Tutorial For How To Conduct A Bayesian network meta-analysis

11/29/2019

Conducting a Bayesian Network Meta-Analysis:

This code is intended as a companion to the Frontiers article "How to conduct a Bayesian network meta-analysis" Hu, Wang, Sargeant, Winder and O'Connor.

Acknowledgement

The jags.bugs code used in this tutorial are directly from, or slightly modified from, the publication. Dias S, Welton NJ, Sutton AJ, et al. NICE DSU Technical Support Document 2: A Generalised Linear Modelling Framework for Pairwise and Network Meta-Analysis of Randomised Controlled Trials [Internet]. London: National Institute for Health and Care Excellence (NICE); 2014 Apr. Available from: https://www.ncbi.nlm.nih.gov/books/NBK310366/ R scripts were modified from code previously used in the following mansucripts.

Getting Started

In this code, we assume the reader knows how to load and install packages and has JAGS already install. JAGS can be install from https://sourceforge.net/projects/mcmc-jags/files/JAGS/4.x/ . For Mac users, another option is to install homebrew at the terminal by coping and pasting the following at the terminal /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install) . After installation of homebrew then install JAGS by coping and pasting the following at the terminal"homebrew brew install jags"

Loading The Required Packages

This R chuck- install_packages - installs and loads all the packages needed for the analysis. It is only nessecary to install packages once using the command "install.packages(package.name)" . However, each time the anlaysis is conducted, t is nessecary to load packages each time the code is used using the "library(package.name)".

```
#install.packages("dplyr")
#install.packages("network")
#install.packages("MASS")
#install.packages("rjags")
#install.packages("EcoSimR")
#install.packages("miceadds")
#install.packages("openxlsx")
#install.packages("usethis")
#install.packages("DT")
#install.packages("mcmcplots")
```

```r
packages <- c("dplyr", "network", "MASS", "rjags", "EcoSimR", "miceadds",
              "openxlsx", "DT", "mcmcplots")
lapply(packages, library, character.only = TRUE)
```

Determine the path saving your figures

The Data Organization

To use these scripts to conduct a network meta-analysis, two datasets are needed.

The Mapping File

The first dataset is a mapping file called "mapping and renaming TXT.xlsx" which contains the list of all treatments in the dataset and maps each treatment to a number starting from 1. This number, rather than the character-based name, is subsequently used in several scripts. When setting up the map_txt file, it is essential that the baseline treatment is labeled as 1. In our "mapping and renaming TXT.xlsx", the baseline treatment is the placebo group, therefore the placebo group has number 1. Make sure your "mapping and renaming TXT.xlsx" file has the same column names as this one because the subsequent code uses exactly these column names. The "mapping and renaming TXT.xlsx" file has 3 columns of data.

- 'name' is the full name of each treatment;
- 'abbr' is the column containing the abbreviation of each treatment. An abbreviation is often needed when creating network plots, if the full name of the treatment is too long. If this is not a concern, then 'name' and 'abbr' can contain the same values.
- 'number' is a column containing the mapping number starting from 1.

```r
map_txt <- openxlsx::read.xlsx(xlsxFile = "./data/mapping and renaming TXT.xlsx")
head(map_txt,3) #The "head: function provides the 1st three rows of R data.frame
```

```
##   abbr number        name
## 1    A      1     Placebo
## 2    B      2 Treatment_B
## 3    C      3 Treatment_C
```

The Extracted Data

Arm-Level Data

The second dataset file used contains the extracted data from the studies and is called "MTCdata.csv" file. This file contains either arm-level data or contrast-level data from the extracted studies.

The arm_level MTCdata file has 11 variables.

- study : a study number or ID.
- Number.of.Event.in.arm.1:number of events in arm 1
- Number.of.Event.in.arm.2: number of events in arm 2
- Total.number.in.arm.1 : number of in the denominator of the risk in arm 1
- Total.number.in.arm.2 : number of in the denominator of the risk in arm 1
- Total : number in the study
- Arm.1 : character indicator of the treatment name which matches "name" in map_txt

- Arm.2 : character indicator of the treatment name which matches “name” in map_txt
- Number.of.arms : interger of study arms
- Arm1 : number of the treatment which matches “number” in map_txt
- Arm2 : number of the treatment which matches “number” in map_txt

If the MTCdata file has studies with more than two arms, add corresponding column names with the same naming format. Also, make sure when you read in the data, set `stringsAsFactors` to be FALSE.

```
MTCdata <- read.csv(file = "./data/MTCdata-3arm.csv", stringsAsFactors = F)
head(MTCdata,3)
```

```
##   study Number.of.Event.in.arm.1 Number.of.Event.in.arm.2
## 1     1                       25                       17
## 2     2                       36                       32
## 3     3                       19                        7
##   Number.of.Event.in.arm.3 Total.number.in.arm.1 Total.number.in.arm.2
## 1                       20                    41                    84
## 2                       NA                    41                    84
## 3                       NA                    25                    25
##   Total.number.in.arm.3 Total Arm.1 Arm.2 Arm.3 Number.of.arms Arm1 Arm2 Arm3
## 1                   100   225     A     B     C              3    1    2    3
## 2                    NA   125     A     B  <NA>              2    1    2   NA
## 3                    NA    50     A     B  <NA>              2    1    2   NA
```

```
names(MTCdata) #The "names: function provides the names of the variables in the R data.frame
```

```
##  [1] "study"                    "Number.of.Event.in.arm.1"
##  [3] "Number.of.Event.in.arm.2" "Number.of.Event.in.arm.3"
##  [5] "Total.number.in.arm.1"    "Total.number.in.arm.2"
##  [7] "Total.number.in.arm.3"    "Total"
##  [9] "Arm.1"                    "Arm.2"
## [11] "Arm.3"                    "Number.of.arms"
## [13] "Arm1"                     "Arm2"
## [15] "Arm3"
```

```
str(MTCdata)  # The "str" functions provides the structire of the variables in the R data.frame
```

```
## 'data.frame':    26 obs. of  15 variables:
##  $ study                    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Number.of.Event.in.arm.1: int  25 36 19 20 41 122 236 23 175 57 ...
##  $ Number.of.Event.in.arm.2: int  17 32 7 5 47 69 53 15 166 20 ...
##  $ Number.of.Event.in.arm.3: int  20 NA NA NA NA NA NA NA NA NA ...
##  $ Total.number.in.arm.1   : int  41 41 25 25 50 160 402 27 281 119 ...
##  $ Total.number.in.arm.2   : int  84 84 25 50 100 314 399 52 274 118 ...
##  $ Total.number.in.arm.3   : int  100 NA NA NA NA NA NA NA NA NA ...
##  $ Total                    : int  225 125 50 75 150 474 801 79 555 237 ...
##  $ Arm.1                    : chr  "A" "A" "A" "A" ...
##  $ Arm.2                    : chr  "B" "B" "B" "B" ...
##  $ Arm.3                    : chr  "C" NA NA NA ...
##  $ Number.of.arms           : int  3 2 2 2 2 2 2 2 2 2 ...
##  $ Arm1                     : int  1 1 1 1 1 1 1 1 2 2 ...
##  $ Arm2                     : int  2 2 2 2 2 5 5 5 5 5 ...
##  $ Arm3                     : int  3 NA NA NA NA NA NA NA NA NA ...
```

```
#datatable(MTCdata, rownames = FALSE, filter="top", options = list(pageLength = 5, scrollX=T) )
```

Contrast-Level Data

It is also possible that the extracted data are contrast-level data. Contrast level describe the comparison of a pair of treatments. An example of a contrast-level data for catagorical outcomes is the odds ratio. For continous data, an example of a contrast level outcome is the mean difference. Contrast-level data are extracted because authors do not report the arm level data or if the reviewer has a preference for the contrast level data. In veterinary clincial trials a common reason for using a contrast-level effect is that the trial was conducted in a setting that required adjustment for non-independent observataions due to clustering by pen or farm.

The contrast-level MTCdata file.

- study : a study number or ID.
- Arm.1 : character indicator of the treatment name which matches "name" in map_txt
- Arm.2 : character indicator of the treatment name which matches "name" in map_txt
- Number.of.arms: interger of study arms
- lor.2 : numerical variable of the log odds of Arm 2 compared to Arm 1
- se.2 : numerical varaible of teh se. of lor.2 0.42 0.527 0.646 0.687 0.419 ...
- Arm1 : number of the treatment which matches "number" in map_txt
- Arm2 : number of the treatment which matches "number" in map_txt
- V : 'V' is the variance of the log odds of 'Arm.1'. 'V' only has values when a study has more than two arms, otherwise it is NA. That's why in this data 'V' does not have numeric values
- PLA.lo : the baseline log odds, this only has values when a study has the baseline otherwise NA.

```
MTCdata_contrast <- read.csv(file = "./data/MTCdata_contrast-3arm.csv", stringsAsFactors = F)
head(MTCdata_contrast, 3)
```

```
##   ï..study Arm.1 Arm.2 Arm.3 Number.of.arms   lor.2     lor.3   se.2   se.3
## 1        1     A     B     C              3 -1.8178 -1.832581 0.4198 0.4062
## 2        2     A     B  <NA>              2 -2.4596        NA 0.5275     NA
## 3        3     A     B  <NA>              2 -2.0971        NA 0.6463     NA
##   Arm1 Arm2 Arm3      V PLA.lo
## 1    1    2    3 0.1025 0.4463
## 2    1    2   NA     NA 1.9741
## 3    1    2   NA     NA 1.1527
```

```
names(MTCdata_contrast)
```

```
##  [1] "ï..study"       "Arm.1"          "Arm.2"          "Arm.3"
##  [5] "Number.of.arms" "lor.2"          "lor.3"          "se.2"
##  [9] "se.3"           "Arm1"           "Arm2"           "Arm3"
## [13] "V"              "PLA.lo"
```

```
str(MTCdata_contrast)
```

```
## 'data.frame':    26 obs. of  14 variables:
##  $ ï..study      : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Arm.1         : chr  "A" "A" "A" "A" ...
##  $ Arm.2         : chr  "B" "B" "B" "B" ...
##  $ Arm.3         : chr  "C" NA NA NA ...
##  $ Number.of.arms: int  3 2 2 2 2 2 2 2 2 2 ...
##  $ lor.2         : num  -1.82 -2.46 -2.1 -3.58 -1.64 ...
##  $ lor.3         : num  -1.83 NA NA NA NA ...
##  $ se.2          : num  0.42 0.527 0.646 0.687 0.419 ...
##  $ se.3          : num  0.406 NA NA NA NA ...
##  $ Arm1          : int  1 1 1 1 1 1 1 1 2 2 ...
##  $ Arm2          : int  2 2 2 2 2 5 5 5 5 5 ...
```

```
##  $ Arm3          : int  3 NA NA NA NA NA NA NA NA NA ...
##  $ V             : num  0.102 NA NA NA NA ...
##  $ PLA.lo        : num  0.446 1.974 1.153 1.386 1.516 ...
```

##Loading the R scripts for the User-Defined Functions To run the analysis, several R scripts and jags scripts needed. These scripts run the analyses, create functions used in subsequent scripts, and generate output tables and plots. With this chuck of code all the scripts are loaded and functions created. The later chucks draw on these user-defined functions to run the analyses.

After you have run the "run-all-scripts" chuck you will also see a large number of user-define R functions that have been created in the global environment. To read more about user-defined functions in R see https://www.statmethods.net/management/userfunctions.html

```
miceadds::source.all("./scripts/")
```

Common User-Defined Function Options

User-defined functions are used in scripts and some have several options exist. Here we describe some of the frequently used options and explain their functionality.

- dataType: This option describes the type of your MTCdata. Available options are "Arm" or "Contrast". This option exists in the direct_comparison.R, general_model.R, genetateMTMNetwork.R, getBaselinePrior.R, mean_ranking_matrix.R, MTC_comparison.R, network_plot.R, pairwise_comp.R
- save_res: This is a logical argument that indicates whether to save the results (the output) into your local directory. Available options are "T" or "F". This option exists in the direct_comparison.R, direct_vs_MTC.R, getBaselinePrior.R, mean_ranking.matrix.R, MTC_comparison.R, network_geometry.R, pairwise_comp.R, prob_rank_table.R
- good_event: This is an indiactor variable. Use the indicator, 1 if the events are good, 0 if the events are adverse (bad). THis option exists in general_model.R, prob_rank_table.R
- arm_name: This options selects the treatment name type to output. Available options are "abbr" or "full" This option exists in direct_vs_MTC.R, mean_ranking.matrix.R, prob_rank_table.R, ranking_plot.R

An Example Of A User-Defined Function: The *network_geometry( )* Function

Here we illustrate a user-defined function – the network_geometry. The reader can click on the function in the Global Environment to see the code. The inputs required for the code are the MTC dataset either arm-level of contrast-level. the "save_res" option is also needed. This function generates several outputs that are used later to create the network plot. The function creates the "network_geom" object which contains the following:

- The PIE index which is a metric for networks connectedness
- The c_score which is a metric of network connectedness
- A network geometry dataframe which containing the study ID and treatment arms.

- A network co-occurance matrix

The network_geom object will appear in the "data" section of the Global Environment. The datasets network_cooc_mat.csv and Network_geiometry.csv are also output into the data folder.

```
network_geom <- network_geometry(MTCdata = MTCdata, save_res = T)
```

```
## Time Stamp:  Tue Dec 10 12:58:05 2019
## Reproducible:
## Number of Replications:
## Elapsed Time:  0.52 secs
## Metric:  c_score
```

```
## Algorithm:  sim9
## Observed Index:  43.2
## Mean Of Simulated Index:  42.312
## Variance Of Simulated Index:  0.68599
## Lower 95% (1-tail):  41.2
## Upper 95% (1-tail):  43.8
## Lower 95% (2-tail):  41.2
## Upper 95% (2-tail):  44.2
## Lower-tail P =  0.891
## Upper-tail P =  0.148
## Observed metric > 852 simulated metrics
## Observed metric < 109 simulated metrics
## Observed metric = 39 simulated metrics
## Standardized Effect Size (SES):  1.0724
```

```
network_geom$PIE
```

```
## $PIE
## [1] 0.7532656
##
## $PIE_max
## [1] 0.8153846
```

```
network_geom$c_score
```

```
## [1] 43.2
```

```
network_geom$network_df
```

```
##    study treatment
## 1      1         A
## 27     1         B
## 53     1         C
## 2      2         A
## 28     2         B
## 3      3         A
## 29     3         B
## 4      4         A
## 30     4         B
## 5      5         A
## 31     5         B
## 6      6         A
## 32     6         E
## 7      7         A
## 33     7         E
## 8      8         A
## 34     8         E
## 9      9         B
## 35     9         E
## 10    10         B
## 36    10         E
## 11    11         B
## 37    11         E
## 12    12         B
## 38    12         E
## 13    13         B
```

```
## 39    13         E
## 14    14         A
## 40    14         B
## 15    15         A
## 41    15         C
## 16    16         A
## 42    16         C
## 17    17         A
## 43    17         C
## 18    18         A
## 44    18         C
## 19    19         A
## 45    19         C
## 20    20         A
## 46    20         C
## 21    21         A
## 47    21         C
## 22    22         A
## 48    22         C
## 23    23         A
## 49    23         B
## 24    24         A
## 50    24         D
## 25    25         C
## 51    25         E
## 26    26         B
## 52    26         C
```

```
network_geom$network_cooc_mat
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]    1    1    1    1    1    1    1    1    0     0     0     0     0     1
## [2,]    1    1    1    1    1    0    0    0    1     1     1     1     1     1
## [3,]    1    0    0    0    0    0    0    0    0     0     0     0     0     0
## [4,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0
## [5,]    0    0    0    0    0    1    1    1    1     1     1     1     1     0
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
## [1,]     1     1     1     1     1     1     1     1     1     1     0     0
## [2,]     0     0     0     0     0     0     0     0     1     0     0     1
## [3,]     1     1     1     1     1     1     1     1     0     0     1     1
## [4,]     0     0     0     0     0     0     0     0     0     1     0     0
## [5,]     0     0     0     0     0     0     0     0     0     0     1     0
```

```
#datatable(network_geom$network_df, rownames = FALSE, filter="top", options = list(pageLength = 5, scro
```

Baseline effects model - the log odds of the event

Using the getBaselinePrior() function

In our workflow of conducting a network meta-anlaysis as described in the Frontiers mansucripts one of the first steps is to create the distribtion of the baseline treatment on the log odds scale. This is achieved with the getBaselinePrior.R function. This functions determines the distribution parameter value for the baseline treatment on the logit scale by seperate mcmc simulation.

The “getBaselinePrior.R” requires the following inputs:

- The dataset to be used – In our example this is the MTCdata
- The option for dataType- Arm or Contrast– In our example - Arm

Below are defaults that are rarely changed, but can be modified in the "getBaselinePrior.R" script in the scripts folder.

- hyperSDInUnif: The prior distribution for heterogeneity parameter sigma is Unif(0, hyperSDInUnif). Default is 5.
- hyperVarInNormMean: The prior distribution for the log odds is N(0, 1/hyperVarInNormMean). Default is 1e-4
- parameters for running MCMC: i.e. n.adapt (default value = 5000), n.iter (default value = 10000), n.chains (default value = 3), n.thin (default value = 1)
- n.burnin_prop: The proportion you want to do burning for a chain. Default is 0.5

The "getBaselinePrior.R" returns the following output if save_res = T

*A dataframe containing the baseline summary statistics called baselinePriorSmry generated from getBaselinePrior()

```
baselinePriorSmry <- getBaselinePrior(MTCdata = MTCdata, dataType = "Arm")
```

```
baselinePriorSmry
```

```
##             Mean         SD       2.5%       50%     97.5%
## R      0.7249094 0.03440391  0.6554285 0.7251168 0.7913867
## R.new  0.7076054 0.13120399  0.4051256 0.7237437 0.9165190
## m      0.9757874 0.17461958  0.6429875 0.9699866 1.3333046
## mu.new 0.9739077 0.69204733 -0.3841531 0.9631081 2.3959636
## sd.m   0.6594163 0.15308383  0.4199804 0.6392035 1.0146902
## tau.m  2.6729390 1.22735262  0.9712545 2.4474941 5.6694624
```

The Workflow Of A Network Meta-analysis

Step 1

Use the comparative effects model and a Markov chain Monte Carlo (MCMC) process to obtain the posterior distributions of the log odds ratios for the basic parameters. From those basic parameters, obtain the posterior distributions of the functional parameters. Ensure convergence by evaluating the trace plots and convergence criteria.

Using the general_model() function

This function runs the comparative effects model as described in the manuscript. The general model here is a random effects model.

The "general_model.R" script requires the following inputs:

- The dataset to be used – In our example this is the MTCdata
- The option for dataType- Arm or Contrast– In our example - Arm
- The option for good_event: Indicator, 1 if the events are good, 0 if the events are bad. If the the event in your dataset is bad event, then you can set good_event to be 0, otherwise 1. This option will affect the ranking of the treatments, but not the calculation of risks or functions of risks.

Defaults that are rarely changed, but can be modified in the "getBaselinePrior.R" script in the scripts folder.

- hyperSDInUnif: The prior distribution for heterogeneity parameter sigma is Unif(0, hyperSDInUnif). Default is 5.
- hyperVarInNormMean: The prior distribution for the log odds is N(0, 1/hyperVarInNormMean). Default is 1e-4
- parameters for running MCMC: i.e. n.adapt (default value = 5000), n.iter (default value = 10000), n.chains (default value = 3), n.thin (default value = 1)
- n.burnin_prop: The proportion you want to do burning for a chain. Default is 0.5

The "general_model.R" script returns the following outputs:

- A dataframe in the Global Environment
- A dataset of the general model summary stored in the data folder called general_model_smry.csv
- the mcmc simulation: an RData object stored in the RDataObjects folder called generalModeljagsOutput.Rdata
- the mcmc simulation summary: an RData object stored in the RDataObjects folder called generalModelSummaryt.Rdata
- model deviance information: generalModeldev_result.RData stored in the RDataObjects folder

The general model summary file uses several abbreviations as follows:

- RR[#N,#D]: The risk ratio of all possible comparisons with the numerator treatment (#N) divided by the denominator treatment (#D) .
- RRa[#N]: the risk ratio of the numerator treatment (#N) divided by the baseline treatment
- T[#] : The absolute risk
- best[#]: The probability of being the best
- d[#D]: The logs odds ratio for the basic parameters with the baseline as the denominator

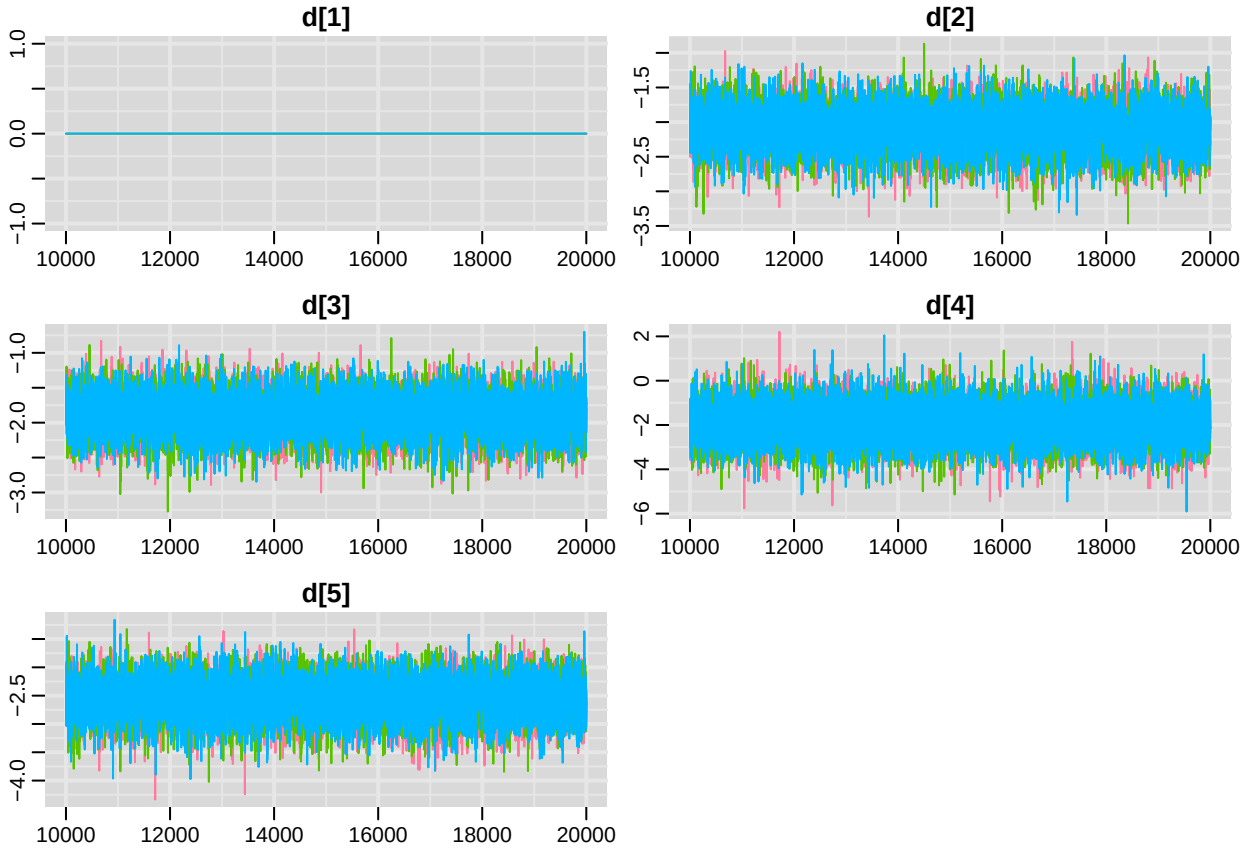
- lor[#N,#D]: The log odds ratio for basic and funcational parameters with the numerator treatment (#N) divided by the denominator treatment (#D).
- or[#N,#D]: The odds ratio of all possible comparisons with the numerator treatment (#N) divided by the denominator treatment (#D)
- resdev[#]: The residual deviance of the study
- rk[#]: The rankings for the treatments

```
jags_output <- general_model(MTCdata = MTCdata, baselinePriorSmry = baselinePriorSmry,
                             good_event = 0, dataType = "Arm")

general_model_smry <- jags_output$general_model_smry
jags.out <- jags_output$jags.out
head(general_model_smry)
tail(general_model_smry)
```

Ensure convergence of the general model, below is a chunk to check the convergence of basic parameters. Note d[1] is the log odds ratio of placebo to placebo, thereforem, it is always 0.

```
traplot(jags.out, "d")
```



Check the goodness of fit using the residual deviance.

The next step of the workflow is to check the goodness of the model's fit using the (residual) deviance. This is obtained from the general_model output.

If the model provides a good fit, then the value of residual deviance should be close to the number of data points. In this example data, we have 25 two-arm study and 1 three-arm study. The residual deviance should be close to 2*25+3 = 53.

```
general_model_smry["totresdev", 1]
```

```
## [1] 52.81857
```

Step 2: The Direct Pairwise Comparative Effects Model

Assessing the consistency assumption requires a pairwise comparative effects model based only on direct estimates. The pairwise_comp() function rearranges the dataset to identify all possible pairwise comparisons. The MTC dataset has data arranged as a row per study, while the output of this function has each pairwise comparison as a unique row. For datasets with only two arms studies, this makes no change to the dataset. However, for datasets with studies which have 3 or more arms, the arrangement is different. The direct_comparison() function conducts the analysis only for pairwise comparison dataset, i.e. no indirected comparisons.

Using the pairwise_comp() function

This code creates a dataset of the pairwise comparison of the direct effects only.

The "pairwise_comp.R" script requires the followinginputs: * The dataset to be used – In our example this is the MTCdata
* The option for dataType- Arm or Contrast– In our example - Arm * The option for save_res are T or F to save the output into your local directory– In our example - T

The "pairwise_comp.R" script returns: * A dataframe containing the pariwise comparison in the Global Environment * A dataset called pairwise_comparison.csv in the data folder if save_res = true

```
pairwise <- pairwise_comp(MTCdata = MTCdata, dataType = "Arm", save_res = T)
head(pairwise, 10)
```

```
##    Number.of.Event.in.arm.1 Number.of.Event.in.arm.2 Total.number.in.arm.1
## 2                        36                       32                    41
## 3                        19                        7                    25
## 4                        20                        5                    25
## 5                        41                       47                    50
## 6                       122                       69                   160
## 7                       236                       53                   402
## 8                        23                       15                    27
## 14                       42                       15                    50
## 15                       64                       34                   154
## 16                       34                       15                    53
##    Total.number.in.arm.2 Arm.1 Arm.2 Arm1 Arm2 rms Comparison
## 2                     84     A     B    1    2   2       2vs1
## 3                     25     A     B    1    2   2       2vs1
## 4                     50     A     B    1    2   2       2vs1
## 5                    100     A     B    1    2   2       2vs1
## 6                    314     A     E    1    5   2       5vs1
## 7                    399     A     E    1    5   2       5vs1
## 8                     52     A     E    1    5   2       5vs1
## 14                   100     A     B    1    2   2       2vs1
## 15                   154     A     C    1    3   2       3vs1
## 16                   106     A     C    1    3   2       3vs1
```

Using the direct_comparisons() function

This function conducts the analysis on the direct comparison only.

The "direct_comparisons.R" script requires the following inputs:

- The dataset to be used –the pairwise dataset: pairwise comparison generated from pairwise_comp()
- The option for dataType- Arm or Contrast– In our example - Arm
- The option for save_res are T or F to save the output into your local directory– In our example - T

Defaults that are rarely changed, but can be modified in the "direct_comparison.R" script in the scripts folder, include:

- hyperSDInUnif: The prior distribution for heterogeneity parameter sigma is Unif(0, hyperSDInUnif). Default is 2.
- hyperVarInNormMean: The prior distribution for the log odds is N(0, 1/hyperVarInNormMean). Default is 1e-4
- parameters for running MCMC: i.e. n.adapt (default value = 5000), n.iter (default value = 10000), n.chains (default value = 3), n.thin (default value = 1)

- n.burnin_prop: The proportion you want to do burn-in for a chain. Default is 0.5

The "direct_comparisons.R" script returns:

- a dataset called direct which contains the direct log OR

```
direct_comp <- direct_comparison(pairwise = pairwise, baselinePriorSmry = baselinePriorSmry, dataType =

head(direct_comp, 3)
```

##Step 3; Assessing Consistency.

Assess the consistency assumption for the treatment comparisons for which there is direct evidence. This is done by subtracting the mean estimated log odds ratios obtained from the posterior distributions of the pairwise meta-analyses from the mean estimated log odds ratios obtained from the posterior distributions of the network meta-analysis and looking for inconsistencies.

Using the direct_vs_MTC() function

This function generates data for a table commonly used in publications about network meta-analysis. The tables contains the direct comparison, MTC, and the indirect comparison on the log odds ratio scale.

The "direct_vs_MTC.R" script requires the following inputs:

- The datasets to be used = map_txt, general_model_sumy, direct_comp
- The option for save_res are T or F to save the output into your local directory– In our example - T
- The option for dataType- Arm or Contrast– In our example - Arm
- The option for arm_name which selects the treatment name type to output. Available options are "abbr" or "full" In our example - "abbr"

The "direct_vs_MTC.R" script returns:

- A dataframe containing the comparison of direct comparison, MTC, and only indirect comparison
- Also save csv file ResultOfIndicretAnddirectComparison.csv if save_res is true

```
direct_vs_MTC_matrix <- direct_vs_MTC(map_txt = map_txt, model_smry = general_model_smry,
                                      direct_compr = direct_comp, arm_name = "abbr", save_res = T)
head(direct_vs_MTC_matrix, 3)
```

```
##   Comparison d_dir sd_dir d_MTC sd_MTC d_rest sd_rest     w sd_w z_score
## 1     B vs A -2.30   0.44 -2.10   0.27  -1.98    0.35 -0.32 0.56   -0.58
## 2     E vs A -2.39   0.42 -2.58   0.32  -2.83    0.48  0.43 0.64    0.68
## 3     C vs A -1.82   0.37 -1.89   0.25  -1.95    0.35  0.12 0.51    0.24
##   p_value
## 1    0.56
## 2    0.50
## 3    0.81
```

If you prefer to show the full name of treatments in this table, you can change the arm_name option.

```
direct_vs_MTC_matrix <- direct_vs_MTC(map_txt = map_txt, model_smry = general_model_smry,
                                      direct_compr = direct_comp, arm_name = "full", save_res = T)
head(direct_vs_MTC_matrix, 3)
```

```
##               Comparison d_dir sd_dir d_MTC sd_MTC d_rest sd_rest     w sd_w
## 1 Treatment_B vs Placebo -2.30   0.44 -2.10   0.27  -1.98    0.35 -0.32 0.56
## 2 Treatment_E vs Placebo -2.39   0.42 -2.58   0.32  -2.83    0.48  0.43 0.64
## 3 Treatment_C vs Placebo -1.82   0.37 -1.89   0.25  -1.95    0.35  0.12 0.51
##   z_score p_value
```

```
## 1   -0.58    0.56
## 2    0.68    0.50
## 3    0.24    0.81
```

Generating Output Tables

Having conducted the analysis and assessed convergence and consistency, the next step is to output the results.

Using the MTC_comparison() function.

The "MTC_comparison.R" script requires the following inputs: * The datasets to be used = map_txt, general_model_sumy, direct_comp * The option for measure: the effect measure users want to compare, options are "RR" for risk ratio, "OR" for odds ratio, "LOR" for log odds ratio. * The option for stat: the statistics users want to use for comparison, options are "mean" and "median" * The option for num_digits: the number of decimal places in the output * The option for save_res are T or F to save the output into your local directory– In our example - T

The "direct_vs_MTC.R" script returns:

- A matrix with upper triangle to be mean or median, lower triangle to be 95% credible interval
- If save_res = T, an csv file: MTC_comparison_risk_ratio_all_treat.csv will be saved in ./data folder.

For example, if you want to show the log odds ratios (LOR) 95% credible interval and the mean of LOR:

```
MTC_comp_LOR <- MTC_comparison(map_txt = map_txt, model_smry = general_model_smry, measure = "LOR",
                               stat = "mean", save_res = T)
MTC_comp_LOR
```

```
##      [,1]              [,2]              [,3]              [,4]
## [1,] "E"               "-0.648"          "-0.689"          "-0.475"
## [2,] "(-2.304_0.983)"  "D"               "-0.041"          "0.174"
## [3,] "(-1.394_0.017)"  "(-1.646_1.559)"  "C"               "0.214"
## [4,] "(-1.058_0.108)"  "(-1.421_1.797)"  "(-0.422_0.85)"   "B"
## [5,] "(-3.208_-1.969)" "(-3.451_-0.415)" "(-2.404_-1.398)" "(-2.653_-1.577)"
##      [,5]
## [1,] "-2.576"
## [2,] "-1.928"
## [3,] "-1.887"
## [4,] "-2.101"
## [5,] "A"
```

For example, the cell [1,2] is the posterior mean of the log odds ratio of treatment A to treatment B, and cell [2,1] is the corresponding 95% credible interval.

If you want to show the mean odds ratios and their 95% credible interval:

```
MTC_comp_OR <- MTC_comparison(map_txt = map_txt, model_smry = general_model_smry, measure = "OR",
                               stat = "mean", save_res = T)
MTC_comp_OR
```

```
##      [,1]              [,2]              [,3]              [,4]          [,5]
## [1,] "E"               "0.743"           "0.535"           "0.65"        "0.08"
## [2,] "(0.1_2.672)"     "D"               "1.347"           "1.678"       "0.196"
## [3,] "(0.248_1.017)"   "(0.193_4.753)"   "C"               "1.305"       "0.157"
## [4,] "(0.347_1.114)"   "(0.241_6.033)"   "(0.656_2.341)"   "B"           "0.127"
```

```
## [5,] "(0.04_0.14)"   "(0.032_0.66)"  "(0.09_0.247)"  "(0.07_0.207)" "A"
```

As for mean risk ratios and their 95% credible interval,

```
MTC_comp_RR <- MTC_comparison(map_txt = map_txt, model_smry = general_model_smry, measure = "RR",
                              stat = "mean", save_res = T)
MTC_comp_RR
```

```
##      [,1]            [,2]            [,3]            [,4]            [,5]
## [1,] "E"             "0.781"         "0.616"         "0.711"         "0.252"
## [2,] "(0.208_2.309)" "D"             "1.074"         "1.26"          "0.423"
## [3,] "(0.326_1.012)" "(0.263_2.543)" "C"             "1.2"           "0.411"
## [4,] "(0.422_1.083)" "(0.31_3.059)"  "(0.736_1.894)" "B"             "0.356"
## [5,] "(0.102_0.496)" "(0.094_0.9)"   "(0.205_0.675)" "(0.168_0.621)" "A"
```

Treatments rank summary

Using the mean_rank_matrix () function

Generate mean ranking matrix

```
mean_rank <- mean_rank <- mean_rank_matrix(map_txt = map_txt, model_smry = general_model_smry,
                                           arm_name = "abbr", save_res = T)
mean_rank
```

```
##   trtNm Mean   SD 2.5% 50% 97.5%
## 1     A 4.99 0.09    5   5     5
## 2     C 3.25 0.74    2   3     4
## 3     D 2.86 1.21    1   3     4
## 4     B 2.60 0.75    1   3     4
## 5     E 1.29 0.54    1   1     3
```

Probability rank table

Generate a list containing two dataframes. One is the probability matrix that one treatment is better than the other based on posterior samples of treatment risk (probability). The other one summarizes the probability for each treatment to be the best.

```
load("./scripts/RDataObjects/generalModelJagasOutput.RData")
prob_rank <- prob_rank_table(jags.out = jags.out, map_txt = map_txt, good_event = 0,
                             arm_name = "abbr",save_res = T)
prob_rank
```

```
## $prob_comp
##      [,1]    [,2]    [,3]    [,4]    [,5]
## [1,] "A"     "0"     "0"     "0.008" "0"
## [2,] "1"     "B"     "0.757" "0.587" "0.052"
## [3,] "1"     "0.243" "C"     "0.476" "0.028"
## [4,] "0.992" "0.413" "0.524" "D"     "0.206"
## [5,] "1"     "0.948" "0.972" "0.794" "E"
##
## $rank_smry
##   Name Rank_1_prob Rank_least_prob
## 1    A       0.000           0.992
## 2    B       0.033           0.000
```

```
## 3    C       0.015           0.000
## 4    D       0.201           0.008
## 5    E       0.751           0.000
```

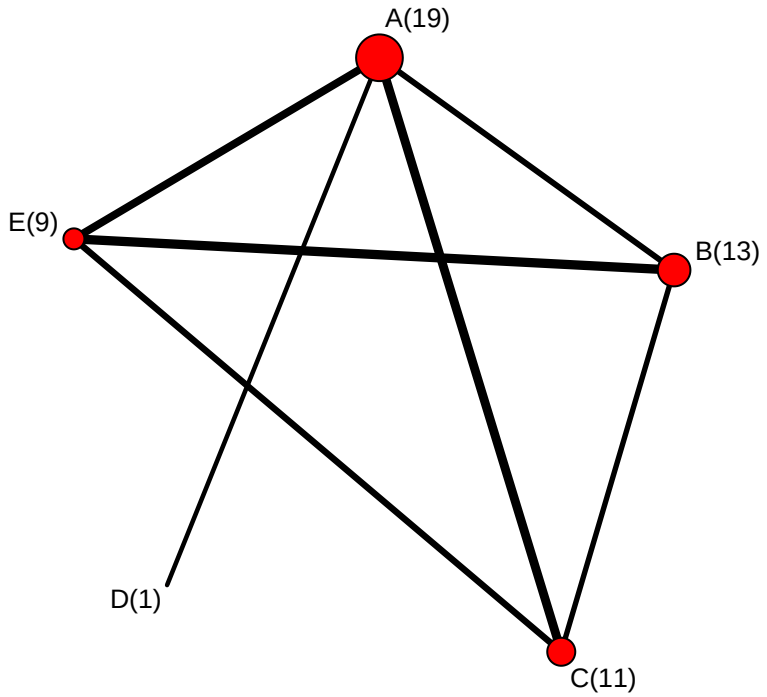
For example, the cell [1,2] in 'prob_comp' element is the probability that Treatment A is better than Treatment B. cell [2,1] is the verse. Column 'Rank_1_prob' in 'rank_smry' component shows the probability that the treatment is ranked top 1.

Common Plots Used Reported for Network Meta-Analysis

Network plot: Using the network_plot() function

The network_plot function generates the plot for the network of trials. You can set the node text size and node size and add plot title in the R script. The node size of each treatment reflects the number of studies containing that treatment. The edge width reflects the studies size of each comparison.

```
network_plot(MTCdata = MTCdata, map_txt = map_txt, dataType = "Arm")
```



Probability distribution plot: Using the network_plot() function

This function enerates the plot for the distribution of probability of retreatmeng for each treatment in the MTC

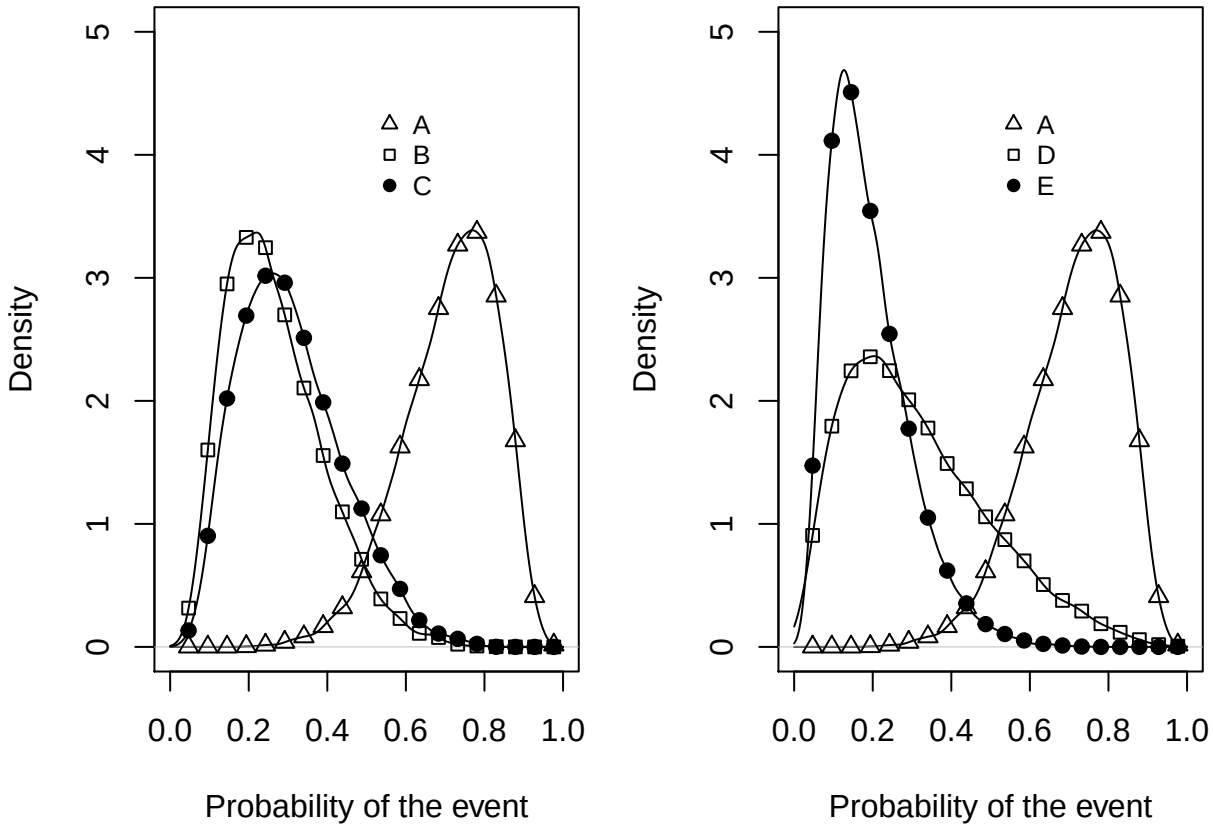
The input for the **network_plot** function are:

- layout: the layout of your figure, i.e c(1,2), c(2,2)

- num_treat_per_plot: the number of treatment shown in each plot
- treat_interested: the group of treatments for which you want to plot probability distribution. The default is "all", you can also input a vector of treatment numbers to specify the treatments you want to plot.
- legend_pos: the position (coordinates) where you want to add your legend
- other ploting parameters: i.e. ylim, main, ylab, xlab, etc

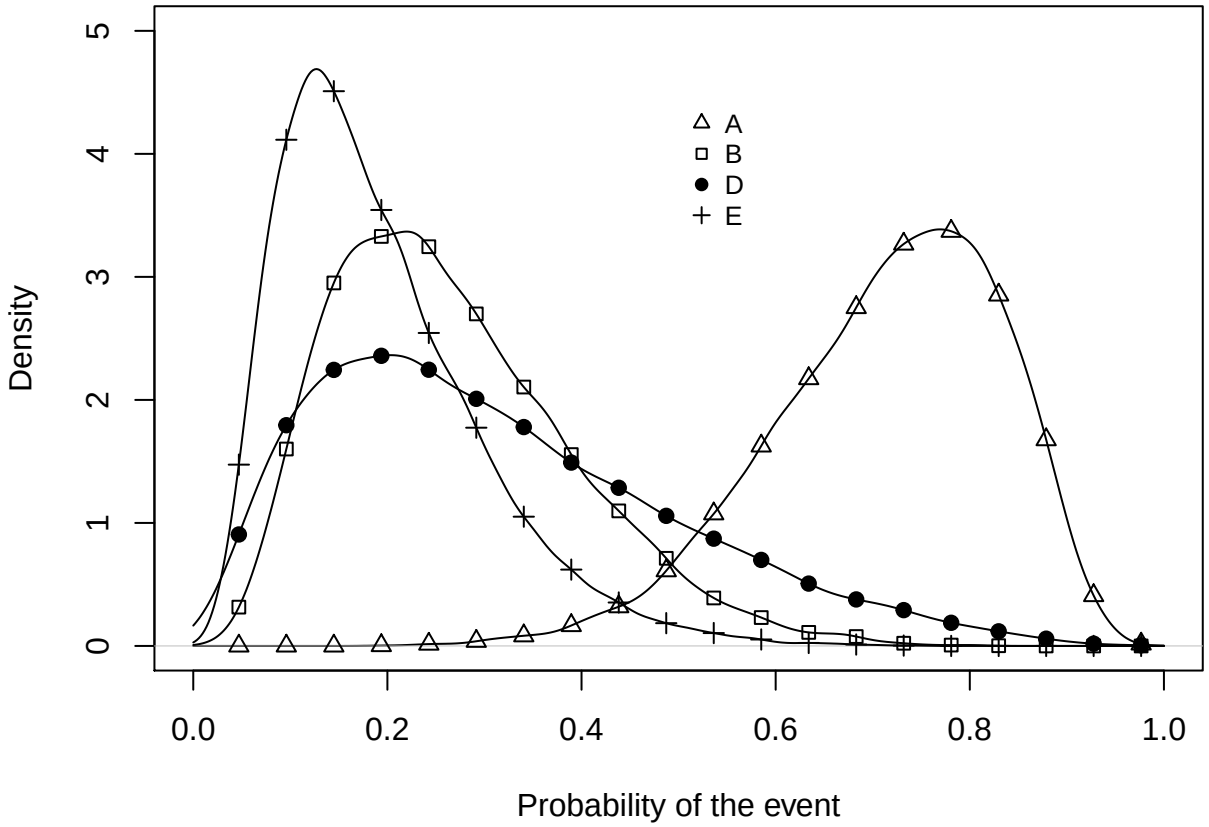
Note that the placebo risk distribution is plotted on each plot. Suppose you want to plot all treatments (five in total) risk distribution plot in one figure with two side by side plots with each one containing three treatments. The 'layout' argument should be set as c(1,2) and 'num_treat_per_plot' = 3.

```
load("./scripts/RDataObjects/generalModelJagasOutput.RData")
dist_prob_treat(jags.out = jags.out, map_txt = map_txt, arm_name = "abbr", layout = c(1,2),
                num_treat_per_plot = 3)
```



If you would like to plot only treatment A, B, D, E in a single plot, then

```
dist_prob_treat(jags.out = jags.out, map_txt = map_txt, arm_name = "abbr", layout = c(1,1),
                num_treat_per_plot = 4, treat_interested = c(1, 2, 4, 5))
```



Note that the treatment numbers should match with the mapping in map_txt.

If your layout and the number of treatment per plot couldn't plot all the treatment distributions you are interested in, then a message will pop up and ask you to change the setting.

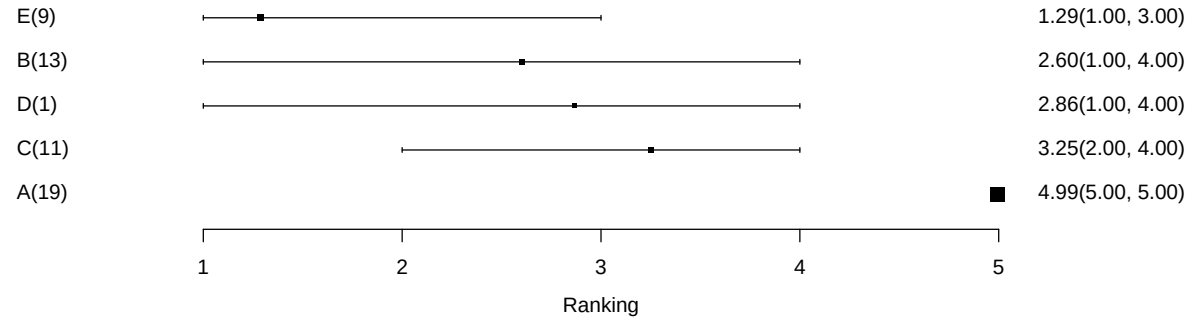
```
dist_prob_treat(jags.out = jags.out, map_txt = map_txt, arm_name = "abbr", layout = c(1,2),
                num_treat_per_plot = 2, treat_interested = "all")
```

```
## Some treatments cannot show up, please change the settings, either "layout" or "num_treat_per_plot"
```

Ranking plot: Using the ranking_plot() function

Generate the ranking plot

```
ranking_plot(MTCdata = MTCdata, map_txt = map_txt, model_smry = general_model_smry,
             arm_name = "abbr", cex = 1)
```



Note that the black dots are the mean ranking of each treatment and its size reflects the precision (1/variance) of the estimates.