Model VA Algorithm Code

Sam Clark

November 9, 2018

Introduction

This document presents R code

- to simulate VA deaths,
- to create SCI from reference deaths with known cause,
- that implements each of the four algorithms - Naive Bayes Classifier, InterVA, InSilicoVA, and Tariff
- to calculate concordance matrices of assigned causes
- to calculate and compare CSMF Accuracies
- to conduct simulated cross-validation studies of VA algorithm performance, holding VA deaths and SCI constant, and
- to conduct a heuristic investigation of the effects of SCI on VA algorithm performance.

R functions are defined for most of the activities above. These can be reused easily and loaded all at once from the file 'Model-VA-Functions.R' using the R command 'source(file="Model-VA-Functions.R").

```
rm(list = ls())
source(file = "./Model-VA-Functions.R")
ls()
```

```
##  [1] "calcConcordance"    "calcCSMF"
##  [3] "calcCSMFaccuracies" "calcInSilicoVA"
##  [5] "calcInterVA"        "calcNBC"
##  [7] "calcSCI"            "calcTariff"
##  [9] "crossValidate"      "expit"
## [11] "logit"              "plotInSilicoVA"
## [13] "simulateDeaths"     "storeSCI"
## [15] "storeVADeaths"
```

All of the code developed and presented below is for **heuristic** demonstration and pedagogical use *only*. The code replicates only the *core logic* of each algorithm without important additional funcationality that is present in all real-world software implementations of the algorithms.

VA Data

VA data consist of deaths (rows) with a set of binary symptoms (columns). Reference deaths are VA deaths that also have a *true* cause. Reference deaths are used to prepare SCI or assess the accuracy of causes assigned by algorithms (or any other mechanism).

Symptom-Cause Information (SCI)

Symptom-Cause Information is any representation of the *true* relationship between VA symptoms and causes of death. SCI can be prepared from refernece deaths, or it can come from another source. When SCI is prepared from reference deaths it captures the true (or observed) relationship between VA symptoms and causes in the dataset of reference deaths, so it is limited by whatever information exists about that relationship in that specific set of reference deaths. Alternativelt SCI can be compiled from other sources of information that describe how symptoms relate to causes. Not all forms of SCI are perfect and they all have limitations based on how thye were derived. Many of the algorithms use SCI that do not account for dependence among symptoms for deaths from a given cause.

VA Cause-Coding Algorithms

VA cause-coding algorithms are effectively logical devices that combine VA data collected from deaths with SCI to assign causes of death that are jointly consistent with the symptoms in the collected VAs and the particular SCI used. VA algorithms are often a combination of logic and computation and sometimes contain statistical procedures as well, e.g. InSilicoVA. Algorithms vary in the inputs they require and the outputs they produce, and some algorithms also include significant data checking/editing functions that are run before the algorithm. The code presented below implements the core logic of each algorithm without any of additional data checking/edition or other extra funcationality included in all real-world versions of the algorithms.

Symptom-Cause Information

Create a function that converts an expandable list of SCI into a neat, usable matrix of SCI values.

```
###### <<Function>>
storeSCI <- function(SCIlist) {

    # converts a list of p(s|c) SCI vectors into a labeled matrix

    # SCIlist: a list of 5-element SCI vectors, each element
    # betweten 0.0 and 1.0 return a labeled cause x p(s|c) matrix

    # create matrix from list
    SCI.mat <- matrix(unlist(SCIlist), ncol = 5, byrow = TRUE)
    # label columns
    colnames(SCI.mat) <- c("s1", "s2", "s3", "s4", "s5")
    # label rows initialize row labels vector
    sci.rownames <- "cause1"
    # loop through causes
    for (i in 2:nrow(SCI.mat)) {
        # append each cause to row labels vector
        sci.rownames <- c(sci.rownames, paste("cause", i, sep = ""))
    }
    rownames(SCI.mat) <- sci.rownames

    # return matrix of SCI
    return(SCI.mat)

}
```

Create an matrix of $\Pr(s|c)$ SCI values.

```
# create de novo SCI consisting of conditional probabilities
# p(s|c): causes x 5 symptoms you can add or subtract causes,
# i.e. rows below
SCI.denovo <- list()
SCI.denovo[[1]] <- c(0.9, 0.4, 0.1, 0.01, 0.8)
SCI.denovo[[2]] <- c(0.3, 0.41, 0.8, 0.75, 0.9)
SCI.denovo[[3]] <- c(0.1, 0.01, 0.01, 0.95, 0.8)
SCI.denovo[[4]] <- c(0.05, 0.9, 0.75, 0.5, 0.2)
# have a look at the list of SCI
SCI.denovo
```

```
## [[1]]
## [1] 0.90 0.40 0.10 0.01 0.80
##
```

```
## [[2]]
## [1] 0.30 0.41 0.80 0.75 0.90
##
## [[3]]
## [1] 0.10 0.01 0.01 0.95 0.80
##
## [[4]]
## [1] 0.05 0.90 0.75 0.50 0.20
```

```
# store the SCI into a matrix
SCI.denovo.mat <- storeSCI(SCI.denovo)
# have a look at the matrix of SCI
SCI.denovo.mat
```

```
##          s1   s2   s3   s4  s5
## cause1 0.90 0.40 0.10 0.01 0.8
## cause2 0.30 0.41 0.80 0.75 0.9
## cause3 0.10 0.01 0.01 0.95 0.8
## cause4 0.05 0.90 0.75 0.50 0.2
```

Simulate Deaths

Create a function to simulate some deaths from a matrix of SCI. For each death, first draw a cause, and then for each symptom, draw a binary value corresponding to whether or not the symptom appears for that death. The causes are drawn according to a set of CSMFs provided as a pameter, and the symptoms are drawn according to the matrix of SCI provided as a parameter.

```
###### <<Function>>
simulateDeaths <- function(simDeaths.num, csmf, psc.mat) {

    # simulates a set of VA deaths using p(s|c) SCI and CSMFs
    # provided

    # simDeaths.num is the number of deaths to simulate csmf is a
    # vector of fractions, one for each cause psc.mat is matrix
    # of conditional probabilities p(s|c) return * simDeaths: a
    # labeled deaths x cause,symptoms matrix of simulated deaths

    # initialize vector simulated deaaths matrix
    simDeaths <- c()
    # loop over deaths
    for (i in 1:simDeaths.num) {
        # draw a cause, 1 or 2
        cause <- which(rmultinom(1, 1, csmf) == 1)
        # select he correct set of conditional probabilities for
        # drawing symptoms
        psc <- psc.mat[cause, ]
        # draw symptoms conditional on this cause initialize symptoms
        # vector
        symptoms <- c()
        # loop over symptoms
        for (s in 1:5) {
            symptoms <- c(symptoms, rbinom(1, 1, psc[s]))
        }
        # add this death to the deaths matrix
```

```
        simDeaths <- rbind(simDeaths, c(cause, symptoms))
    }
    # name the columns initialize column names vector
    death.colnames <- c("cause", "s1")
    # add symptom labels
    for (s in 2:5) {
        death.colnames <- c(death.colnames, paste("s", s, sep = ""))
    }
    colnames(simDeaths) <- death.colnames
    # initialize row names vector
    death.rownames <- "death1"
    # add row labels
    for (i in 2:nrow(simDeaths)) {
        death.rownames <- c(death.rownames, paste("death", i,
            sep = ""))
    }
    rownames(simDeaths) <- death.rownames

    # return simulated deaths
    return(simDeaths)

}
```

Simulate some deaths from the SCI matrix we created earlier. First create a set of CSMFs for the four causes that we defined in the SCI.

```
# fraction of deaths that are for each cause: cause1, cause2,
# etc.
csmf <- c(0.25, 0.45, 0.2, 0.1)
sum(csmf)  # check that the elements of csmf add to 1.0
```

```
## [1] 1
```

```
# simulate 1000 deaths with the numbers of each determined by
# csmf and the symptoms generated by the conditional
# probabilities in psc.mat
simDeaths <- simulateDeaths(1000, csmf, SCI.denovo.mat)
# have a look
head(simDeaths)
```

```
##        cause s1 s2 s3 s4 s5
## death1     2  0  0  1  1  1
## death2     4  0  1  0  1  1
## death3     1  1  1  0  0  1
## death4     3  0  0  0  1  1
## death5     1  1  0  1  0  1
## death6     4  0  1  1  1  0
```

```
# have a look at the dimensions
dim(simDeaths)
```

```
## [1] 1000    6
```

```
# count the deaths by cause
table(simDeaths[, 1])
```

```
##
```

```
##   1   2   3   4
## 252 454 185 109
```

```
# CSMF of the simulated deaths
table(simDeaths[, 1])/1000
```

```
##
##     1     2     3     4
## 0.252 0.454 0.185 0.109
```

```
# for deaths of cause 1, count the those with/without symptom
# 1
num <- table(simDeaths[which(simDeaths[, 1] == 1), 2])[2]
den <- sum(table(simDeaths[which(simDeaths[, 1] == 1), 2]))
num/den  # P(s1 | c1): yes it matches
```

```
##         1
## 0.8492063
```

```
# for deaths of cause 1, count the those with/without symptom
# 2
table(simDeaths[which(simDeaths[, 1] == 1), 3])
```

```
##
##   0   1
## 163  89
```

```
# for deaths of cause 1, count the those with/without symptom
# 2
table(simDeaths[which(simDeaths[, 1] == 1), 4])
```

```
##
##   0   1
## 223  29
```

```
# for deaths of cause 1, count the those with/without symptom
# 2
table(simDeaths[which(simDeaths[, 1] == 1), 5])
```

```
##
##   0   1
## 251   1
```

```
# for deaths of cause 1, count the those with/without symptom
# 2
table(simDeaths[which(simDeaths[, 1] == 1), 6])
```

```
##
##   0   1
##  51 201
```

```
# for deaths of cause 2, count the those with/without symptom
# 1
table(simDeaths[which(simDeaths[, 1] == 2), 2])
```

```
##
##   0   1
## 322 132
```

Sample Simulated Deaths for Train/Test Split

Select a sample of simulated deaths. The sample will be used to create SCI, and the rest of the deaths will be used as test deaths to investigate the performance of the algorithms.

```
# set the fraction of deaths to sample
samp.frac <- 0.5
# draw the sample
sample <- sample(nrow(simDeaths), samp.frac * nrow(simDeaths))
# have a look at the first few elements of the sample: these
# are the row numbers of the sampled deaths
head(sample)
```

```
## [1] 385 869 791 186 410 196
```

```
# create a dataset of the sampol
SCIdeaths <- simDeaths[sample, ]
# have a look at the new set of deaths
head(SCIdeaths)
```

```
##          cause s1 s2 s3 s4 s5
## death385     2  0  0  1  1  1
## death869     1  0  0  0  0  0
## death791     2  0  1  1  1  1
## death186     2  0  0  1  1  1
## death410     1  0  1  0  0  1
## death196     1  1  1  0  0  1
```

```
# make sure the number corresponds to the sample fraction
dim(SCIdeaths)
```

```
## [1] 500   6
```

```
# complement of the sample is for testing
testDeaths <- simDeaths[(-sample), ]
head(testDeaths)
```

```
##        cause s1 s2 s3 s4 s5
## death1     2  0  0  1  1  1
## death2     4  0  1  0  1  1
## death4     3  0  0  0  1  1
## death6     4  0  1  1  1  0
## death8     1  1  1  0  0  1
## death9     1  1  1  0  0  1
```

```
dim(testDeaths)
```

```
## [1] 500   6
```

```
# make sure the SCI and test deaths are completely
# complementary sets match tests if any element of v1 is
# found in v2 and returns an 'NA' if not all tests if *all*
# the elements of the result of is.na() are 'NA' expect a
# single valued respons of 'TRUE'
all(is.na(match(rownames(SCIdeaths), rownames(testDeaths))))
```

```
## [1] TRUE
```

Calculate CSMFs

Create a function to calculate the CSMFs corresponding to a set of deaths.

```
###### <<Function>>
calcCSMF <- function(deaths) {

    # calculate CSMFs for a set of deaths

    # deaths is a deaths x cause,symptoms matrix of deaths retuns
    # a causes-element vector of CSMFs

    # identify unique causes
    causes <- sort(unique(deaths[, 1]))
    # initialize vector of CSMFs
    csmfs <- c()
    # loop over causes
    for (c in 1:length(causes)) {
        # calculate the fraction of deaths with cause c and add it to
        # the matrix of CSMFs
        csmfs <- c(csmfs, nrow(deaths[which(deaths[, 1] == c),
            ])/nrow(deaths))
    }
    # make csmfs result into a matrix
    csmfs <- matrix(csmfs, nrow = 1)
    # label columns
    csmfs.colnames <- "cause1"
    for (i in 2:length(causes)) {
        csmfs.colnames <- c(csmfs.colnames, paste("cause", i,
            sep = ""))
    }
    colnames(csmfs) <- csmfs.colnames
    # label rows - subsitute() returns the name of its input
    rownames(csmfs) <- c(substitute(deaths))

    # return CSMFs
    return(csmfs)

}
```

Calculate the CSMFs of the test deaths and be sure that they are similar to the CSMFs provided when they were simulated.

```
testDeaths.CSMF <- calcCSMF(testDeaths)
# have a look and ensure that they correspond to the csmf
# used to simulate the deaths
testDeaths.CSMF
```

```
##            cause1 cause2 cause3 cause4
## testDeaths  0.236  0.474  0.176  0.114
```

```
# csmf used to simulate the deaths
csmf
```

```
## [1] 0.25 0.45 0.20 0.10
```

The match is good.

Calculate SCI from Reference Deaths

Create a function to calculate SCI from a set of deaths with a reference cause. For all the deaths of a given cause, for each symptom, calculate the fraction of deaths for which the symptom exists.

```
###### <<Function>>
calcSCI <- function(deaths) {

    # calculate SCI from a set of deaths with causes and symptoms

    # deaths is a deaths x cause,symptoms matrix of deaths return
    # a cause x symptoms matrix of conditional probabilities
    # p(s|c)

    # identify unique causes
    causes <- sort(unique(deaths[, 1]))
    # initialize list of vectors of SCI
    sci <- list()
    # loop over causes
    for (c in 1:length(causes)) {
        # calculate the fraction of deaths with each symptom for this
        # cause add the result vector to the list of SCI
        sci[[c]] <- colSums(deaths[which(deaths[, 1] == c), 2:6])/nrow(deaths[which(deaths[,
            1] == c), ])
    }
    # convert the list to a matrix
    sci <- matrix(unlist(sci), ncol = 5, byrow = TRUE)
    colnames(sci) <- c("s1", "s2", "s3", "s4", "s5")
    # initialize row labels vector
    sci.rownames <- "cause1"
    # loop over causes and create row labels vector
    for (i in 2:length(causes)) {
        # append each cause to row labels vector
        sci.rownames <- c(sci.rownames, paste("cause", i, sep = ""))
    }
    rownames(sci) <- sci.rownames

    # return cause x symptoms matrix of SCI
    return(sci)

}
```

Calculate SCI from the the sample of SCI deaths, created above.

```
# calculate SCI from the datasset of SCI deaths
SCIdeaths.SCI <- calcSCI(SCIdeaths)
# have a look at the SCI
SCIdeaths.SCI
```

```
##                s1        s2        s3          s4        s5
## cause1 0.83582090 0.2835821 0.1268657 0.007462687 0.7686567
## cause2 0.30875576 0.3594470 0.7603687 0.755760369 0.8755760
## cause3 0.13402062 0.0000000 0.0000000 0.958762887 0.8041237
## cause4 0.05769231 0.8846154 0.7692308 0.461538462 0.3076923
```

```
# check to be sure that the calculated SCI is similar to the
# SCI used to simulate the deaths have a look at the
# calculated SCI rounded to 2 decimal places
round(SCIdeaths.SCI, 2)
```

```
##          s1   s2   s3   s4   s5
## cause1 0.84 0.28 0.13 0.01 0.77
## cause2 0.31 0.36 0.76 0.76 0.88
## cause3 0.13 0.00 0.00 0.96 0.80
## cause4 0.06 0.88 0.77 0.46 0.31
```

```
# have a look at the SCI used to simulate the deaths that
# were used to calculate the SCI
SCI.denovo.mat
```

```
##          s1   s2   s3   s4  s5
## cause1 0.90 0.40 0.10 0.01 0.8
## cause2 0.30 0.41 0.80 0.75 0.9
## cause3 0.10 0.01 0.01 0.95 0.8
## cause4 0.05 0.90 0.75 0.50 0.2
```

```
# calculate and display the differences between the two
round(SCIdeaths.SCI, 2) - SCI.denovo.mat
```

```
##           s1    s2    s3    s4    s5
## cause1 -0.06 -0.12  0.03  0.00 -0.03
## cause2  0.01 -0.05 -0.04  0.01 -0.02
## cause3  0.03 -0.01 -0.01  0.01  0.00
## cause4  0.01 -0.02  0.02 -0.04  0.11
```

Calculate the CSMFs of the SCI deaths.

```
# calculate CSMFs for SCIdeaths
SCIdeaths.CSMF <- calcCSMF(SCIdeaths)
SCIdeaths.CSMF
```

```
##           cause1 cause2 cause3 cause4
## SCIdeaths  0.268  0.434  0.194  0.104
```

```
sum(SCIdeaths.CSMF)
```

```
## [1] 1
```

```
csmf
```

```
## [1] 0.25 0.45 0.20 0.10
```

This looks good.

Create Dataset of User-defined (Arbitrary) Deaths

Create a function that converts an expandable list of VA deaths into a neat, usable matrix of VA death symptom values, similar to teh storeSCI function above.

```
###### <<Function>>
storeVADeaths <- function(deathList) {

    # converts a list of VA death vectors into a labeled matrix
```

```
    # deathList: a list of 5-element VA symptom vectors, each
    # element 0 or 1 return a labeled death x symptoms matrix

    # create a matrix with the deaths & signs/symptoms: rows for
    # deaths, columns for signs/symptoms
    deaths.mat <- matrix(unlist(deathList), ncol = 5, byrow = TRUE)
    # label columns
    colnames <- c("s1", "s2", "s3", "s4", "s5")
    colnames(deaths.mat) <- colnames
    # label rows initialize row labels vector
    death.rownames <- "death1"
    # loop over deaths
    for (i in 2:nrow(deaths.mat)) {
        # append each death top row labels vector
        death.rownames <- c(death.rownames, paste("death", i,
            sep = ""))
    }
    rownames(deaths.mat) <- death.rownames

    # return labeled matrix of ddeaths
    return(deaths.mat)

}
```

Create a matrix of deaths with symptoms from an example list of VA deaths. This list can be edited, expanded, or contracted to create an arbitrary set of VA deaths.

```
# create a list of n VA deaths with 5 binary signs/symptoms
# each
VAs <- list()

# make up 10 VA deaths, each with 5 symptoms
VAs[[1]] <- c(0, 1, 1, 1, 1)
VAs[[2]] <- c(1, 1, 0, 0, 0)
VAs[[3]] <- c(0, 1, 0, 0, 1)
VAs[[4]] <- c(1, 1, 1, 0, 1)
VAs[[5]] <- c(0, 1, 0, 1, 1)
VAs[[6]] <- c(1, 1, 0, 0, 1)
VAs[[7]] <- c(0, 1, 1, 0, 1)
VAs[[8]] <- c(1, 1, 1, 0, 1)
VAs[[9]] <- c(0, 0, 0, 0, 1)
VAs[[10]] <- c(0, 1, 0, 0, 0)

# Create a matrix of deaths x symptoms
VAexampleDeaths <- storeVADeaths(VAs)
# have a look at the VA deaths dataset
VAexampleDeaths
```

```
##         s1 s2 s3 s4 s5
## death1   0  1  1  1  1
## death2   1  1  0  0  0
## death3   0  1  0  0  1
## death4   1  1  1  0  1
## death5   0  1  0  1  1
## death6   1  1  0  0  1
```

```
## death7   0  1  1  0  1
## death8   1  1  1  0  1
## death9   0  0  0  0  1
## death10  0  1  0  0  0
```

Naive Bayes Classifier Algorithm

Create a function to implement the Naive Bayes Classifier (NBC) VA coding algorithm. Comments in the code explain each step.

```
###### <<Function>>
calcNBC <- function(SCI,SCI.c,deaths) {

  # implements the NBC VA algorithm

  # SCI is a cause x symptoms matrix of SCI conditional probabilities p(s|c)
  # SCI.c is p(c)
  # deaths is a deaths x symptoms matrix of VA deaths containing binary symptom values
  # returns a list containing
  #   * deaths x cause,(largest)cause-probability matrix of NBC-identified causes
  #   * deaths x cause-probabilities matrix of all cause-probabilities
  #   * CSMFs calculated by summing 'fractional deaths' for each cause,
  #   * CSMFs calculated by calculating the fraction of deaths assigned to each cause
  #   * when: date and time when finished
  #   * deaths: the input deaths
  #   * sci: the input SCI

  # NBC.probs are the NBC probabilities for each death: deaths x causes
  NBC.probs <- array(0,dim=c(nrow(deaths),nrow(SCI)))

  # NBC.causes are causes for maximum NBC probabilities
  NBC.causes <- data.frame(
    cause=as.character(rep("",nrow(deaths))),
    probability=rep(0,nrow(deaths)),
    stringsAsFactors = FALSE
  )
  rownames(NBC.causes) <- rownames(deaths)

  # calculate the NBC conditional probabilities of causes given symptoms vectors, p(c|s)
  # loop over deaths
  for (d in 1:nrow(deaths)) {
    # loop over causes and calculate NBC products
    # initialize a vector of probabilities for each cause
    prob.causes <- c()
    for (c in 1:nrow(SCI)) {
      # loop over symptoms and update NBC products
      # initialize the NBC product for this cause
      nbc.product <- 1
      # loop over symptoms
      for (s in 1:ncol(SCI)) {
        # update the NBC product for this symptom
        nbc.product <- nbc.product * SCI[c,s]^deaths[d,s] * (1-SCI[c,s])^(1-deaths[d,s])
      }
      # multiple NBC product by p(c) and add to the results by cause
```

```
    prob.causes <- c(prob.causes,SCI.c[c] * nbc.product)
  }
  # now that we have NBC products for each cause, loop over causes again and
  #   calculate NBC probability for each cause and this death's set of symptoms,
  #   e.g. normalize the NBC products
  # loop over causes
  for (c in 1:nrow(SCI)) {
    NBC.probs[d,c] <- prob.causes[c] / sum(prob.causes)
  }
  # for each death identify the cause with the largest probability
  # take the first element from the which() statement because sometimes all values
  #   of NBC.probs[d,] are equal
  NBC.causes[d,1] <- paste("cause",which(NBC.probs[d,]==max(NBC.probs[d,]))[1],sep="")
  NBC.causes[d,2] <- NBC.probs[d,which(NBC.probs[d,]==max(NBC.probs[d,]))[1]]
}

# store all NBC cause-specific probabilities for each death in a labeled matrix
NBC.summaries <- NBC.probs
rownames(NBC.summaries) <- rownames(deaths)
# initialize vector of column labels, cause-probabilities
causes.NBC <- c("cause1.probability")
# loop over causes
for (c in 2:nrow(SCI)) {
  # append each cause-probability to the column labels
  causes.NBC <- c(causes.NBC,paste("cause",c,".probability",sep=""))
}
colnames(NBC.summaries) <- causes.NBC

# calculate NBC CSMFs by summing 'fractional deaths', i.e. cause-probabilities for each cause
csmf.fracDeath <- colSums(NBC.summaries)/nrow(NBC.summaries)
# make csmf.fracDeath a matrix
csmf.fracDeath <- matrix(csmf.fracDeath,nrow=1)
# label columns with cause names
colnames(csmf.fracDeath) <- rownames(SCI)
# label rows - subsitute() returns the name of its input
rownames(csmf.fracDeath) <- c(paste(substitute(deaths),".fracDeath",sep=""))

# calcualte CSMF by summing up classified deaths for each cause
# initialize vector of CSMFs
csmf.topCause <- c()
# loop over causes
for (c in 1:nrow(SCI)) {
  # calculate and append fraction of deaths assigned to each cause
  csmf.topCause <- c(csmf.topCause,length(which(NBC.causes[,1]==paste("cause",c,sep="")))/nrow(deaths)
}
# make csmf.topCause a matrix
csmf.topCause <- matrix(csmf.topCause,nrow=1)
# label columns
colnames(csmf.topCause) <- rownames(SCI)
# label rows - subsitute() returns the name of its input
rownames(csmf.topCause) <- c(paste(substitute(deaths),".topCause",sep=""))

# create list of outputs to return
```

```
  ret.list = list(
    # deaths x cause,(largest)cause-probability matrix of NBC-identified causes
    top.cause = NBC.causes,
    # deaths x cause-probabilities matrix of all cause-probabilities
    cause.probabilities = NBC.summaries,
    # CSMFs calculated by summing 'fractional deaths' for each cause,
    #   i.e. cause-probabilities for each cause
    csmf.fracDeath = csmf.fracDeath,
    # CSMFs calculated by calculating the fraction of deaths assigned to each cause
    csmf.topCause = csmf.topCause,
    # datetime when saved
    when = format(Sys.time(),"%a %b %d %X %Y"),
    # deaths
    deaths = deaths,
    # SCI
    sci = SCI
  )

  # return list containing various outputs
  return(ret.list)

}
```

Use the NBC algorithm to identify causes for the test deaths created above, and then look at all the elements of the list of results returned by the *calcNBC()* function.

```
# prob.c is set of (marginal) p(c) required by the algorithm
# don't know what it is, so use a uniform distribution
prob.c <- rep(1/nrow(SCIdeaths.SCI), nrow(SCIdeaths.SCI))
# to use the correct marginals that were used to simulate the
# deaths, use SCI.c <- csmf verify that it's what you expect
prob.c
```

```
## [1] 0.25 0.25 0.25 0.25
```

```
# create a 'VAdeaths' dataset from the test deaths
VAdeaths <- testDeaths[, 2:6]  # test deaths

# assign causes to the test deaths using the NBC algorithm
NBC.results <- calcNBC(SCIdeaths.SCI, prob.c, VAdeaths)
```

Have a look at the causes assigned to each death.

```
head(NBC.results$top.cause)
```

```
##         cause probability
## death1 cause2   0.9490393
## death2 cause2   0.5884834
## death4 cause3   0.8997796
## death6 cause4   0.9201668
## death8 cause1   0.9538533
## death9 cause1   0.9538533
```

Make sure the result dataset has a row for each death.

```
dim(NBC.results$top.cause)
```

```
## [1] 500   2
```

Have a look at the probabilities calculated for each cause for each death.

```
head(NBC.results$cause.probabilities)
```

```
##        cause1.probability cause2.probability
## death1       3.646292e-04         0.94903925
## death2       3.482985e-03         0.58848343
## death4       7.939397e-04         0.09462430
## death6       4.579821e-05         0.07978741
## death8       9.538533e-01         0.03435839
## death9       9.538533e-01         0.03435839
##        cause3.probability cause4.probability
## death1          0.0000000        0.050596121
## death2          0.0000000        0.408033589
## death4          0.8997796        0.004802171
## death6          0.0000000        0.920166790
## death8          0.0000000        0.011788292
## death9          0.0000000        0.011788292
```

Make sure there is a set of probabilities for each death.

```
dim(NBC.results$cause.probabilities)
```

```
## [1] 500   4
```

Have a look at the CSMFs calculated by summing fractional deaths.

```
NBC.results$csmf.fracDeath
```

```
##                        cause1    cause2   cause3    cause4
## VAdeaths.fracDeath 0.2586384 0.3469112 0.189263 0.2051873
```

Have a look at the CSMFs calculated by summing assigned causes.

```
NBC.results$csmf.topCause
```

```
##                    cause1 cause2 cause3 cause4
## VAdeaths.topCause  0.288   0.35  0.204  0.158
```

Have a look when these causes where assigned.

```
NBC.results$when
```

```
## [1] "Fri Nov 09 09:01:24 2018"
```

Have a look at the input deaths.

```
head(NBC.results$deaths)
```

```
##        s1 s2 s3 s4 s5
## death1  0  0  1  1  1
## death2  0  1  0  1  1
## death4  0  0  0  1  1
## death6  0  1  1  1  0
## death8  1  1  0  0  1
## death9  1  1  0  0  1
```

Have a look at the input SCI.

```
NBC.results$sci
```

```
##                s1        s2        s3          s4        s5
## cause1 0.83582090 0.2835821 0.1268657 0.007462687 0.7686567
## cause2 0.30875576 0.3594470 0.7603687 0.755760369 0.8755760
## cause3 0.13402062 0.0000000 0.0000000 0.958762887 0.8041237
## cause4 0.05769231 0.8846154 0.7692308 0.461538462 0.3076923
```

InterVA Algorithm

Create a function to implement the InterVA VA coding algorithm. Comments in the code explain each step.

```
###### <<Function>>
calcInterVA <- function(SCI,SCI.c,deaths) {

  # implements the InterVA VA algorithm

  # SCI is a cause x symptoms matrix of SCI conditional probabilities p(s|c)
  # SCI.c is p(c)
  # deaths is a deaths x symptoms matrix of VA deaths containing binary symptom values
  # returns a list containing
  #   * top.cause: deaths x cause,(largest)cause-propensity matrix of InterVA-identified causes
  #   * cause.probabilities: deaths x cause-propensities matrix of all cause-propensities
  #   * csmf.fracDeath: CSMFs calculated by summing 'fractional deaths' for each cause,
  #   * csmf.topCause: CSMFs calculated by calculating the fraction of deaths assigned to each cause
  #   * when: date and time when finished
  #   * deaths: the input deaths
  #   * sci: the input SCI

  # InterVA.props are the InterVA propensities for each death: deaths x causes
  InterVA.props <- array(0,dim=c(nrow(deaths),nrow(SCI)))

  # InterVA.causes are causes for maximum InterVA propensities
  InterVA.causes <- data.frame(
    cause=as.character(rep("",nrow(deaths))),
    propensity=rep(0,nrow(deaths)),
    stringsAsFactors = FALSE
  )
  rownames(InterVA.causes) <- rownames(deaths)

  # calculate the InterVA propensities for each cause given symptoms vectors, p(c|s)
  # loop over deaths
  for (d in 1:nrow(deaths)) {
    # initialize the propensities
    InterVA.props[d,] <- SCI.c
    # loop over symptoms
    for (s in 1:ncol(SCI)) {
      # if the symptom exists
      if (deaths[d,s]==1) {
        # loop over causes
        for (c in 1:nrow(SCI)) {
          # update InterVA.propsensity for the current cause
          InterVA.props[d,c] <- InterVA.props[d,c]*SCI[c,s]
        }
```

```
      # normalize the propensities
      InterVA.props[d,] <- InterVA.props[d,]/sum(InterVA.props[d,])
      # # the following is an InterVA4.02 bug that we can turn on by uncommenting these lines
      # # truncate the propensities at 0.00001
      # for (c in 1:nrow(SCI)) {
      #   if (InterVA.props[d,c]<0.00001) {InterVA.props[d,c] <- 0}
      # }
    }
  }
  # for each death identify the cause with the largest propensity greater than 0.4,
  #   or if none, label as 'indeterminate'
  # take the first element from the which() statement because sometimes all values
  #   of InterVA.props[d,] are equal
  InterVA.causes[d,1] <- ifelse(InterVA.props[d,which(InterVA.props[d,]==max(InterVA.props[d,]))[1]]>
    paste("cause",which(InterVA.props[d,]==max(InterVA.props[d,]))[1],sep=""),"indeterminate")
  InterVA.causes[d,2] <- ifelse(InterVA.props[d,which(InterVA.props[d,]==max(InterVA.props[d,]))[1]]>
    InterVA.props[d,which(InterVA.props[d,]==max(InterVA.props[d,]))[1]],0)
}

# store all InterVA cause-specific propensities for each death in a labeled matrix
InterVA.summaries <- InterVA.props
rownames(InterVA.summaries) <- rownames(deaths)
# initialize vector of column labels, cause-probabilities
causes.InterVA <- c("cause1.propensity")
# loop over causes
for (c in 2:nrow(SCI)) {
  # append each cause-probability to the column labels
  causes.InterVA <- c(causes.InterVA,paste("cause",c,".propensity",sep=""))
}
colnames(InterVA.summaries) <- causes.InterVA

# calculate InterVA CSMFs by summing 'fractional deaths', i.e. cause-probabilities for each cause
csmf.fracDeath <- colSums(InterVA.summaries)/nrow(InterVA.summaries)
# make csmf.fracDeath a matrix
csmf.fracDeath <- matrix(csmf.fracDeath,nrow=1)
# label columns with cause names
colnames(csmf.fracDeath) <- rownames(SCI)
# label rows - subsitute() returns the name of its input
rownames(csmf.fracDeath) <- c(paste(substitute(deaths),".fracDeath",sep=""))

# calcualte CSMF by summing up classified deaths for each cause
# initialize vector of CSMFs
csmf.topCause <- c()
# loop over causes
for (c in 1:nrow(SCI)) {
  # calculate and append fraction of deaths assigned to each cause
  csmf.topCause <- c(csmf.topCause,length(which(InterVA.causes[,1]==paste("cause",c,sep="")))/nrow(dea
}
# make csmf.topCause a matrix
csmf.topCause <- matrix(csmf.topCause,nrow=1)
# label columns
colnames(csmf.topCause) <- rownames(SCI)
# label rows - subsitute() returns the name of its input
```

```
  rownames(csmf.topCause) <- c(paste(substitute(deaths),".topCause",sep=""))

  # create list of outputs to return
  ret.list = list(
    # deaths x cause,(largest)cause-probability matrix of InterVA-identified causes
    top.cause = InterVA.causes,
    # deaths x cause-probabilities matrix of all cause-probabilities
    cause.propensities = InterVA.summaries,
    # CSMFs calculated by summing 'fractional deaths' for each cause,
    #   i.e. cause-probabilities for each cause
    csmf.fracDeath = csmf.fracDeath,
    # CSMFs calculated by calculating the fraction of deaths assigned to each cause
    csmf.topCause = csmf.topCause,
    # datetime when saved
    when = format(Sys.time(),"%a %b %d %X %Y"),
    # deaths
    deaths = deaths,
    # SCI
    sci = SCI
  )

  # return list containing various outputs
  return(ret.list)

}
```

Use the InterVA algorithm to identify causes for the test deaths created above, and then look at all the elements of the list of results returned by the *calcInterVA()* function.

```
# define inputs for test prob.c is set of (marginal) p(c)
prob.c <- rep(1/nrow(SCIdeaths.SCI), nrow(SCIdeaths.SCI))  # don't know what it is, so uniform distribu
# prob.c <- csmf # use the correct marginals verify that it's
# what you expect
prob.c
```

```
## [1] 0.25 0.25 0.25 0.25
```

```
# create a 'VAdeaths' dataset from the test deaths
VAdeaths <- testDeaths[, 2:6]  # test deaths

# assign causes to the test deaths using the InterVA
# algorithm
InterVA.results <- calcInterVA(SCIdeaths.SCI, prob.c, VAdeaths)
```

Have a look at the results, very similar to the NBC algorithm above.

```
head(InterVA.results$top.cause)
```

```
##         cause propensity
## death1 cause2  0.8206435
## death2 cause2  0.6514659
## death4 cause3  0.4878167
## death6 cause4  0.6029366
## death8 cause1  0.6174554
## death9 cause1  0.6174554
```

```
dim(InterVA.results$top.cause)
```

```
## [1] 500   2
```

```
head(InterVA.results$cause.propensities)
```

```
##        cause1.propensity cause2.propensity
## death1      0.0011869272         0.8206435
## death2      0.0044553851         0.6514659
## death4      0.0036295288         0.4186977
## death6      0.0005154312         0.3965480
## death8      0.6174553914         0.3293252
## death9      0.6174553914         0.3293252
##        cause3.propensity cause4.propensity
## death1         0.0000000        0.17816957
## death2         0.0000000        0.34407874
## death4         0.4878167        0.08985601
## death6         0.0000000        0.60293656
## death8         0.0000000        0.05321942
## death9         0.0000000        0.05321942
```

```
dim(InterVA.results$cause.propensities)
```

```
## [1] 500   4
```

```
InterVA.results$csmf.fracDeath
```

```
##                        cause1    cause2    cause3    cause4
## VAdeaths.fracDeath 0.1612453 0.5220687 0.1158391 0.2008469
```

```
InterVA.results$csmf.topCause
```

```
##                   cause1 cause2 cause3 cause4
## VAdeaths.topCause  0.192   0.47  0.174    0.1
```

```
head(InterVA.results$deaths)
```

```
##        s1 s2 s3 s4 s5
## death1  0  0  1  1  1
## death2  0  1  0  1  1
## death4  0  0  0  1  1
## death6  0  1  1  1  0
## death8  1  1  0  0  1
## death9  1  1  0  0  1
```

```
InterVA.results$when
```

```
## [1] "Fri Nov 09 09:01:24 2018"
```

```
InterVA.results$sci
```

```
##                s1        s2        s3          s4        s5
## cause1 0.83582090 0.2835821 0.1268657 0.007462687 0.7686567
## cause2 0.30875576 0.3594470 0.7603687 0.755760369 0.8755760
## cause3 0.13402062 0.0000000 0.0000000 0.958762887 0.8041237
## cause4 0.05769231 0.8846154 0.7692308 0.461538462 0.3076923
```

InSilicoVA

InSilicoVA Algorithm

Create a function to implement the InSilicoVA VA coding algorithm. Comments in the code explain each step.

```
###### <<Function>>
calcInSilicoVA <- function(SCI,SCI.c,deaths,samples=500,alpha=0.05) {

  # implements the InSilicoVA VA algorithm

  # SCI is a cause x symptoms matrix of SCI conditional probabilities p(s|c)
  # SCI.c is p(c)
  # deaths is a deaths x symptoms matrix of VA deaths containing binary symptom values
  # samples is the number of samples to take from the joint distribution in the estimation step,
  #   default to 500
  # alpha is the Dirichlet alpha parameter, default to 0.05
  # returns a list containing
  #   * top.cause: deaths x cause,(largest)cause-probability matrix of InterVA-identified causes
  #   * cause.probabilities: deaths x cause-probabilities matrix of all cause-probabilities
  #   * csmf.median: CSMFs calculated as the median of the CSMF distributions
  #   * csmf.fivenum: five number summary of the CSMF distributions:
  #   * time: time taken to execute
  #   * when: date and time when finished
  #   * pi: array of individual probabilities of being assigned each cause for
  #   * Y: array of causes assigned for each sample: sample x causes x deaths
  #   * C: cause-specific mortality fractions for each sample: sample x causes
  #   * deaths: the input deaths
  #   * sci: the input SCI

  # record start time
  start.time <- Sys.time()

  # InSilicoVA.pi is array of individual probabilities of being assigned each cause for
  #   each sample: sample x causes x deaths
  InSilicoVA.pi <- array(0,dim=c(samples,nrow(SCI),nrow(deaths)))
  # InSilicoVA.Y is array of causes assigned for each sample: sample x causes x deaths
  InSilicoVA.Y <- array(0,dim=c(samples,nrow(SCI),nrow(deaths)))
  # InSilicoVA.C stores cause-specific mortality fractions for each sample: sample x causes
  InSilicoVA.C <- array(0,dim=c(samples,nrow(SCI)))
  # C is vector of cause-specific mortality fractions in each sample, initialize to SCI.c
  C <- SCI.c
  # Y is array of causes assigned in each sample: causes by deaths
  Y <- array(0,dim=c(nrow(SCI),nrow(deaths)))
  # pi is an array of probabilities of being assigned each cause in each sample: causes by deaths
  pi <- array(0,dim=c(nrow(SCI),nrow(deaths)))
  # m is vector of Dirichlet parameters
  m <- c(rep(0,nrow(SCI)))

  # loop over the sample, in each pass draw one value for the cause assignments and the CSMFs
  for (samp in 1:samples) {
    # loop over deaths,
    #   first time, initialize pi to first guess of CSMFs
    #   after that, update pi with current CSMFs
```

```
  for (i in 1:nrow(deaths)) {
    pi[,i] <- C
  }
  # for each death, update the probability of being assigned each cause conditional on current C, e.g
  # loop over deaths
  for (d in 1:nrow(deaths)) {
    # loop over symptoms
    for (s in 1:ncol(SCI)) {
      # loop over causes
      for (c in 1:nrow(SCI)) {
        # naive Bayes classifier used to update probabilities of being assigned each cause
        # multiplies the current fraction of deaths due to that cause by the naive Bayes liklihood
        #     of observing the recorded signs/symptoms for each death, conditional on the cause
        pi[c,d] <- pi[c,d] * SCI[c,s]^deaths[d,s] * (1-SCI[c,s])^(1-deaths[d,s])
      }
    }
  }
  # draw new causes for each death
  # loop over deaths
  for (d in 1:nrow(deaths)) {
    # draw a cause from a categorical distribution parameterized with the updated probabilities of
    # dying from each cause (death-specific, above)
    Y[,d] <- rmultinom(1,1,pi[,d])
    # for each death, record the normalized (sum to 1.0) death-specific probabilities of
    #   dying from each cause (the area under the full joint distribution adds to 1.0,
    #   not the area under the p(c|s) curve for a specific c and s)
    # loop over causes
    for (c in 1:nrow(SCI)) {
      InSilicoVA.pi[samp,c,d] <- pi[c,d]/sum(pi[,d])
    }
  }
  # record the new cause assignments
  InSilicoVA.Y[samp,,] <- Y
  # create the updated Dirichlet parameter vector
  for (c in 1:nrow(SCI)) {
    # for each cause of death, sum alpha and the total number of deaths just assigned to that cause
    m[c] <- alpha + sum(Y[c,])
  }
  # draw a new set of cause-specific mortality fractions from a Dirichlet using the updated
  #   parameter vector
  C <- rdirichlet(1,m)
  # record the updated cause-specific mortality fractions
  InSilicoVA.C[samp,] <- C
}

# store the median and mean cause-assignment probabilities
# initialize summaries with one row per death and columns for causes
InSilicoVA.summaries <- array(0,dim=c(nrow(deaths),(nrow(SCI)*2)))
rownames(InSilicoVA.summaries) <- rownames(deaths)
# initilize column names
sums.colnames <- c()
# loop over causes
for (c in 1:nrow(SCI)) {
```

```
  # make a column name for median and mean associated with this cause
  tmp1 <- paste("cause",c,".median",sep="")
  tmp2 <- paste("cause",c,".mean",sep="")
  # append the column names
  sums.colnames <- c(sums.colnames,tmp1,tmp2)
}
colnames(InSilicoVA.summaries) <- sums.colnames
# calculate and store the median and mean cause-assignment probabilities for each death
# loop over deaths
for (d in 1:nrow(deaths)) {
  # loop over causes
  for (c in 1:nrow(SCI)) {
    # calculate and store median and mean cause-assignment probabilities
    InSilicoVA.summaries[d,(c*2-1)] <- round(summary(InSilicoVA.pi[,c,d])[3],4)
    InSilicoVA.summaries[d,(c*2)] <- round(summary(InSilicoVA.pi[,c,d])[4],4)
  }
}

# InSilicoVA.causes are causes with largest median cause-assignment probability
InSilicoVA.causes <- data.frame(
  cause=as.character(rep("",nrow(deaths))),
  median.probability=rep(0,nrow(deaths)),
  stringsAsFactors = FALSE
)
rownames(InSilicoVA.causes) <- rownames(deaths)
# loop over deaths
for (d in 1:nrow(deaths)) {
  # loop over causes to calculate median cause-assignment probabilities
  # initialize medians
  medians <- rep(0,nrow(SCI))
  # loop over causes
  for (c in 1:nrow(SCI)) {
    # calculate and store the median cause-assignment probability for each cause
    medians[c] <- summary(InSilicoVA.pi[,c,d])[3]
  }
  # for each death identify the cause with the largest probability and store both
  #   the cause label and the probability
  # take the first element from the which() statement because sometimes all values
  #   of medians are equal
  InSilicoVA.causes[d,1] <- paste("cause",which(medians==max(medians))[1],sep="")
  InSilicoVA.causes[d,2] <- medians[which(medians==max(medians))[1]]
}

# define the CSMFs using median of the CSMF distributions
# matrix of cause-specific medians, transpose of apply output to create 1x5 matrix
csmf.median <- t(matrix(apply(InSilicoVA.C,2,median,nrow=1)))
# make column names
# initialize cause labels vector
causes <- "cause1"
# loop over causes
for (c in 2:nrow(SCI)) {
  # append each cause to the column labels
  causes <- c(causes,paste("cause",c,sep=""))
```

```
  }
  colnames(csmf.median) <- causes
  # label rows - subsitute() returns the name of its input
  rownames(csmf.median) <- c(paste(substitute(deaths),".median",sep=""))

  # calculate the five number summary of the CSMF distributions
  csmf.fivenum <- apply(InSilicoVA.C,2,fivenum)
  colnames(csmf.fivenum) <- causes
  rownames(csmf.fivenum) <- c("min","Q1","median","Q3","max")

  # record end time
  end.time <- Sys.time()

  # create list of outputs to return
  ret.list = list(
    # deaths x cause,(largest)cause-probability matrix of InterVA-identified causes
    top.cause = InSilicoVA.causes,
    # deaths x cause-probabilities matrix of all cause-probabilities
    cause.probabilities = InSilicoVA.summaries,
    # CSMFs calculated as the median of the CSMF distributions
    csmf.median = csmf.median,
    # five number summary of the CSMF distributions:
    #   smallest, 1st quartile, median, 3rd quartile, largest
    csmf.fivenum = csmf.fivenum,
    # time taken to execute
    time = end.time - start.time,
    # datetime when saved
    when = format(Sys.time(),"%a %b %d %X %Y"),
    # pi is array of individual probabilities of being assigned each cause for
    #   each sample: sample x causes x deaths
    pi = InSilicoVA.pi,
    # Y is array of causes assigned for each sample: sample x causes x deaths
    Y = InSilicoVA.Y,
    # C stores cause-specific mortality fractions for each sample: sample x causes
    C = InSilicoVA.C,
    # deaths
    deaths = deaths,
    # SCI
    sci = SCI
  )

  # return list containing various outputs
  return(ret.list)

}
```

Assign causes to the test deaths using the InSilicoVA VA cause-coding algorithm. InSilicoVA is different from the others and uses a sampling procedure to approximate the distributions of CSMFs and cause-assignment probabilities for each death. Because of this it takes longer to run and produces a number of additional outputs.

```
# define inputs for test prob.c is set of (marginal) p(c)
prob.c <- rep(1/nrow(SCIdeaths.SCI), nrow(SCIdeaths.SCI))  # don't know what it is, so uniform distribu
# prob.c <- csmf # use the correct marginals verify that it's
```

```r
# what you expect
prob.c
```

```
## [1] 0.25 0.25 0.25 0.25
```

```r
# test

# create a 'VAdeaths' dataset from the test deaths
VAdeaths <- testDeaths[, 2:6]  # test deaths

# assign casues to the test deaths using the InSilicoVA VA
# cause-coding algorithm With 500 samples, InSilicoVA takes
# about 20 seconds to run on a fast laptop, maybe longer on
# your computer
InSilicoVA.results <- calcInSilicoVA(SCIdeaths.SCI, prob.c, VAdeaths,
    500, 0.05)
# for faster testing ...  InSilicoVA.results <-
# calcInSilicoVA(SCIdeaths.SCI,prob.c,VAdeaths,5,0.05)
```

Have a look at the InSilicoVA resutls. There are outputs simmilar to the NBC and InterVA outputs, and there are additional InSilicoVA-specific outputs.

```r
head(InSilicoVA.results$top.cause)
```

```
##         cause median.probability
## death1 cause2          0.9860390
## death2 cause2          0.8441616
## death4 cause3          0.7499393
## death6 cause4          0.7511965
## death8 cause1          0.9315502
## death9 cause1          0.9315502
```

InSilicoVA produces estimated distributions of cause-assignment probabilities for each death. Have a look at the medians and means of those distributions for each death.

```r
head(InSilicoVA.results$cause.probabilities)
```

```
##        cause1.median cause1.mean cause2.median cause2.mean
## death1        0.0002      0.0002        0.9860      0.9854
## death2        0.0027      0.0027        0.8442      0.8394
## death4        0.0011      0.0011        0.2454      0.2470
## death6        0.0001      0.0001        0.2487      0.2481
## death8        0.9316      0.9311        0.0628      0.0631
## death9        0.9316      0.9311        0.0628      0.0631
##        cause3.median cause3.mean cause4.median cause4.mean
## death1        0.0000      0.0000        0.0138      0.0144
## death2        0.0000      0.0000        0.1533      0.1579
## death4        0.7499      0.7485        0.0033      0.0034
## death6        0.0000      0.0000        0.7512      0.7518
## death8        0.0000      0.0000        0.0058      0.0058
## death9        0.0000      0.0000        0.0058      0.0058
```

InSilicoVA produces estimated distributions of CSMFs. Have a look at the medians of those distributions.

```r
InSilicoVA.results$csmf.median
```

```
##                     cause1   cause2    cause3    cause4
## VAdeaths.median 0.2514812 0.469517 0.1512532 0.1252563
```

Have a look at the five-number summary of the estimated CSMF distributions.

```
InSilicoVA.results$csmf.fivenum
```

```
##           cause1    cause2    cause3     cause4
## min    0.1831128 0.3461120 0.1036031 0.06519362
## Q1     0.2358886 0.4494867 0.1395037 0.11227251
## median 0.2514812 0.4695170 0.1512532 0.12525626
## Q3     0.2665723 0.4918033 0.1625148 0.14070242
## max    0.3108396 0.5601692 0.2269965 0.20370514
```

```
InSilicoVA.results$time
```

```
## Time difference of 17.9235 secs
```

Have a look at the intermediate values produced by InSilicoVA for the individual cause-assignment probabilities assigned to each death at each iteration.

```
head(InSilicoVA.results$pi)
```

```
## [1] 0.0003646292 0.0002652768 0.0002366941 0.0002110826
## [5] 0.0002269591 0.0002178335
```

There are 500 iterations for 500 deaths; one probability for each of four causes.

```
dim(InSilicoVA.results$pi)
```

```
## [1] 500   4 500
```

Have a look at the intermediate values produced by InSilicoVA for the causes assigned to each death at each iteration.

```
head(InSilicoVA.results$Y)
```

```
## [1] 0 0 0 0 0 0
```

There are 500 iterations for 500 deaths; one of the four causes is assigned at each iteration.

```
dim(InSilicoVA.results$Y)
```

```
## [1] 500   4 500
```

Have a look at the intermediate values produced by InSilicoVA for the CSMFs assigned at each iteration.

```
head(InSilicoVA.results$C)
```

```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.2464234 0.3461120 0.2068372 0.2006274
## [2,] 0.2450134 0.3876852 0.1809727 0.1863287
## [3,] 0.2399865 0.4258668 0.1304416 0.2037051
## [4,] 0.2591402 0.4306369 0.1610991 0.1491239
## [5,] 0.2646780 0.4588201 0.1281346 0.1483674
## [6,] 0.2496273 0.4491892 0.1343381 0.1668453
```

There a CSMF for each of the four causes at each of the 500 iterations.

```
dim(InSilicoVA.results$C)
```

```
## [1] 500   4
```

```
InSilicoVA.results$when
```

```
## [1] "Fri Nov 09 09:01:42 2018"
```

```r
head(InSilicoVA.results$deaths)
```

```
##        s1 s2 s3 s4 s5
## death1  0  0  1  1  1
## death2  0  1  0  1  1
## death4  0  0  0  1  1
## death6  0  1  1  1  0
## death8  1  1  0  0  1
## death9  1  1  0  0  1
```

```r
InSilicoVA.results$sci
```

```
##                s1        s2        s3          s4        s5
## cause1 0.83582090 0.2835821 0.1268657 0.007462687 0.7686567
## cause2 0.30875576 0.3594470 0.7603687 0.755760369 0.8755760
## cause3 0.13402062 0.0000000 0.0000000 0.958762887 0.8041237
## cause4 0.05769231 0.8846154 0.7692308 0.461538462 0.3076923
```

InSilicoVA Plotting

Create a funcion to plot various outputs from InSilicoVA. The plotting function creates PDF files and saves them in the specified directory. The plots inlude

- convergence plots for individual cause-assignment probabilities for each cause for each death
- histograms for individual cause-assignment probability distributions for each cause for each death
- histograms of estimated CSMF distributions
- boxplots of estimated CSMF disributions

The plotting function requires a 'results' object produced by InSilicoVA. This contains all the deaths and intermediate values that are required for plotting.

```r
###### <<Function>>
plotInSilicoVA <- function(results, path, conv = 0, pi = 0, csmfH = 0,
    csmfB = 0) {

    # plotting functions for InSilicoVA results

    # results is an InSilicoVA results list path points to where
    # you want PDF files save, relative to the working directory
    # must be in form './figs/' including last slash if conv=1,
    # create convergence plots if pi=1, create histograms of pi
    # if csmfH=1, create CSMF histograms if csmfB=1, create CSMF
    # boxplots

    # create a list of cause names
    causes <- c("Cause 1")
    for (c in 2:nrow(results$sci)) {
        causes <- c(causes, paste("Cause", c))
    }
    # create a set of colors, one for each cause
    colors.cause <- rainbow(nrow(results$sci), alpha = 0.5)

    # grab the deaths from the results
    deaths <- results$deaths

    # assess sample convergence: plot the probability of being
```

```
    # assigned to each cause by sample
    if (conv == 1) {
        # loop over deaths
        for (d in 1:nrow(deaths)) {
            # loop over causes
            for (c in 1:nrow(results$sci)) {
                title <- paste("death ", d, ",", " cause ", c,
                  sep = "")
                file.name <- paste(path, "convergence-death",
                  d, ",", "cause", c, ".pdf", sep = "")
                pdf(file = file.name)
                # plot pi by sample
                plot(results$pi[, c, d], ylim = c(0, 1), main = title,
                  xlab = "Sample", ylab = "Probability")
                dev.off()
            }
        }
    }

    # a single histogram for each death with all cause-assignment
    # probability distributions
    if (pi == 1) {
        # loop over deaths
        for (d in 1:nrow(deaths)) {
            pdf(paste(path, "cause-distributions-death", d, ".pdf",
                sep = ""))
            hist(results$pi[, 1, d], col = colors.cause[1], xlim = c(0,
                1), main = paste("Death", d), xlab = "Probability")
            for (c in 2:nrow(results$sci)) {
                hist(results$pi[, c, d], col = colors.cause[c],
                  add = T)
            }
            legend("topright", causes, fill = colors.cause)
            dev.off()
        }
    }

    # plot CSMFs on one plot
    if (csmfH == 1) {
        pdf(paste(path, "CSMFs.pdf", sep = ""))
        hist(results$C[, 1], col = colors.cause[1], main = "CSMFs",
            xlab = "Probability", xlim = c(0, 1))
        for (c in 2:nrow(results$sci)) {
            hist(results$C[, c], col = colors.cause[c], add = T)
        }
        legend("topright", causes, fill = colors.cause)
        dev.off()
    }

    # boxplots of the CSMFs
    if (csmfB == 1) {
        pdf(paste(path, "CSMF-boxplots.pdf", sep = ""))
        boxplot(results$C, names = causes, main = "Summary of CSMF Distributions")
```

```
        dev.off()
    }

}
```

Test the plotting function. PDFs containing plots will be stored in '../figure/insilicova/'. This takes some time and saves many files, so evaluation is set to 'FALSE'. If you want to run it, change that to 'TRUE'.

```
getwd()
plotInSilicoVA(InSilicoVA.results, "../figure/insilicova/", 1,
    0, 0, 0)
plotInSilicoVA(InSilicoVA.results, "../figure/insilicova/", 0,
    1, 0, 0)
plotInSilicoVA(InSilicoVA.results, "../figure/insilicova/", 0,
    0, 1, 0)
plotInSilicoVA(InSilicoVA.results, "../figure/insilicova/", 0,
    0, 0, 1)
```

Tariff Algorithm

Create a function to implement the Tariff VA coding algorithm. Comments in the code explain each step.

```
###### <<Function>>
calcTariff <- function(GSdeaths,deaths,normalize,resample) {

  # implements the Tariff VA algorithm

  # GSdeaths is deaths by cause,sypmtoms matrix of gold standard deaths containing
  #   reference cause and binary symptoms values
  # deaths is a deaths x symptoms matrix of VA deaths containing binary symptom values
  # normalize is boolean indicating whether or not to normalize the tariff sums
  # resample is a boolean indicating whether or not to resample the gold standard deaths
  # returns a list containing
  #   * top.cause: deaths x cause,tariff-Q matrix of Tariff-identified causes
  #   * cause.tariffScores: deaths x cause-Q, tariff rank, matrix with all cause-Qs
  #   * csmf: CSMFs calculated from identified causes
  #   * when: datetime when saved
  #   * deaths: input deaths
  #   * tariffs: tariff values calculated from gold standard deaths, i.e. SCI for Tariff

  # calculate the Tariffs from the gold standard deaths

  # identify the causes in the gold standard deaths
  causes <- sort(unique(GSdeaths[,1]))
  # store the number of causes
  num.causes <- length(causes)
  # store the number of symptoms
  num.symptoms <- ncol(GSdeaths)-1
  # create a vector of cause labels
  cause.labels <- "cause1"
  for (i in 2:num.causes) {
    # create and append the cause label
    cause.labels <- c(cause.labels,paste("cause",i,sep=""))
  }
```

```
# create vector of symptoms labels
symptom.labels <- "s1"
for (s in 2:num.symptoms) {
  # create and append the symptom label
  symptom.labels <- c(symptom.labels,paste("s",s,sep=""))
}

# for each cause, calculate the fraction of deaths with each symptom, the tariff x
# initilize the list of (raw) tariff x vectors
x <- list()
# loop over causes
for (c in 1:num.causes) {
  # calculate the fraction of gold standard deaths that endorse each symptom, a by-symptom vector
  # append that vector to the list
  if (normalize) {
    x[[c]] <- colSums(GSdeaths[which(GSdeaths[,1]==c),2:6])/nrow(SCIdeaths[which(SCIdeaths[,1]==c),])
  } else {
    x[[c]] <- colSums(GSdeaths[which(GSdeaths[,1]==c),2:6])
  }
}
# create a cause x symptom matrix from the raw tariff list
x.mat <- matrix(unlist(x), ncol = 5, byrow = TRUE)

# create the tariff values by standardizing the (raw) tariff x values using the median
#   and interquartile range *across causes*
# create an empty T matrix
T <- x.mat-x.mat
# calculate the medians of x across causes, i.e. by column
medians <- apply(x.mat,2,median)
# calculate the interquartile ranges of x across causes
iqrs <- apply(x.mat,2,IQR)
# loop over causes
for (c in 1:num.causes) {
  # calculate and store the median-IQR standardized tariff values
  T[c,] <- (x.mat[c,] - medians) / iqrs
}
colnames(T) <- symptom.labels
rownames(T) <- cause.labels

# create the gold standard tariff scores for standardizing Q values
# resample the gold standard deaths with replacement to create a uniform distribution by cause
pi.bootstrap <- rdirichlet(1, rep(1, length(causes)))
# n.bootstrap <- round(pi.bootstrap * dim(GSdeaths)[1])
n.bootstrap <- table(sample(causes,nrow(GSdeaths),replace=TRUE,prob=pi.bootstrap))
# ensure that each cause is sampled at least once
for (c in 1:num.causes) {
  if (is.na(n.bootstrap[c])) {
    n.bootstrap[c] <- 1
  }
}
# create array for bootstrapped gold standard deaths
GSdeaths.bootstrap <-  array(0,dim=c(sum(n.bootstrap), ncol(GSdeaths)))
```

```
# sample the gold standard deaths with replacement by cause with the number for each cause
#   determined by the dirichlet sample drawn just above
# initialize the index for the array of bootstrapped gold standard deaths to 0,
#   none stored yet
nsum <- 0
# loop over causes
for(c in 1:length(causes)){
  # store the number of deaths to sample for this cause
  nn <- n.bootstrap[c]
  # store the sampled deaths for this cause in the bootstrapped gold standard deaths array
  GSdeaths.bootstrap[(nsum+1):(nsum+nn),] <-
    GSdeaths[sample(which(GSdeaths[,1]==c),nn,replace=TRUE),]
  # increment the index for the array of bootstrapped gold standard deaths by the number of
  #   deaths just sampled for this cause
  nsum <- nsum + nn
}
# ensure that the bootstrapped data set has at most the original number of deaths
GSdeaths.bootstrap <- GSdeaths.bootstrap[1:nrow(GSdeaths),]

# pick the gold standard deaths to use when calculating the gold standard tariff scores, just below
#   if 'resample' is TRUE, then use the resampled (bootstrapped) gold standard deaths
if (resample) {
  GSdeaths.S <- GSdeaths.bootstrap
} else {
  GSdeaths.S <- GSdeaths
}

# create the gold standard tariff scores for each cause and each death for
#   standardizing Q values, later
# initilize a deaths x causes array of S values
S.gs <-  array(0,dim=c(nrow(GSdeaths.S),num.causes))
# loop over gold standard deaths
for (d in 1:nrow(GSdeaths.S)) {
  # loop over causes
  for (c in 1:num.causes) {
    # loop over symptoms
    for (s in 1:num.symptoms) {
      # calculate and store the tariff score for this death and cause
      S.gs[d,c] <- S.gs[d,c] + T[c,s] * GSdeaths.S[d,(s+1)]
    }
  }
}
# label the cause columns of the gold standard tariff scores
colnames(S.gs) <- cause.labels

# implement cause-assignment using tariffs

# calculate tariff scores for each cause for each death in the data (not gold standard)
# initialize a deaths x cause arrage of S
S <-  array(0,dim=c(nrow(deaths),num.causes))
# loop over deaths
for (d in 1:nrow(deaths)) {
  # loop over causes
```

```
  for (c in 1:num.causes) {
    # loop over symptoms
    for (s in 1:num.symptoms) {
      # calcualte and store the tariff score for this death and cause
      S[d,c] <- S[d,c] + T[c,s] * deaths[d,s]
    }
  }
}

# calculate tariff score rank in the gold standard distribution of tariff scores by cause
# create an empty Q matrix
Q <- S-S
# loop over deaths
for (d in 1:nrow(deaths)) {
  # loop over causes
  for (c in 1:num.causes) {
    # rank of the tariff score for this cause in the distribution of gold standard tariff
    #   scores for this cause
    # add the tariff score for this death to the gold standard distribuion and calculate
    #   a normalized rank score
    Q[d,c] <- tail(rank(c(S.gs[which(GSdeaths[,1]==c),c],S[d,c]))
                   /length(c(S.gs[which(GSdeaths[,1]==c),c],S[d,c])),n=1)
  }
}

# for each death identify causes with largest tariff scores
# create an empty data frame to hold results
Tariff.causes <- data.frame(
  cause=as.character(rep("",nrow(deaths))),
  Q=rep(0,nrow(deaths)),
  stringsAsFactors = FALSE
)
rownames(Tariff.causes) <- rownames(deaths)
# loop over deaths
for (d in 1:nrow(deaths)) {
  # identify and store the cause and Q value for the cause with the largest Q for this death
  Tariff.causes[d,1] <- paste("cause",which(Q[d,]==max(Q[d,]))[1],sep="")
  Tariff.causes[d,2] <- Q[d,which(Q[d,]==max(Q[d,]))[1]]
}

# store all Tariff results
Tariff.summaries <- Q
rownames(Tariff.summaries) <- rownames(deaths)
# create a vector of tariff Q labels
causes.tariff <- c("cause1.Q")
for (c in 2:num.causes) {
  # create and append each tariff Q label
  causes.tariff <- c(causes.tariff,paste("cause",c,".Q",sep=""))
}
colnames(Tariff.summaries) <- causes.tariff

# calcualte and store tariff CSMFs
# initialize vector of CSMFs
```

```
  csmf <- c()
  # loop over causes
  for (c in 1:num.causes) {
    # calculate the fraction of deaths assigned to this cause and append to the vector
    csmf <- c(csmf,length(which(Tariff.causes[,1]==paste("cause",c,sep="")))/nrow(deaths))
  }
  # create a matrix from the vector
  csmf <- matrix(csmf,nrow=1)
  colnames(csmf) <- cause.labels
  # label rows - subsitute() returns the name of its input
  rownames(csmf) <- c(substitute(deaths))

  # create list of outputs to return
  ret.list = list(
    # deaths x cause,tariff-Q matrix of Tariff-identified causes
    top.cause = Tariff.causes,
    # deaths x cause-Q, tariff rank, matrix with all cause-Qs
    cause.tariffScores = Tariff.summaries,
    # CSMFs calculated from identified causes
    csmf = csmf,
    # gold standard deaths
    deaths.gs = GSdeaths,
    # gold standard tariff scores for each death for each cause
    scores.gs = S.gs,
    # datetime when saved
    when = format(Sys.time(),"%a %b %d %X %Y"),
    # deaths
    deaths = deaths,
    # SCI
    tariffs = T
  )

  # return list containing various outputs
  return(ret.list)

}
```

Use the Tariff algorithm to identify causes for the test deaths created above, and then look at all the elements of the list of results returned by the *calcTariff()* function.

```
# create a 'VAdeaths' dataset from the test deaths
VAdeaths <- testDeaths[, 2:6]  # test deaths

# assign causes to the test deaths using the InterVA
# algorithm set 'normalize' to TRUE and 'resample' to TRUE so
# that Tariff is run the way it was originally written
Tariff.results <- calcTariff(SCIdeaths, VAdeaths, TRUE, TRUE)
```

Tariff causes are assigned by choosing the cause with the largest normalized Tariff Score, Q, value.

```
head(Tariff.results$top.cause)
```

```
##         cause         Q
## death1 cause2 0.8646789
## death2 cause2 0.7064220
```

```
## death4 cause3 0.8826531
## death6 cause4 0.9905660
## death8 cause1 0.7666667
## death9 cause1 0.7666667
```

Tariff does not produce probabilities for each cause. What it does produce are normalized Tariff Scores or 'Q' values. Have a look at those for each cause for each death

```
head(Tariff.results$cause.tariffScores)
```

```
##          cause1.Q  cause2.Q  cause3.Q  cause4.Q
## death1 0.05925926 0.8646789 0.6479592 0.5283019
## death2 0.10370370 0.7064220 0.4132653 0.6415094
## death4 0.23703704 0.5412844 0.8826531 0.3207547
## death6 0.04074074 0.6467890 0.2142857 0.9905660
## death8 0.76666667 0.6788991 0.1581633 0.6226415
## death9 0.76666667 0.6788991 0.1581633 0.6226415
```

Tariff CSMFs are calculated by summing assigned causes.

```
Tariff.results$csmf
```

```
##          cause1 cause2 cause3 cause4
## VAdeaths  0.184  0.406  0.166  0.244
```

```
Tariff.results$when
```

```
## [1] "Fri Nov 09 09:01:43 2018"
```

```
head(Tariff.results$deaths)
```

```
##        s1 s2 s3 s4 s5
## death1  0  0  1  1  1
## death2  0  1  0  1  1
## death4  0  0  0  1  1
## death6  0  1  1  1  0
## death8  1  1  0  0  1
## death9  1  1  0  0  1
```

```
Tariff.results$tariffs
```

```
##                s1         s2         s3         s4
## cause1  1.8871740 -0.1364219 -0.4745803 -1.3112277
## cause2  0.2683415  0.1364219  0.4745803  0.3208586
## cause3 -0.2683415 -1.1563086 -0.6646598  0.7636205
## cause4 -0.5027770  2.0251599  0.4878582 -0.3208586
##                s5
## cause1 -0.1051989
## cause2  0.5290692
## cause3  0.1051989
## cause4 -2.8397376
```

Look at the (resampled) gold standard death Tariff scores. The reason for the last ranking step in comparing Tariff scores is because for each cause the Tariff scores have a characteristic value and distribution, so they are not directly comparable. What is comparable is their rank withing the gold standard distributions of Tariff Scores. Below the different characteristic values for each cause are obvious.

```
# have a look at the gold standard tariff scores dataset
head(Tariff.results$scores.gs)
```

```
##          cause1    cause2     cause3     cause4
## [1,]  1.7819751 0.7974107 -0.1631426 -3.3425145
## [2,] -0.2416208 0.6654911 -1.0511098 -0.8145777
## [3,]  1.7819751 0.7974107 -0.1631426 -3.3425145
## [4,]  1.8871740 0.2683415 -0.2683415 -0.5027770
## [5,]  1.7819751 0.7974107 -0.1631426 -3.3425145
## [6,] -0.2416208 0.6654911 -1.0511098 -0.8145777
```

```
dim(Tariff.results$scores.gs)
```

```
## [1] 500   4
```

```
# loop through causes and have a look at the summary
# statistics for the distributions of gold standard tariff
# scores for the
num.causes <- ncol(Tariff.results$scores.gs)
for (c in 1:num.causes) {
    print(colnames(Tariff.results$scores.gs)[c])
    print(summary(Tariff.results$scores.gs[which(Tariff.results$deaths.gs[,
        1] == c), c]))
}
```

```
## [1] "cause1"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.0274 -1.4164 -0.2416 -0.1537  1.4126  1.8872
## [1] "cause2"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.5892  0.8499  0.8492  1.1183  1.7293
## [1] "cause3"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.9841 -0.9521 -0.1631 -0.2288  0.7636  0.8688
## [1] "cause4"
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -3.6634 -3.1606 -2.6727 -1.9271 -0.6476  2.1922
```

Tools for Comparing Results Produced by Each Algorithm and Characterizing the Performance the Algorithms

Create a function to calculate the number of discordant deaths among results from all of the algorithms.

```
###### <<Function>>
calcConcordance <- function(topCauses,refDeaths) {

  # calculate concordance among real cause and assigned causes

  # topCauses is a named list of top assigned causes, e.g.
    # topCauses <- list(
    #   NBC =  NBC.results$top.cause,
    #   InterVA = InterVA.results$top.cause,
    #   InSilicoVA = InSilicoVA.results$top.cause,
    #   Tariff = Tariff.results$top.cause
    # )
  # refDeaths is a set of deaths with a reference cause, e.g.
    # refCSMF <- testDeaths.CSMF
  # returns a list containing
```

```
#   * discordant, matrix of discordant cause counts
#   * percent.correct, matrix of percent count of causes that are the same

# creat a dataframe with the cause assignments from all the inputs
# first, strip off the cause-assignment probabilities, etc.
# loop over the inputs
for (i in 1:length(topCauses)) {
  # store just the cause label for this input
  tmp <- topCauses[[i]][,1]
  # bring the death names along as labels for the causes
  names(tmp) <- rownames(topCauses[[i]])
  # store the assigned causes back into the 'topCauses' list
  topCauses[[i]] <- tmp
}
# second, merge the inputs into a dataframe
# merge on the row names that identify the deaths, the ensures that deaths are
#   matched across inputs
# merge the first two inputs' assigned causes and store in a dataframe called 'cause.df'
cause.df <- merge(topCauses[[1]],topCauses[[2]],by="row.names")
# assign the merged death names to rownames of the dataframe
rownames(cause.df) <- cause.df$Row.names
# remove the merged death names column from the dataframe
cause.df <- cause.df[,2:ncol(cause.df)]
# name the columns of the dataframe using the names from teh 'topCauses' list
colnames(cause.df) <- c(names(topCauses)[1:2])
# if there are more than two inputs, repeat the previous few steps for the remainder of the inputs
if (length(topCauses)>2) {
  for (i in 3:length(topCauses)) {
    column.names <- c(colnames(cause.df),names(topCauses)[i])
    cause.df <- merge(cause.df,topCauses[[i]],by="row.names")
    rownames(cause.df) <- cause.df$Row.names
    cause.df <- cause.df[,2:ncol(cause.df)]
    colnames(cause.df) <- column.names
  }
}

# identify all the cause names
# initialize a vector of all the cause names using the names from the first input
cause.names <- as.character(cause.df[,1])
# loop over the inputs, columns in 'cause.df'
for (c in 2:ncol(cause.df)) {
  # append the cause names for this input
  cause.names <- c(cause.names,as.character(cause.df[,c]))
}
# find the set of unique cause names
cause.names <- unique(cause.names)
# store number of unique cause names
causes <- length(cause.names)
# store the levels of the cause names
cause.levels <- levels(as.factor(cause.names))
# recode test death causes from numeric labels to character, i.e. 'cause1' or 'cause2'
# initialize new cause labels vector
refDeaths.causes <- rep("",nrow(refDeaths))
```

```
  # create recode string
  recode.str <- "1='cause1'"
  # loop over causes
  for (c in 2:causes) {
    recode.str <- paste(recode.str,";",c,"='cause",c,"'",sep="")
  }
  # recode!
  suppressWarnings(refDeaths.causes <- recode(refDeaths[,1],recode.str))

  # add real causes to Results.causes
  # store the existing column names
  column.names <- colnames(cause.df)
  # merge the reference death causes
  cause.df <- merge(refDeaths.causes,cause.df,by="row.names")
  # name the deaths using the merged death names
  rownames(cause.df) <- cause.df[,1]
  # remove the merged death names column
  cause.df <- cause.df[,2:ncol(cause.df)]
  # name the columns
  colnames(cause.df) <- c("Reference",column.names)
  # ensure that all columns of 'cause.df' have the same levels
  for (c in 1:ncol(cause.df)) {
    levels(cause.df[,c]) <- cause.levels
  }

  # count discordant causes for all combinations of inputs and refernece cause
  # number of inputs and reference
  comparitors <- ncol(cause.df)
  # initialize array of discordant counts
  discordant <- array(0,dim=c(comparitors,comparitors))
  # loop over comparitors
  for (ref in 1:comparitors) {
    # loop over comparitors
    for (comp in 1:comparitors) {
      # count dicordant causes for current combination of comparitors
      discordant[ref,comp] <- length(which(cause.df[,ref]!=cause.df[,comp]))
    }
  }
  # name the rows and columns of discordant with the names of the comparitors
  rownames(discordant) <- colnames(cause.df)
  colnames(discordant) <- colnames(cause.df)

  # create the return list
  ret <- list(
    # the discordant matrix
    discordant = discordant,
    # the percent correct matrix
    percent.correct = 100-100*discordant/nrow(VAdeaths)
  )

  return(ret)

}
```

Calculate the discordant and percent correct matrices for all the results generated so far.

```
# test creat a named list of the top cause assignments
topCauses <- list(NBC = NBC.results$top.cause, InterVA = InterVA.results$top.cause,
    InSilicoVA = InSilicoVA.results$top.cause, Tariff = Tariff.results$top.cause,
    InSilicoVA = InSilicoVA.results$top.cause, Tariff = Tariff.results$top.cause)
# calculate and have a look at the concordance matrices
# InSilicoVA and Tariff included twice to test the ability to
# include an arbitrary number of inputs
concordance <- calcConcordance(topCauses, testDeaths)
```

Have a look at the discordant matrix.

```
concordance$discordant
```

```
##            Reference NBC InterVA InSilicoVA Tariff
## Reference          0  98      92         78    123
## NBC               98   0      92         56    100
## InterVA           92  92       0         56     91
## InSilicoVA        78  56      56          0    108
## Tariff           123 100      91        108      0
## InSilicoVA        78  56      56          0    108
## Tariff           123 100      91        108      0
##            InSilicoVA Tariff
## Reference          78    123
## NBC                56    100
## InterVA            56     91
## InSilicoVA          0    108
## Tariff            108      0
## InSilicoVA          0    108
## Tariff            108      0
```

Have a look at the percent correct matrix.

```
concordance$percent.correct
```

```
##            Reference   NBC InterVA InSilicoVA Tariff
## Reference      100.0  80.4    81.6       84.4   75.4
## NBC             80.4 100.0    81.6       88.8   80.0
## InterVA         81.6  81.6   100.0       88.8   81.8
## InSilicoVA      84.4  88.8    88.8      100.0   78.4
## Tariff          75.4  80.0    81.8       78.4  100.0
## InSilicoVA      84.4  88.8    88.8      100.0   78.4
## Tariff          75.4  80.0    81.8       78.4  100.0
##            InSilicoVA Tariff
## Reference        84.4   75.4
## NBC              88.8   80.0
## InterVA          88.8   81.8
## InSilicoVA      100.0   78.4
## Tariff           78.4  100.0
## InSilicoVA      100.0   78.4
## Tariff           78.4  100.0
```

Create a function to calculate CSMF Accuracies to compare CSMFs produced by all four algorithms to the CSMFs of the reference deaths.

```
###### <<Function>>
calcCSMFaccuracies <- function(CSMFs, refCSMF) {
```

```
    # calculate CSMF accuracies, etc.

    # CSMFs is a named list of variable number algorithm result
    # objects, e.g.  CSMFs <- list( NBC.frac =
    # NBC.results$csmf.fracDeath, NBC.top =
    # NBC.results$csmf.topCause, InterVA.frac =
    # InterVA.results$csmf.fracDeath, InterVA.top =
    # InterVA.results$csmf.topCause, InSilicoVA =
    # InSilicoVA.results$csmf.median, Tariff =
    # Tariff.results$csmf ) refCSMF is a reference set of CSMFs
    # produced by calcCSMF(), e.g.  refCSMF <- testDeaths.CSMF
    # returns a list containing * CSMF.errors, raw CSMF errors
    # compared to the reference * CSMF.tae, total absolute errors
    # * CSMF.percErr, percent proportional errors *
    # CSMF.percErr.tae, total absolute percent proportional
    # errors * CSMF.accuracies, CSMF Accuracies

    # wrap up CSMFs nicely
    CSMF.mat <- rbind(refCSMF, CSMFs[[1]])
    for (c in 2:length(CSMFs)) {
        CSMF.mat <- rbind(CSMF.mat, CSMFs[[c]])
    }
    rownames(CSMF.mat) <- c("reference", names(CSMFs))
    CSMF.mat

    # Error in CSMF, subtract first (reference) row from all rows
    CSMF.errors <- sweep(CSMF.mat, 2, CSMF.mat[1, ])
    # total absolute error in CSMFs
    CSMF.tae <- rowSums(abs(CSMF.errors))

    # proportional error
    CSMF.percErr <- 100 * sweep(CSMF.errors, 2, CSMF.mat[1, ],
        FUN = "/")
    # total absolute proportional error
    CSMF.percErr.tae <- rowSums(abs(CSMF.percErr))

    # CSMF accuracy
    CSMF.accuracies <- c()
    for (i in 1:nrow(CSMF.mat)) {
        CSMF.accuracies <- c(CSMF.accuracies, 1 - sum(abs(CSMF.mat[i,
            ] - CSMF.mat[1, ]))/(2 * (1 - min(CSMF.mat[1, ]))))
    }
    names(CSMF.accuracies) <- rownames(CSMF.mat)
    CSMF.accuracies

    ret <- list(CSMF.errors = CSMF.errors, CSMF.tae = CSMF.tae,
        CSMF.percErr = CSMF.percErr, CSMF.percErr.tae = CSMF.percErr.tae,
        CSMF.accuracies = CSMF.accuracies)

    return(ret)

}
```

Calculate and comapare the CSMF Accuracies of all four algorithms.

```
# test create named list of CSMFs; names will be used
# throughout, so must be meaningful repeats test for
# expandable list of inputs
CSMFs <- list(NBC.frac = NBC.results$csmf.fracDeath, NBC.top = NBC.results$csmf.topCause,
    InterVA.frac = InterVA.results$csmf.fracDeath, InterVA.top = InterVA.results$csmf.topCause,
    InSilicoVA = InSilicoVA.results$csmf.median, Tariff = Tariff.results$csmf,
    InterVA.top = InterVA.results$csmf.topCause, InSilicoVA = InSilicoVA.results$csmf.median)
# calculate CSMF accuracies
CSMF.accuracies <- calcCSMFaccuracies(CSMFs, testDeaths.CSMF)
CSMF.accuracies$CSMF.accuracies
```

```
##    reference     NBC.frac      NBC.top InterVA.frac
##    1.0000000    0.8565590    0.8600451    0.8477251
##  InterVA.top   InSilicoVA       Tariff  InterVA.top
##    0.9638826    0.9684158    0.8532731    0.9638826
##   InSilicoVA
##    0.9684158
```

The larger the CSMF Accuracy, the closer the algorithm came to the truth or reference deaths. A value of 1.0 means that the CMSFs match the reference deaths exactly. This is always true when the reference deaths are compared to themsevles, 'cause' above, and usually InSilicoVA is has the largest CSMF Accuracy in this test. The consistent ranking is InSilicoVA followed by either InterVA.top or Tariff.

Cross Validation

Create a function to run a cross validation simulation. The inputs are a set of SCI and a set of CSMFs. A set of deaths are simulated using those and then sampled using an input sampling fraction. The sample of deaths are used to prepare a set of SCI, and the remaining deaths are used as test deaths. Starting from the simulated deaths, the idea is to test the algorithms using the same SCI, and additionally, having that SCI come from the same population of deaths. The algorithms are all run using the prepared SCI and tested on the same test deaths, and the results from each are compared.

```
crossValidate <- function(SCI.denovo, csmf, sim.deaths, samp.frac,
    insilico.reps = 500, insilico.alpha = 0.05) {

    # run a cross validation set * simulate deaths * draw a
    # sample of the simulated deaths * use the sample to create
    # SCI * run all the algorithms using the SCI from the sample
    # to assign casues to the deaths that were not sampled, the
    # 'test deaths' * compare the assigned causes from each
    # algorithm to the real causes in the test deaths

    # SCI.denovo is a list of <symptom>-element SCI vectors, one
    # for each cause csmf is a <cause>-element vector of CSMFs
    # sim.deaths is number of deaths to simulate samp.frac is a
    # decimal number representing the fraction of deaths to
    # sample insilico.reps is the number of sampling repetitions
    # for InSilicoVA insilico.alpha is InSilicoVA's alpha
    # parameter

    # returns a list of * simDeaths, a matrix of the simulated
    # deaths * result objects for each algorithm, NBC.results,
    # etc.  * concordance, a concordance object comparing the
```

```
# results * CSMF.accuracies, a CSMF accuracies object
# comparing CSMF results

# store SCI defined above
SCI <- storeSCI(SCI.denovo)
# simulate a bunch of deaths using the SCI and csmf defined
# above
simDeaths <- simulateDeaths(sim.deaths, csmf, SCI)
# draw sample for to prepare SCI
sample <- sample(nrow(simDeaths), samp.frac * nrow(simDeaths))
# select the sampled deaths to be used to create SCI
SCIdeaths <- simDeaths[sample, ]
# select the inverse of the sample as deaths for testing
testDeaths <- simDeaths[(-sample), ]
# calcualte teh CSMF of the SCI and test deaths
testDeaths.CSMF <- calcCSMF(testDeaths)
# calcualte SCI from the SCI deaths
SCIdeaths.SCI <- calcSCI(SCIdeaths)

# create a starting p(c) for all the methods, a uniform
# distribution
prob.c <- rep(1/nrow(SCIdeaths.SCI), nrow(SCIdeaths.SCI))
VAdeaths <- testDeaths[, 2:6]
# run NBC
NBC.results <- calcNBC(SCIdeaths.SCI, prob.c, VAdeaths)
# run InterVA
InterVA.results <- calcInterVA(SCIdeaths.SCI, prob.c, VAdeaths)
# run InSilicoVA - it take a lot longer!
InSilicoVA.results <- calcInSilicoVA(SCIdeaths.SCI, prob.c,
    VAdeaths, insilico.reps, insilico.alpha)
# run Tariff
Tariff.results <- calcTariff(SCIdeaths, VAdeaths, TRUE, TRUE)

# concordance
topCauses <- list(NBC = NBC.results$top.cause, InterVA = InterVA.results$top.cause,
    InSilicoVA = InSilicoVA.results$top.cause, Tariff = Tariff.results$top.cause)
# calculate and have a look at the concordance matrices
concordances <- calcConcordance(topCauses, testDeaths)

# CSMF accuracies create named list of CSMFs; names will be
# used throughout, so must be meaningful
CSMFs <- list(NBC.frac = NBC.results$csmf.fracDeath, NBC.top = NBC.results$csmf.topCause,
    InterVA.frac = InterVA.results$csmf.fracDeath, InterVA.top = InterVA.results$csmf.topCause,
    InSilicoVA = InSilicoVA.results$csmf.median, Tariff = Tariff.results$csmf)
# calculate CSMF accuracies
CSMF.accuracies <- calcCSMFaccuracies(CSMFs, testDeaths.CSMF)

# make a return list
rep <- list(simDeaths = simDeaths, NBC.results = NBC.results,
    InterVA.results = InterVA.results, InSilicoVA.results = InSilicoVA.results,
    Tariff.results = Tariff.results, concordance.list = concordances,
    CSMFaccuracies.list = CSMF.accuracies)
```

```
    return(rep)

}
```

Test the cross validation function.

```
# test: run a simulation create de novo SCI consisting of
# conditional probabilities p(s|c): causes x 5 symptoms you
# can add or subtract causes, i.e. rows below
SCI.denovo <- list()
SCI.denovo[[1]] <- c(0.9, 0.4, 0.1, 0.01, 0.8)
SCI.denovo[[2]] <- c(0.3, 0.41, 0.8, 0.75, 0.9)
SCI.denovo[[3]] <- c(0.1, 0.01, 0.01, 0.95, 0.8)
SCI.denovo[[4]] <- c(0.05, 0.9, 0.75, 0.5, 0.2)
# define a set of CSMFs, fraction of deaths that are for each
# cause: cause1, cause2, etc.
csmf <- c(0.25, 0.45, 0.2, 0.1)
# number of deaths to simulate
sim.deaths <- 1000
# sample for preparing SCI
samp.frac <- 0.5
# run the cross validation; leave the InSilicoVA parameters
# at their default values
crossVal <- crossValidate(SCI.denovo, csmf, sim.deaths, samp.frac)
```

Have a look at the percent correct cause assignments,

```
crossVal$concordance.list$percent.correct
```

```
##            Reference   NBC InterVA InSilicoVA Tariff
## Reference      100.0  85.0    83.4       84.4   73.6
## NBC             85.0 100.0    83.0       88.2   78.4
## InterVA         83.4  83.0   100.0       93.4   82.6
## InSilicoVA      84.4  88.2    93.4      100.0   87.4
## Tariff          73.6  78.4    82.6       87.4  100.0
```

and the CSMF Accuracies.

```
crossVal$CSMFaccuracies.list$CSMF.accuracies
```

```
##    reference     NBC.frac      NBC.top InterVA.frac
##    1.0000000    0.9047389    0.9049774    0.8148146
##  InterVA.top   InSilicoVA       Tariff
##    0.9162896    0.9762592    0.8371041
```

Now conduct do 50 simulations and have a look at the average performance of the algorithms

```
# InSilicoVA iterations insilico.iter <- 500
insilico.iter <- 500
# number of cross validation simulations to run sims <- 250
sims <- 250
# list to store CSMF comparisons from each
csmf.list <- list()
# number of deaths to simulate sim.deaths <- 1000
sim.deaths <- 1000
# list to store discordant matrices from each
discord.list <- list()
# loop over the simulations
```

```
for (sim in 1:sims) {

    # print the sim number
    print(sim)
    # run the cross validation
    crossVal <- crossValidate(SCI.denovo, csmf, sim.deaths, samp.frac,
        insilico.iter)
    # store the csmf comparisons and concordance matrix
    csmf.list[[sim]] <- crossVal$CSMFaccuracies.list$CSMF.accuracies
    discord.list[[sim]] <- crossVal$concordance.list$percent.correct

}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
## [1] 29
## [1] 30
## [1] 31
## [1] 32
## [1] 33
## [1] 34
## [1] 35
## [1] 36
## [1] 37
## [1] 38
## [1] 39
## [1] 40
## [1] 41
```

```
## [1] 42
## [1] 43
## [1] 44
## [1] 45
## [1] 46
## [1] 47
## [1] 48
## [1] 49
## [1] 50
## [1] 51
## [1] 52
## [1] 53
## [1] 54
## [1] 55
## [1] 56
## [1] 57
## [1] 58
## [1] 59
## [1] 60
## [1] 61
## [1] 62
## [1] 63
## [1] 64
## [1] 65
## [1] 66
## [1] 67
## [1] 68
## [1] 69
## [1] 70
## [1] 71
## [1] 72
## [1] 73
## [1] 74
## [1] 75
## [1] 76
## [1] 77
## [1] 78
## [1] 79
## [1] 80
## [1] 81
## [1] 82
## [1] 83
## [1] 84
## [1] 85
## [1] 86
## [1] 87
## [1] 88
## [1] 89
## [1] 90
## [1] 91
## [1] 92
## [1] 93
## [1] 94
## [1] 95
```

```
## [1] 96
## [1] 97
## [1] 98
## [1] 99
## [1] 100
## [1] 101
## [1] 102
## [1] 103
## [1] 104
## [1] 105
## [1] 106
## [1] 107
## [1] 108
## [1] 109
## [1] 110
## [1] 111
## [1] 112
## [1] 113
## [1] 114
## [1] 115
## [1] 116
## [1] 117
## [1] 118
## [1] 119
## [1] 120
## [1] 121
## [1] 122
## [1] 123
## [1] 124
## [1] 125
## [1] 126
## [1] 127
## [1] 128
## [1] 129
## [1] 130
## [1] 131
## [1] 132
## [1] 133
## [1] 134
## [1] 135
## [1] 136
## [1] 137
## [1] 138
## [1] 139
## [1] 140
## [1] 141
## [1] 142
## [1] 143
## [1] 144
## [1] 145
## [1] 146
## [1] 147
## [1] 148
## [1] 149
```

```
## [1] 150
## [1] 151
## [1] 152
## [1] 153
## [1] 154
## [1] 155
## [1] 156
## [1] 157
## [1] 158
## [1] 159
## [1] 160
## [1] 161
## [1] 162
## [1] 163
## [1] 164
## [1] 165
## [1] 166
## [1] 167
## [1] 168
## [1] 169
## [1] 170
## [1] 171
## [1] 172
## [1] 173
## [1] 174
## [1] 175
## [1] 176
## [1] 177
## [1] 178
## [1] 179
## [1] 180
## [1] 181
## [1] 182
## [1] 183
## [1] 184
## [1] 185
## [1] 186
## [1] 187
## [1] 188
## [1] 189
## [1] 190
## [1] 191
## [1] 192
## [1] 193
## [1] 194
## [1] 195
## [1] 196
## [1] 197
## [1] 198
## [1] 199
## [1] 200
## [1] 201
## [1] 202
## [1] 203
```

```
## [1] 204
## [1] 205
## [1] 206
## [1] 207
## [1] 208
## [1] 209
## [1] 210
## [1] 211
## [1] 212
## [1] 213
## [1] 214
## [1] 215
## [1] 216
## [1] 217
## [1] 218
## [1] 219
## [1] 220
## [1] 221
## [1] 222
## [1] 223
## [1] 224
## [1] 225
## [1] 226
## [1] 227
## [1] 228
## [1] 229
## [1] 230
## [1] 231
## [1] 232
## [1] 233
## [1] 234
## [1] 235
## [1] 236
## [1] 237
## [1] 238
## [1] 239
## [1] 240
## [1] 241
## [1] 242
## [1] 243
## [1] 244
## [1] 245
## [1] 246
## [1] 247
## [1] 248
## [1] 249
## [1] 250
```

Have a look at the results. First calculate the fraction of the number of simulations won by each algorithm in terms of CSMF Accuracy.

```
# create a matix of CSMF Accuracies - rows are simulations,
# columns are CSMF Accuracies for each algorithm the number
# of columns is the number of CSMF accuracies
csmf.mat <- matrix(unlist(csmf.list), ncol = length(csmf.list[[1]]),
```

```
    byrow = TRUE)
# name the columns of csmf.mat using the algorithm names from
# the CSMF Accuracies
colnames(csmf.mat) <- names(csmf.list[[1]])
# remove the 'reference' CSMF Accuracy
csmf.mat <- csmf.mat[, 2:ncol(csmf.mat)]
# identify the maximum CSMF Accuracy by row, i.e. which
# algorithm won
csmf.max <- apply(csmf.mat, 1, max)
# loop through the simulations, rows, and identify which
# column has the maximum CSMF Accuracy
csmf.winner <- c()
for (s in 1:length(csmf.max)) {
    csmf.winner <- c(csmf.winner, which(csmf.mat[s, ] == csmf.max[s]))
}
# create a table of the fraction of simulations won by each
# algorithm
winner.fracs <- table(csmf.winner)/length(csmf.max)
# identify the names of the winning algorithms and label the
# winner.fracs columns
names(winner.fracs) <- colnames(csmf.mat)[as.numeric(names(winner.fracs))]
# display the winning fractions as percents: percent of cross
# validation simulations won by each algorithm
100 * winner.fracs
```

```
## InterVA.top  InSilicoVA      Tariff
##          25          70           5
```

Now compare the average across all simulations of the percent correct cause assignments for each algorithm.

```
# create a matrix of zeroes corresponding to the matrix of
# percent correct cause assignments
d <- discord.list[[1]] - discord.list[[1]]
# loop over allf the simulations and cummulate the percent
# correct
for (i in 1:sims) {
    d <- d + discord.list[[i]]
}
# divide the total discordant counts by the number of
# simulations
d <- round(d/sims, 0)
# average percent correct cause cause assignments in 'sims'
# simulations
d
```

```
##            Reference NBC InterVA InSilicoVA Tariff
## Reference        100  82      82         85     73
## NBC               82 100      82         91     80
## InterVA           82  82     100         89     74
## InSilicoVA        85  91      89        100     78
## Tariff            73  80      74         78    100
```

Characterizing the Effects of SCI

Changing the SCI should have an effect on how VA algorithms assign causes. Below we conduct a heuristic exploration of this effect by creating different SCI and then using them with each algorithm to assign causes to the test deaths. We compare the results to those produced by running each algorithm on the test deaths using the correct SCI, i.e. the SCI produced from the complementary sample of the simulated deaths, above.

Start by defining SCI, simulating deaths with that SCI, and dividing the simulated deaths into 'SCI' and 'test deaths' by drawing a random sample - same as above.

```
# create de novo SCI consisting of conditional probabilities
# p(s|c): causes x 5 symptoms you can add or subtract causes,
# i.e. rows below
SCI.denovo <- list()
SCI.denovo[[1]] <- c(0.9, 0.4, 0.1, 0.01, 0.8)
SCI.denovo[[2]] <- c(0.3, 0.41, 0.8, 0.75, 0.9)
SCI.denovo[[3]] <- c(0.1, 0.01, 0.01, 0.95, 0.8)
SCI.denovo[[4]] <- c(0.05, 0.9, 0.75, 0.5, 0.2)
# define a set of CSMFs, fraction of deaths that are for each
# cause: cause1, cause2, etc.  number of deaths to simulate
sim.deaths <- 1000
csmf <- c(0.25, 0.45, 0.2, 0.1)
# sample for preparing SCI
samp.frac <- 0.5

# store SCI defined above
SCI <- storeSCI(SCI.denovo)
# simulate a bunch of deaths using the SCI and csmf defined
# above
simDeaths <- simulateDeaths(sim.deaths, csmf, SCI)
head(simDeaths)
```

```
##        cause s1 s2 s3 s4 s5
## death1     3  0  0  0  1  1
## death2     1  1  0  0  0  1
## death3     4  0  1  1  0  0
## death4     3  0  0  0  1  1
## death5     2  0  0  0  1  0
## death6     2  1  0  1  1  1
```

```
# draw sample for to prepare SCI
sample <- sample(nrow(simDeaths), samp.frac * nrow(simDeaths))
# select the sampled deaths to be used to create SCI
SCIdeaths <- simDeaths[sample, ]
# select the inverse of the sample as deaths for testing
testDeaths <- simDeaths[(-sample), ]
```

Calculate the CSMFs of the test deaths.

```
# calcualte the CSMF of the SCI and test deaths
testDeaths.CSMF <- calcCSMF(testDeaths)
```

Create SCI from the 'sample' deaths. This is the *true* SCI derived from deaths that are indistinguishable from the test deaths.

```
# calcualte SCI from the SCI deaths
SCIdeaths.SCI <- calcSCI(SCIdeaths)
```

Create a couple convenience functions to use when creating modified SCI. *logi()* calculates the logit transform of a number between 0 and 1 and effectively stretches them onto the whole real line. *expit()* is its inverse.

```
logit <- function(x) {
    log(x/(1 - x))
}
expit <- function(x) {
    exp(x)/(1 + exp(x))
}
```

Now create a modified SCI matrix. Start with the real SCI and massage it a bit, and have a look at what's happening as you go.

```
# vector of modifications that can be edited easily first
# element scales whole SCI up or down, around 1.0 the rest
# modify symptoms 2,3,5 and cause 3, below
mods <- c(1.5, -0.5, 0.9, 0.7, -0.7)

# scale the whole SCI matrix
SCI.less <- expit(logit(SCI) * mods[1])
# initialize SCI.modified with SCI.less
SCI.modified <- SCI.less

# reduce the influence of symptom 2 on logit scale
SCI.modified[, 2] <- expit(logit(SCI.less[, 2]) + mods[2])
# increase the influence of symptom 3 by 20% on logit scale
SCI.modified[, 3] <- expit(logit(SCI.less[, 3]) + mods[3])
# increase the influence of symptom 5 by 75% on logit scale
SCI.modified[, 5] <- expit(logit(SCI.less[, 5]) + mods[4])
# make all symptoms less likely for cause 3
SCI.modified[3, ] <- expit(logit(SCI.less[3, ]) + mods[5])
```

Have a look at the original SCI compared to the modified SCI.

```
SCI
```

```
##          s1   s2   s3   s4  s5
## cause1 0.90 0.40 0.10 0.01 0.8
## cause2 0.30 0.41 0.80 0.75 0.9
## cause3 0.10 0.01 0.01 0.95 0.8
## cause4 0.05 0.90 0.75 0.50 0.2
```

```
SCI.modified
```

```
##                s1           s2           s3         s4
## cause1 0.96428571 0.2482070527 0.0834907077 0.00101416
## cause2 0.21909522 0.2600039422 0.9516366751 0.83860952
## cause3 0.01805989 0.0005038743 0.0005038743 0.97626211
## cause4 0.01193046 0.9424504412 0.9274335486 0.50000000
##               s5
## cause1 0.9415547
## cause2 0.9819401
## cause3 0.7989013
## cause4 0.2010987
```

```
SCI - SCI.modified
```

```
##                 s1           s2           s3          s4
## cause1 -0.06428571  0.151792947  0.016509292  0.00898584
```

```
## cause2  0.08090478  0.149996058 -0.151636675 -0.08860952
## cause3  0.08194011  0.009496126  0.009496126 -0.02626211
## cause4  0.03806954 -0.042450441 -0.177433549  0.00000000
##                  s5
## cause1 -0.141554720
## cause2 -0.081940110
## cause3  0.001098706
## cause4 -0.001098706
```

Run all the algorithms as we have done before using 1) the true SCI, and 2) the modified SCI. Store the results of both.

```
# create a starting p(c) for all the methods, a uniform
# distribution
prob.c <- rep(1/nrow(SCIdeaths.SCI), nrow(SCIdeaths.SCI))
VAdeaths <- testDeaths[, 2:6]

# run NBC
NBC.results <- calcNBC(SCIdeaths.SCI, prob.c, VAdeaths)
NBC.results.modified <- calcNBC(SCI.modified, prob.c, VAdeaths)
# have a look at the results
topCauses <- list(NBC = NBC.results$top.cause, NBC.mod = NBC.results.modified$top.cause)
# calculate and have a look at the concordance matrices
concordance.nbc <- calcConcordance(topCauses, testDeaths)
concordance.nbc$discordant
```

```
##           Reference NBC NBC.mod
## Reference         0  78      86
## NBC              78   0      17
## NBC.mod          86  17       0
```

```
concordance.nbc$percent.correct
```

```
##           Reference   NBC NBC.mod
## Reference     100.0  84.4    82.8
## NBC            84.4 100.0    96.6
## NBC.mod        82.8  96.6   100.0
```

```
# calculate and have a look at the CSMF Accuracies
CSMFs <- list(NBC = NBC.results$csmf.topCause, NBC.mod = NBC.results.modified$csmf.topCause)
# calculate CSMF accuracies
CSMF.accuracies.nbc <- calcCSMFaccuracies(CSMFs, testDeaths.CSMF)
CSMF.accuracies.nbc$CSMF.accuracies
```

```
## reference       NBC   NBC.mod
## 1.0000000 0.8930131 0.8624454
```

```
# run InterVA
InterVA.results <- calcInterVA(SCIdeaths.SCI, prob.c, VAdeaths)
InterVA.results.modified <- calcInterVA(SCI.modified, prob.c,
    VAdeaths)
# have a look at the results
topCauses <- list(InterVA = InterVA.results$top.cause, InterVA.mod = InterVA.results.modified$top.cause
# calculate and have a look at the concordance matrices
concordance.interva <- calcConcordance(topCauses, testDeaths)
concordance.interva$discordant
```

```
##             Reference InterVA InterVA.mod
```

```
## Reference           0      99         137
## InterVA            99       0         113
## InterVA.mod       137     113           0
```

```
concordance.interva$percent.correct
```

```
##             Reference InterVA InterVA.mod
## Reference       100.0    80.2        72.6
## InterVA          80.2   100.0        77.4
## InterVA.mod      72.6    77.4       100.0
```

```
# calculate and have a look at the CSMF Accuracies
CSMFs <- list(InterVA = InterVA.results$csmf.topCause, InterVA.mod = InterVA.results.modified$csmf.topCa
# calculate CSMF accuracies
CSMF.accuracies.interva <- calcCSMFaccuracies(CSMFs, testDeaths.CSMF)
CSMF.accuracies.interva$CSMF.accuracies
```

```
##   reference     InterVA InterVA.mod
##   1.0000000   0.9465066   0.7936681
```

```
# run InSilicoVA - it take a lot longer!
InSilicoVA.results <- calcInSilicoVA(SCIdeaths.SCI, prob.c, VAdeaths,
    500, 0.05)
InSilicoVA.results.modified <- calcInSilicoVA(SCI.modified, prob.c,
    VAdeaths, 500, 0.05)
# have a look at the results
topCauses <- list(InSilicoVA = InSilicoVA.results$top.cause,
    InSilicoVA.mod = InSilicoVA.results.modified$top.cause)
# calculate and have a look at the concordance matrices
concordance.insilicova <- calcConcordance(topCauses, testDeaths)
concordance.insilicova$discordant
```

```
##                Reference InSilicoVA InSilicoVA.mod
## Reference              0         70             83
## InSilicoVA            70          0             46
## InSilicoVA.mod        83         46              0
```

```
concordance.insilicova$percent.correct
```

```
##                Reference InSilicoVA InSilicoVA.mod
## Reference          100.0       86.0           83.4
## InSilicoVA          86.0      100.0           90.8
## InSilicoVA.mod      83.4       90.8          100.0
```

```
# calculate and have a look at the CSMF Accuracies
CSMFs <- list(InSilicoVA = InSilicoVA.results$csmf.median, InSilicoVA.mod = InSilicoVA.results.modified
# calculate CSMF accuracies
CSMF.accuracies.insilicova <- calcCSMFaccuracies(CSMFs, testDeaths.CSMF)
CSMF.accuracies.insilicova$CSMF.accuracies
```

```
##      reference     InSilicoVA InSilicoVA.mod
##      1.0000000      0.9671658      0.9175117
```

```
# run Tariff
Tariff.results <- calcTariff(SCIdeaths, VAdeaths, TRUE, TRUE)
# simulate 'gold standard' deaths from the modified SCI for
# Tariff to use
SCIDeaths.modified <- simulateDeaths(sim.deaths, csmf, SCI.modified)
head(SCIDeaths.modified)
```

```
##        cause s1 s2 s3 s4 s5
## death1     2  0  1  1  1  1
## death2     3  0  0  0  1  1
## death3     4  0  0  1  0  0
## death4     3  0  0  0  1  1
## death5     2  0  0  1  1  1
## death6     3  0  0  0  1  1
```

```
# run Tariff with the deaths simulated from the modified SCI
Tariff.results.modified <- calcTariff(SCIDeaths.modified, VAdeaths,
    TRUE, TRUE)
# have a look at the results
topCauses <- list(Tariff = Tariff.results$top.cause, Tariff.mod = Tariff.results.modified$top.cause)
# calculate and have a look at the concordance matrices
concordance.tariff <- calcConcordance(topCauses, testDeaths)
concordance.tariff$discordant
```

```
##            Reference Tariff Tariff.mod
## Reference          0    174         77
## Tariff           174      0        141
## Tariff.mod        77    141          0
```

```
concordance.tariff$percent.correct
```

```
##            Reference Tariff Tariff.mod
## Reference      100.0   65.2       84.6
## Tariff          65.2  100.0       71.8
## Tariff.mod      84.6   71.8      100.0
```

```
# calculate and have a look at the CSMF Accuracies
CSMFs <- list(Tariff = Tariff.results$csmf, Tariff.mod = Tariff.results.modified$csmf)
# calculate CSMF accuracies
CSMF.accuracies.tariff <- calcCSMFaccuracies(CSMFs, testDeaths.CSMF)
CSMF.accuracies.tariff$CSMF.accuracies
```

```
##  reference     Tariff Tariff.mod
##  1.0000000  0.7641921  0.9650655
```

Review the CSMF Accuracies of each combination of SCI and algorithm, starting with NBC.

```
CSMF.accuracies.nbc$CSMF.accuracies
```

```
## reference       NBC   NBC.mod
## 1.0000000 0.8930131 0.8624454
```

InterVA.

```
CSMF.accuracies.interva$CSMF.accuracies
```

```
##   reference     InterVA InterVA.mod
##   1.0000000   0.9465066   0.7936681
```

InSilicoVA.

```
CSMF.accuracies.insilicova$CSMF.accuracies
```

```
##      reference     InSilicoVA InSilicoVA.mod
##      1.0000000      0.9671658      0.9175117
```

Tariff.

```
CSMF.accuracies.tariff$CSMF.accuracies
```

```
##  reference     Tariff Tariff.mod
##  1.0000000  0.7641921  0.9650655
```

Now have a look at how badly each algorithm's performance was degraded by using the modfied SCI. True CSMF Accuracy minus that produced using the modified SCI.

NBC.

```
CSMF.accuracies.nbc$CSMF.accuracies[2] - CSMF.accuracies.nbc$CSMF.accuracies[3]
```

```
##        NBC
## 0.03056769
```

InterVA.

```
CSMF.accuracies.interva$CSMF.accuracies[2] - CSMF.accuracies.interva$CSMF.accuracies[3]
```

```
##   InterVA
## 0.1528384
```

InSilicoVA.

```
CSMF.accuracies.insilicova$CSMF.accuracies[2] - CSMF.accuracies.insilicova$CSMF.accuracies[3]
```

```
## InSilicoVA
## 0.04965405
```

Tariff.

```
CSMF.accuracies.tariff$CSMF.accuracies[2] - CSMF.accuracies.tariff$CSMF.accuracies[3]
```

```
##     Tariff
## -0.2008734
```

Wrap it all up so that we can look at both the CSMFs and the difference between the two.

```
# NBC
nbc.test <- c(CSMF.accuracies.nbc$CSMF.accuracies[2:3], CSMF.accuracies.nbc$CSMF.accuracies[2] -
    CSMF.accuracies.nbc$CSMF.accuracies[3])
names(nbc.test) <- c(names(nbc.test)[1:2], "difference")
# InterVA
interva.test <- c(CSMF.accuracies.interva$CSMF.accuracies[2:3],
    CSMF.accuracies.interva$CSMF.accuracies[2] - CSMF.accuracies.interva$CSMF.accuracies[3])
names(interva.test) <- c(names(interva.test)[1:2], "difference")
# InSilicoVA
insilicova.test <- c(CSMF.accuracies.insilicova$CSMF.accuracies[2:3],
    CSMF.accuracies.insilicova$CSMF.accuracies[2] - CSMF.accuracies.insilicova$CSMF.accuracies[3])
names(insilicova.test) <- c(names(insilicova.test)[1:2], "difference")
# Tariff
tariff.test <- c(CSMF.accuracies.tariff$CSMF.accuracies[2:3],
    CSMF.accuracies.tariff$CSMF.accuracies[2] - CSMF.accuracies.tariff$CSMF.accuracies[3])
names(tariff.test) <- c(names(tariff.test)[1:2], "difference")
```

Display the results.

NBC.

```
nbc.test
```

```
##        NBC    NBC.mod difference
## 0.89301310 0.86244541 0.03056769
```

InterVA.

```
interva.test
```

```
##     InterVA InterVA.mod  difference
##   0.9465066   0.7936681   0.1528384
```

InSilicoVA.

```
insilicova.test
```

```
##     InSilicoVA InSilicoVA.mod     difference
##     0.96716580     0.91751174     0.04965405
```

Tariff.

```
tariff.test
```

```
##     Tariff Tariff.mod difference
##  0.7641921  0.9650655 -0.2008734
```