for RStudio.

2. **Check whether Java is installed** on your machine by opening terminal (or Command Prompt in Windows system), and typing in *"java -version"*. If Java is installed, it will show the Java version number. In general, version number higher than 1.6 should be sufficient.

3. **If no Java is installed, or version too low**, you should download and install a newer Java JDK by following the instructions at `http://www.oracle.com/technetwork/java/javase/downloads/index.html`. On that page, there are downloads for both "Java SE 8uXX" and "Java SE 7uXX" (XX is the specific most recent released subversion number), which stands for Standard Edition Java 8 and 7. At this point, the development and test of the package is performed using Java 7, but both should work. The required download is JDK (Java Development Kit). Click on download link for JDK and choose the appropriate version. You should choose "x86" version if your R version is 32-bit, and "x64" if R is 64-bit. Then follow the instructions to finish download and install Java.

4. After successfully installing Java, try again typing *"java -version"* on terminal or Command Prompt. It should show the correct version number just installed.

5. Install and load rJava package by typing in R

```
install.packages("rJava")
library(rJava)
```

6. If there are errors from installing rJava, a few possible tricks are (*Note: this is not a complete list of troubleshooting solutions, just a few tricks that worked in some systems. If you have trouble after trying these, please contact us at the email above.*):

   - Open terminal/Command Prompt and run *"sudo R CMD javareconf"*, then trying installing rJava again.
   - If rJava could be installed by not loaded, i.e., errors pop out at "library(rJava)", try typing in R the following code:

     ```
     if (Sys.getenv("JAVA_HOME")!="")
     Sys.setenv(JAVA_HOME="")
     library(rJava)
     ```

   - to-be-updated-with-more

7. In many systems, rJava package could be a pain to install. Feel free to contact me with any problems, or Google how to install rJava on windows/mac/linux/etc./

2 Install InSilicoVA package

Once rJava is successfully installed, the required environment for the InSilicoVA package is ready. Before InSilicoVA is released to CRAN, installing the package requires a few more steps:

- **Install dependencies manually** Since it is not on CRAN yet, dependent packages also need to be installed manually using:

  ```
  install.packages('coda')
  install.packages('ggplot2')
  ```

- **Download InSilicoVA source file and install manually** The source file is available at the following address: `https://github.com/richardli/InSilicoVA-beta/blob/master/InSilicoVA_1.0.tar.gz`. Click the "view raw" option and it will download the source file. After putting it in the working directory, the package could be installed using (change the file directory to where you store the file):

  ```
  install.packages("InSilicoVA_1.0.tar.gz", type = "source", repos = NULL)
  ```

- After the first two steps, now the package could be loaded and browsed by

  ```
  library(InSilicoVA)
  ```

3 InSilicoVA methodologies

The complete study is posted on `http://arxiv.org/abs/1411.3042`.

4 Loading data

Here a quick demo of InSilicoVA package is provided with external data source. The data input is exactly as the InterVA input, except the missing data are represented as ".". The data should consist of 246 columns with the first column being the ID of the death, as in the InterVA input. The first column should always be ID, but the rest of the symptom columns could change orders. But notice if the order is not the same as the original InterVA input, the column names must be correctly labeled. The data could also consist of additional columns that are not used, as long as their column names are not the same as the symptom columns.

Here is an example of reading data files and looking at some entries. In order to maintain compatibility for the dataset created for InterVA, either the empty cell or "NA" could be used to represent a "No" in data. Missing, on the other hand, is strictly restricted to ".".

```
data <- read.csv("data/random_interva4.csv")
## use only the first 2000 deaths to illustrate in this document
data <- data[1:100, ]
## showing what the data looks like
head(data[, 1:10])
```

ID	elder	midage	adult	child	under5	infant	neonate	male	female
1		Y		NA	NA	.	.	Y	
2	Y			NA	NA	.	.	Y	
3			Y	NA	NA	.	.	Y	
4			Y	NA	NA	.	.		Y
5	Y			NA	NA	.	.	Y	
6	Y			NA	NA	.	.	Y	

5 Running InSilicoVA model

The core algorithm is performed using the function below. For illustration purposes, we only run the model with 400 iterations before reaching convergence. **If you are not familiar with Bayesian modeling and MCMC convergence diagnostics, please email us and we will happily help adjust these codes as necessary for your data.** The command described below runs one MCMC chain. In practice, we recommend running multiple chains started from diverse initial values.

```
fit<- insilico( data, subpop = NULL,
  length.sim = 400, thin = 10, burnin = 200,
  auto.length = FALSE, conv.csmf = 0.02,
  useProbbase = FALSE, keepProbbase.level = TRUE,
  datacheck = TRUE, warning.write = FALSE,
  external.sep = TRUE,
  java_option = "-Xmx1g", seed = 1)

## Performing data consistency check...
## Data check finished.
## Not all causes with CSMF > 0.02 are convergent.
##  Please check using csmf.diag() for more information.
```

The above codes could be run with minimum setting of `data`, `subpop`, `length.sim`, `burnin`, and using default setting for the rest of the parameters. For more flexible setup, a few key parameters are listed here:

- `length.sim`, `burnin`, `thin`
  The length of chain is decided by these three factors. The common practice is to run the MCMC chain for K iterations and discard the first half of the chain, which means setting `length.sim = K` and `burnin = K/2`. The *thin* variables specifies the frequency of saving the chain. For instance, $thin = 10$ means all variables gets saved every 10 iterations. It is useful especially when the computer's memory is limited. In practice, a chain of length $5,000$ should be a reasonable starting value for most cases.

- `auto.length`, `conv.csmf`
  If the length of the chain is not clear ahead of time, there is the option of setting `auto.length` to be TRUE, which will keep doubling the length of the chain automatically (at most twice to avoid too heavy RAM usage) if the top causes are not convergent after the initial iterations. In practice, the rare causes will be very difficult to converge by conventional statistical test, but as long as the their fractions are very little, say smaller than 2% in the population, the influence is minimum. Thus by setting `conv.csmf = 0.02` will tell the algorithm only to check causes with fractions higher than 0.02.

- `useProbbase`, `keepProbbase.level`
  One of the key features of InSilicoVA is that the conditional probabilities of symptoms given causes ($\Pr(S|C)$) do not rely entirely on expert knowledge (as in InterVA-4), but are re-estimated from the data. If this re-estimation is not desired, setting `useProbbase = TRUE` tells the algorithm not to perform re-estimating. If re-estimation is performed, `keepProbbase.level` specifies that re-estimation is only performed on the $\Pr(S|C)$ table instead of the whole matrix. In general, estimating $\Pr(S|C)$ only through the table that consists of 15 levels (i.e., "I", "A+", "A", etc.) is more robust. More details are presented in the original paper.

Several other parameters for controlling the algorithm implementation for easier use in practice:

- `datacheck`, `warning.write`
  In InterVA, if the user enters deaths with impossible symptom combinations, e.g., male with pregnant symptoms, the algorithm will correct the input following a set of rules. This step is the same as in InSilicoVA if `datacheck` is set to be TRUE, and the messages will be printed out in a txt file if warning.write is set to TRUE.

- `external.sep`
  For external causes such as traffic accident, accidental fall, suicide, etc., deciding a death to be caused by them is relatively easy and separated from other general health symptoms. Setting this indicator to TRUE will tell the algorithm to estimate those causes separately, which is usually more robust, with the trade-off of not estimating confidence intervals for these causes.

- `seed`
  Setting Seed is usually good practice. Since the algorithm is stochastic, there could be some randomness in the outcome. But it will produce the same outcome with the same seed in each machine. It could be set to any numerical numbers.

Several more advanced fine-tuning parameters controlling the sampler are listed below. It is suggested to check the original paper first for better understanding of the influence of them. In practice, keeping the default values for these parameters should be sufficient.

- `jump.scale`
  The scale of Metropolis proposal in the Normal model. Default is set to be 0.1.

- `levels.prior`, `levels.strength`
  `levels.prior` is a vector of prior expectation of conditional probability levels. They do not have to be scaled. The algorithm internally calibrate the scale to the working scale through `levels.strength`. If NULL the algorithm will use InterVA-4 table values as prior. `levels.strength` is the scaling factor for the strength of prior beliefs in the conditional probability levels. Larger value constrain the posterior estimates to be closer to prior expectation. Default value 1 scales `levels.prior` to a suggested scale that works empirically.

- `trunc.min`, `trunc.max`
  Minimum and maximum possible value for estimated conditional probability table. Default to be 0.0001 and 0.9999

After fitting the model, you could see some summary of the fitted results by simply calling

```
fit

## InSilicoVA fitted object:
## 1930 death processed
## 400 iterations performed, with first 200 iterations discarded
##  20 iterations saved after thinning
## Fitted with re-estimated InterVA4 conditional probability level table
```

Or more summary of CSMFs (top 10 causes are summarized here for demonstration) by

```
summary(fit, top = 10)

## InSilicoVA Call:
## 1930 death processed
## 400 iterations performed, with first 200 iterations discarded
##  20 iterations saved after thinning
## Fitted with re-estimated InterVA4 conditional probability level table
## Data consistency check performed as in InterVA4
##
## Top 10 CSMFs:
##                                     Mean Std.Error  Lower Median  Upper
## HIV/AIDS related death            0.3286    0.0178 0.2978 0.3272 0.3535
## Acute resp infect incl pneumonia  0.1509    0.0208 0.1113 0.1542 0.1743
## Other and unspecified infect dis  0.1176    0.0182 0.0849 0.1191 0.1429
## Pulmonary tuberculosis            0.1028    0.0124 0.0844 0.0995 0.1234
## Diabetes mellitus                 0.0550    0.0087 0.0430 0.0532 0.0729
## Other and unspecified neoplasms   0.0350    0.0234 0.0070 0.0279 0.0749
## Assault                           0.0264    0.0000 0.0264 0.0264 0.0264
## Other and unspecified external CoD 0.0259    0.0000 0.0259 0.0259 0.0259
## Road traffic accident             0.0171    0.0000 0.0171 0.0171 0.0171
## Other transport accident          0.0135    0.0000 0.0135 0.0135 0.0135
```

And CSMF plots for the top 10 causes (you could change how many causes to show by setting `top`) could be made with the following line of codes. The error bars in the CSMF plot indicate the 95% credible interval.

```
plot(fit, top = 10)
```

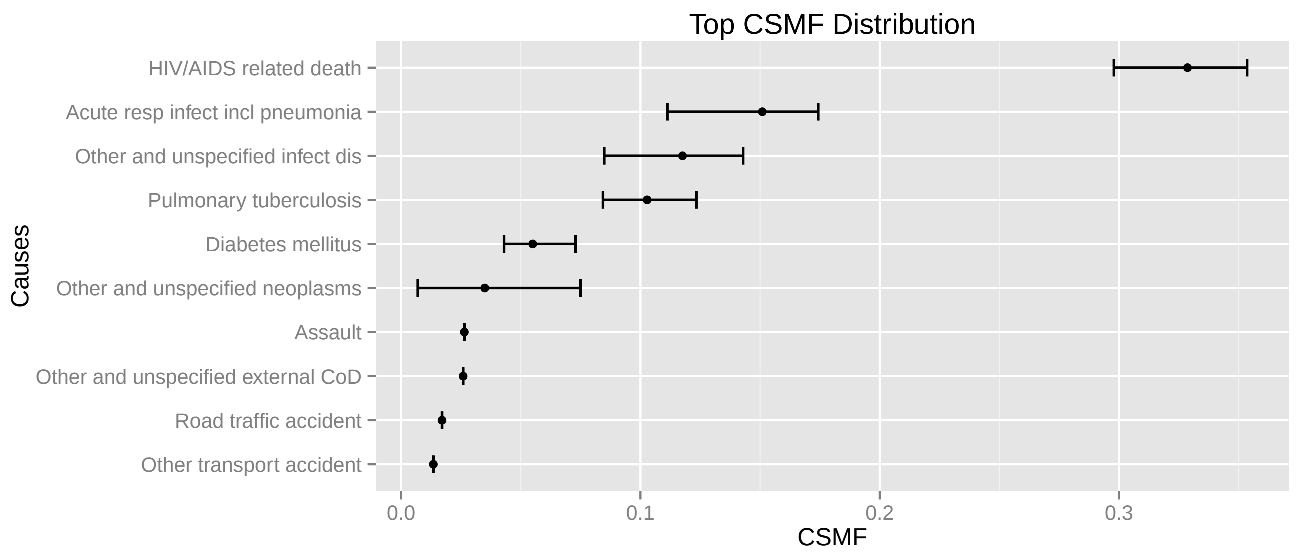


Figure 1: Top 10 CSMF

And it also have a bar plot version (plotted in black and white style)

```
plot(fit, type = "bar", top = 10, bw = TRUE)
```

All CSMF's at each iterations, a matrix where each row represents one sample of CSMF's, could be extracted by

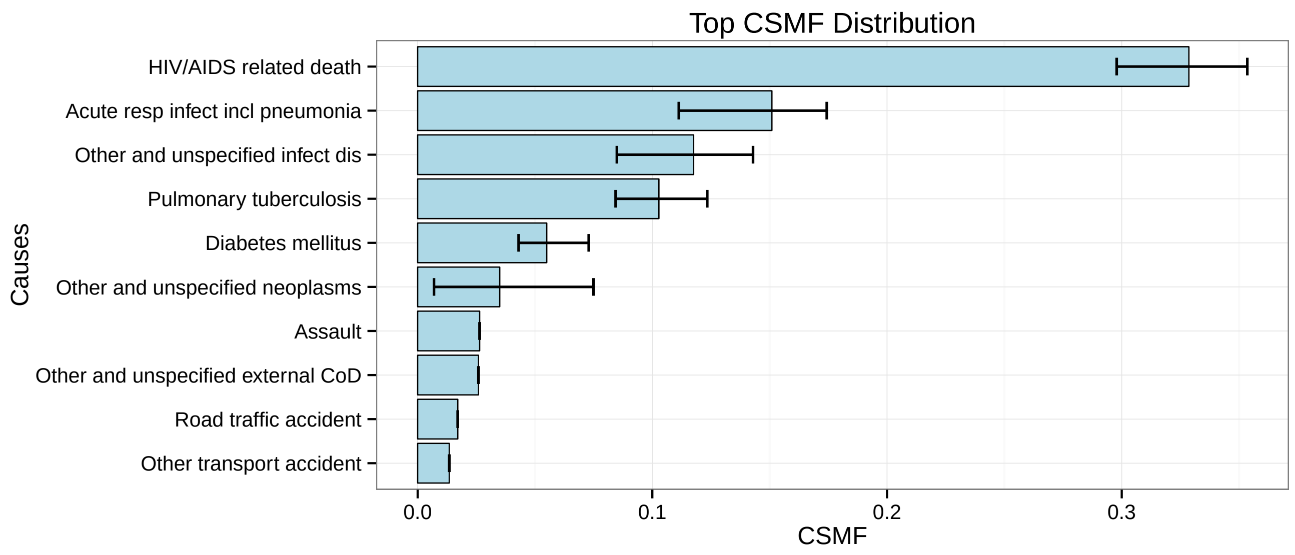


Figure 2: Top 10 CSMF

```
csmf.each.iteraions <- fit$csmf
```

Direct analysis could be performed using CSMF's at each iteration, but some simple summary statistics could also be directly obtained using the summary methods for a InSilicoVA fit object. The lower and upper value in the CSMF summary are the 95% credible interval.

```
# summarizing a fitted model
sf <- summary(fit)
csmf.mean <- sf$csmf
head(csmf.mean)
```

	Mean	Std.Error	Lower	Median	Upper
Sepsis (non-obstetric)	0.0001826	0.0000799	0.0001086	0.0001409	0.0003375
Acute resp infect incl pneumonia	0.1509368	0.0207586	0.1112712	0.1541998	0.1743077
HIV/AIDS related death	0.3286146	0.0178320	0.2978467	0.3271540	0.3535100
Diarrhoeal diseases	0.0045538	0.0029442	0.0011198	0.0040100	0.0101707
Malaria	0.0006057	0.0002592	0.0002570	0.0005418	0.0010628
Measles	0.0034510	0.0035028	0.0007114	0.0014787	0.0110974

The same summary statistics could be ordered by mean CSMF too using:

```
# summarizing a fitted model
csmf.mean.ordered <- sf$csmf.ordered
head(csmf.mean.ordered)
```

	Mean	Std.Error	Lower	Median	Upper
HIV/AIDS related death	0.3286146	0.0178320	0.2978467	0.3271540	0.3535100
Acute resp infect incl pneumonia	0.1509368	0.0207586	0.1112712	0.1541998	0.1743077
Other and unspecified infect dis	0.1175847	0.0182350	0.0848809	0.1191085	0.1429026
Pulmonary tuberculosis	0.1027992	0.0124187	0.0843518	0.0994663	0.1233993
Diabetes mellitus	0.0550057	0.0087265	0.0430144	0.0532392	0.0728888
Other and unspecified neoplasms	0.0349776	0.0234305	0.0069576	0.0278872	0.0749339

6 Individual-specific analysis

In this section, a simple analysis on individual Cause of Death is provided. The individual COD distribution could be obtained by the following codes:

```
indiv.prob <- fit$indiv.prob
indiv.id <- fit$id
```

The `indiv.prob` now contains the matrix of individual COD distribution, i.e., each column represent a cause, and each row a death. The values in each row add up to 1 or 0 (no symptoms, or age/sex missing if data check is turned on). The `indiv.id` contains the death IDs of the corresponding rows.

```
check_prob_sum <- apply(indiv.prob, 1, sum)
table(check_prob_sum)
```

0	1
198	1732

Or the above could be exported to CSV file for other analysis easily using

```
write.csv(summary(fit)$indiv, file = "insilico_indiv_prob.csv", row.names=FALSE)
```

The top cause for each death could then be examined. Just for illustrating purpose, here we include another cause "Undetermined" for deaths without symptoms, failed data check, or the largest cause has probability less than 0.4 (the arbitrary cut-off following InterVA-4 convention).

```
# find the max probability for each death
max.prob <- apply(indiv.prob, 1, max)
# find which deaths have max probability < 0.4 (including 0)
undeterm <- which(max.prob < 0.4)
# for all deaths find which cause is the maximum
which.max.cause <- apply(indiv.prob, 1, which.max)
# replace the undertmined death with value NA
which.max.cause[undeterm] <- NA

# replace numeric cause by the name
# cause names
names <- colnames(indiv.prob)
# save the names to a new vector
max.cause <- names[which.max.cause]
# rename NA to "undetermined"
max.cause[which(is.na(max.cause))] <- "Undetermined"
# get a table summary
cause.table <- table(max.cause)
# reorder the table in deceasing by frequency
cause.table <- cause.table[order(cause.table, decreasing = TRUE)]
# move undetermined to last
where_underterm_is <- which(names(cause.table) == "Undetermined")
cause.table <- c(cause.table[-where_underterm_is], cause.table[where_underterm_is])
cause.table

##             HIV/AIDS related death  Acute resp infect incl pneumonia
##                                633                               261
##  Other and unspecified infect dis           Pulmonary tuberculosis
##                                260                               206
##                  Diabetes mellitus   Other and unspecified neoplasms
##                                116                               113
## Other and unspecified cardiac dis              Severe malnutrition
##                          28                             28
##                      Asthma               Obstructed labour
##                           5                              4
##                      Stroke                  Acute abdomen
##                           4                              3
##             Liver cirrhosis                  Renal failure
##                           3                              3
##   Reproductive neoplasms MF                 Severe anaemia
##                           3                              3
##                    Epilepsy          Obstetric haemorrhage
##                           2                              2
##       Respiratory neoplasms          Acute cardiac disease
##                           2                              1
##         Digestive neoplasms   Pregnancy-induced hypertension
##                           1                              1
##                Undetermined
##                         248
```

This is only presenting a simple analysis of tabulate top CODs (with a arbitrary decision on "Undetermined"), many analyses could be carried out as well.

7 Convergence diagnostics

See the example in the package help file for `csmf.diag`.

8 Physician de-bias

See the example in the package help file for `physician_debias`.

9 Advanced features

9.1 Sub-population specific CSMFs

We could also specify sub-populations. This could be helpful if you wish to, for example, run a model that separately estimates CSMFs for multiple regions within the same country or if you want to estimate CSMFs separately for different groups in the population. In the example below, we assume some members of the population have known HIV status and others do not. We perform the analysis to generate separate CSMFs for each status. We are estimating different CSMF within each sub-population group, but allowing them to share information from the jointly estimated conditional probability matrix. A subpopulation in the code could be just a vector of length N (the same length as the data). It could be either strings or numeric values. The names of the sub-populations will maintain through the model. For example, we constructed this fake subpopulation with the first 1000 death assigned to "Group1" and the next 1000 assigned to "Group2":

```r
N <- dim(data)[1]
subpop <- c(rep("Group1", round(N/2)), rep("Group2", N-round(N/2)))
```

Another way to specify sub-population is by telling the algorithm which columns contain the information. For example,

```r
data(RandomVA2)
# see the added column named "subpop"
head(RandomVA2[, 1:5])
# You don't need to know more details of the function call at this point,
# but only the way subpopulaiton is specified...
fit2<- insilico(RandomVA2, subpop = "subpop",
               length.sim = 4000, burnin = 2000, thin = 10, auto.length = FALSE)
# Or if you want to use two factors in data to specify sub-population,
# say, we create these two fake HIV indicators,
# and you want to use the combination of the two as sub-population
HIV_indicator <- RandomVA2$subpop
HIV_indicator2 <- sample(RandomVA2$subpop)
data_more <- cbind(data, HIV_indicator, HIV_indicator2)
# again, just check out how to specify two levels of sub-population
fit3<- insilico( data_more, subpop = list("HIV_indicator", "HIV_indicator2"),
               length.sim = 4000, burnin = 2000, thin = 10 , auto.length = FALSE)
```

9.2 Run with Subpopulations

The InSilicoVA package could fit the model with multiple sub-populations, for example, deaths with different HIV status. Using the randomly generated sub-population in previous section, the model fitting syntax is mostly the same as before, except specifying `subpop`.

```r
fit2 <- insilico( data, subpop = subpop,
  length.sim = 400, thin = 10, burnin = 200,
  auto.length = FALSE, conv.csmf = 0.02,
  useProbbase = FALSE, keepProbbase.level = TRUE,
  datacheck = TRUE, warning.write = FALSE,
  external.sep = TRUE,
  java_option = "-Xmx1g", seed = 1)

## Performing data consistency check...
## Data check finished.
## Not all causes with CSMF > 0.02 are convergent.
##  Please check using csmf.diag() for more information.
```

And the summary methods will separate out two sub-populations. For the space limit in this document, only top 3 causes are shown.

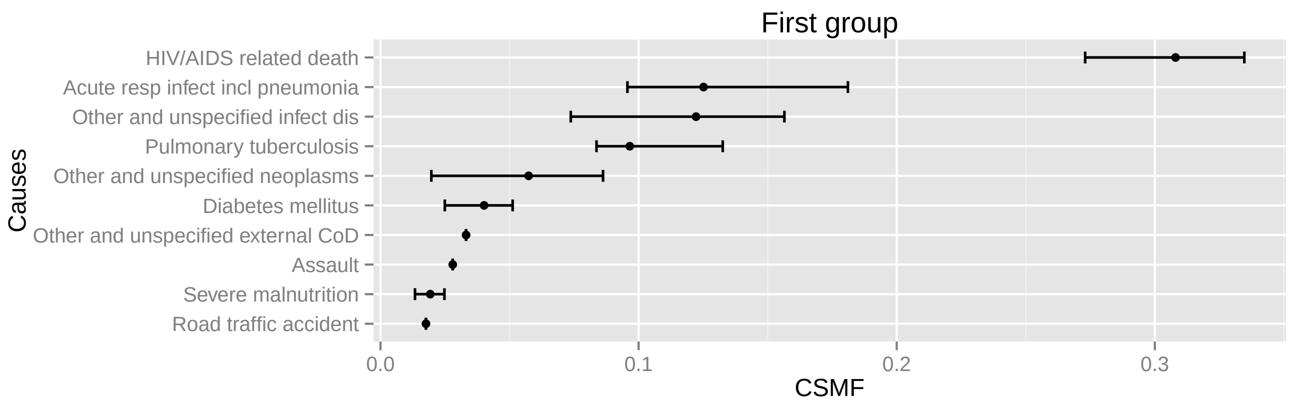
```
summary(fit2, top = 3)

## InSilicoVA Call:
## 1930 death processed
## 400 iterations performed, with first 200 iterations discarded
##  20 iterations saved after thinning
## Fitted with re-estimated InterVA4 conditional probability level table
## Data consistency check performed as in InterVA4
## Sub population frequencies:
## Group1 Group2
##    857    875
##
## Group1 - Top 3 CSMFs:
##                                    Mean Std.Error  Lower Median  Upper
## HIV/AIDS related death           0.3080    0.0169 0.2730 0.3113 0.3347
## Acute resp infect incl pneumonia 0.1251    0.0234 0.0957 0.1231 0.1811
## Other and unspecified infect dis 0.1222    0.0240 0.0737 0.1289 0.1565
##
## Group2 - Top 3 CSMFs:
##                                    Mean Std.Error  Lower Median  Upper
## HIV/AIDS related death           0.3558    0.0284 0.3192 0.3527 0.4006
## Acute resp infect incl pneumonia 0.1348    0.0124 0.1069 0.1384 0.1503
## Pulmonary tuberculosis           0.1006    0.0126 0.0844 0.0944 0.1225
```

For each sub-population, the same CSMF graph could be generated as before.

```
# plot Group1
plot(fit2, which.sub = "Group1", title = "First group")
# plot Group2
plot(fit2, which.sub = "Group2", title = "Second group")
```

With multiple sub-population, the CSMF graph could also be plotted for comparing the differences. The following line of codes plots the top 5 largest CSMFs (note there are 6 causes in total, since the top 5 causes differ in the two sub-populations) in the two sub-populations in one figure.

```
# plot Comparing top 5 causes in the two groups
plot(fit2, type = "compare", top = 5)
```



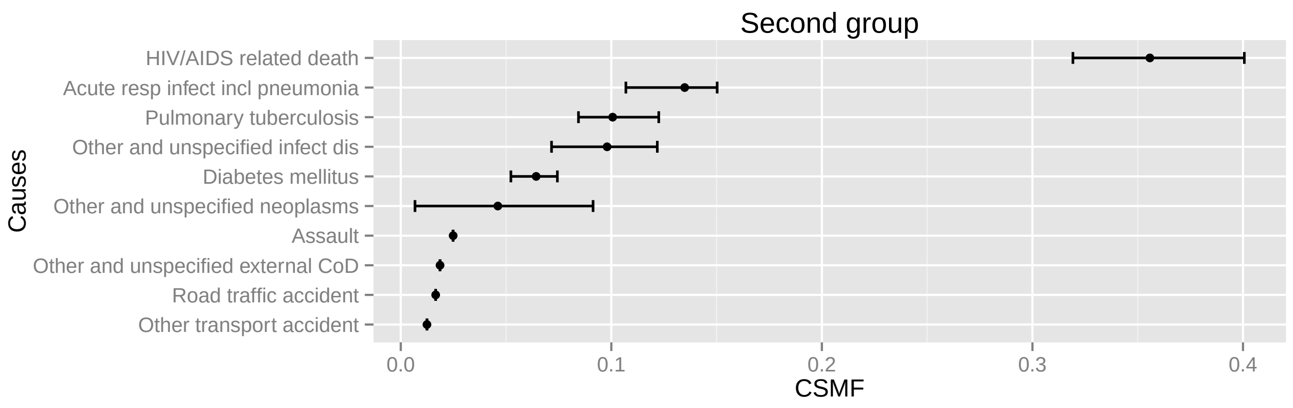


Figure 3: Top 10 CSMF in Group 1 and 2

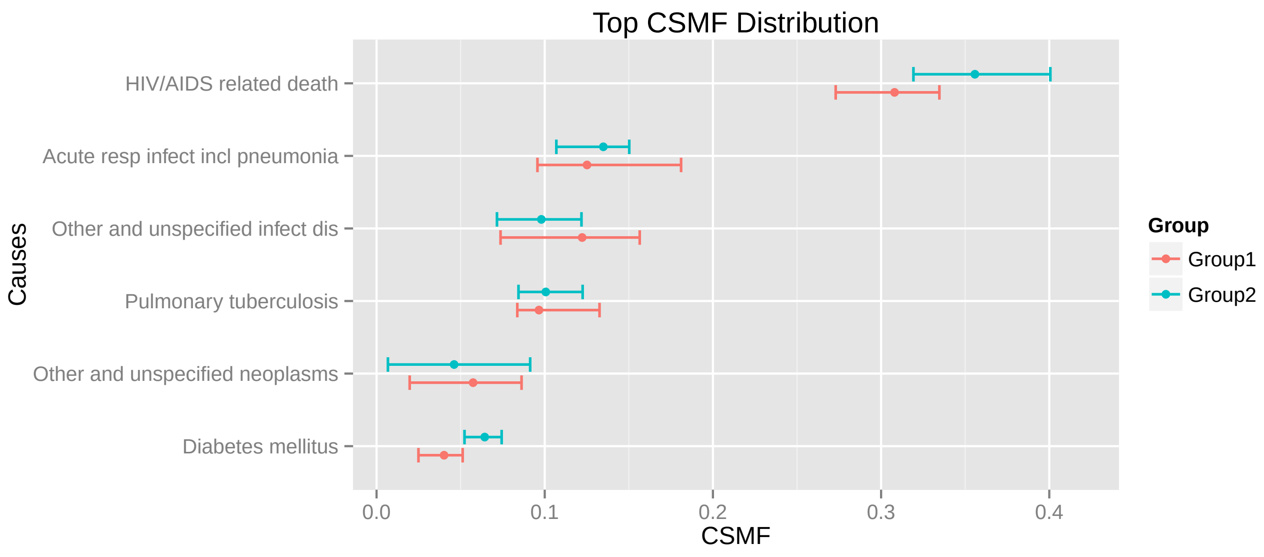


Figure 4: Top 10 CSMF comparison