

Package ‘netie’

July 22, 2025

Type Package

Title Antigen T Cell Interaction Estimation

Version 1.0

Date 2021-9-28

Author Tianshi Lu

Maintainer Tianshi Lu <tianshi.lu@utsouthwestern.edu>

Description The Bayesian hierarchical model named antigen-T cell interaction estimation is to estimate the history of the immune pressure on the evolution of the tumor clones. The model is based on the estimation result from Andrew Roth (2014) <[doi:10.1038/nmeth.2883](https://doi.org/10.1038/nmeth.2883)>.

Depends R (>= 3.6.0)

License Apache License

NeedsCompilation no

Repository CRAN

Date/Publication 2021-09-29 08:20:05 UTC

Contents

netie-package	1
input_data	3
netie	3

Index	11
--------------	-----------

netie-package	<i>Antigen T Cell Interaction Estimation</i>
---------------	--

Description

The Bayesian hierarchical model named antigen-T cell interaction estimation is to estimate the history of the immune pressure on the evolution of the tumor clones. The model is based on the estimation result from Andrew Roth (2014) <[doi:10.1038/nmeth.2883](https://doi.org/10.1038/nmeth.2883)>.

Details

The DESCRIPTION file:

```
Package:      netie
Type:         Package
Title:        Antigen T Cell Interaction Estimation
Version:      1.0
Date:         2021-9-28
Author:       Tianshi Lu
Maintainer:   Tianshi Lu <tianshi.lu@utsouthwestern.edu>
Description:  The Bayesian hierarchical model named antigen-T cell interaction estimation is to estimate the history of the in
Depends:      R (>= 3.6.0)
License:      Apache License
```

Index of help topics:

```
input_data      input_data
netie           Neoantigen-T cell interaction estimation
netie-package   Antigen T Cell Interaction Estimation
```

~~ An overview of how to use the package, including the most important functions ~~ netie(input_data,sigma_square = 100000,alpha = 10,beta = 2,sigma_p_sqr = 0.1,sigma_a_sqr = NULL,max_iter = 1000,multi_sample = T) Please refer to <https://github.com/tianshilu/Netie> for more details.

Author(s)

Tianshi Lu

Maintainer: Tianshi Lu <tianshi.lu@utsouthwestern.edu>

References

<https://github.com/tianshilu/Netie>

Examples

```
data(input_data)
netie(input_data,sigma_square=100000,alpha=10,beta=2,
sigma_p_sqr=0.1,max_iter=1000,multi_sample=TRUE)
```

input_data	<i>input_data</i>
------------	-------------------

Description

one kidney cancer patient with 2 samples

Usage

```
data("input_data")
```

Format

A data frame with 297 observations on the following 7 variables.

- mutation_id a character vector
- sample_id a character vector
- cluster_id a numeric vector
- cellular_prevalence a numeric vector
- cellular_prevalence_std a numeric vector
- variant_allele_frequency a numeric vector
- neo_load a numeric vector

Examples

```
data(input_data)
## maybe str(input_data) ; plot(input_data) ...
```

netie	<i>Neoantigen-T cell interaction estimation</i>
-------	---

Description

The Bayesian Hierarchical Model named Neoantigen-T cell interaction estimation (Netie) is to estimate the history of the immune pressure on the evolution of the tumor clones.

Usage

```
netie(input_one_patient,sigma_square,alpha,beta,
sigma_p_sqr,sigma_a_sqr,max_iter,multi_sample)
```

Arguments

<code>input_one_patient</code>	a list with each data frame as the data for each patient. Each data frame consists 7 columns and each row is for one mutation. Please refer to https://github.com/tianshilu/Netie for more details.
<code>sigma_square</code>	hyperparameters for prior distributions. Please refer to https://github.com/tianshilu/Netie for more details.
<code>alpha</code>	hyperparameters for prior distributions. Please refer to https://github.com/tianshilu/Netie for more details.
<code>beta</code>	hyperparameters for prior distributions. Please refer to https://github.com/tianshilu/Netie for more details.
<code>sigma_p_sqr</code>	hyperparameters for prior distributions. Please refer to https://github.com/tianshilu/Netie for more details.
<code>sigma_a_sqr</code>	hyperparameters for prior distributions. Please refer to https://github.com/tianshilu/Netie for more details.
<code>max_iter</code>	the iterations of Markov chain Monte Carlo.
<code>multi_sample</code>	use True if one patient has more than one sample.

Value

The output is a list with the information of the anti-tumor selection pressure for each clone `ac` and for the whole tumor `a`.

Author(s)

Tianshi Lu

References

<https://github.com/tianshilu/Netie>

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==> Define data, use random,
##--or do help(data=index) for the standard data sets.
data(input_data)
netie(input_data,sigma_square=100000,alpha=10,beta=2,
sigma_p_sqr=0.1,max_iter=1000,multi_sample=TRUE)
## The function is currently defined as
function (input_one_patient, sigma_square, alpha, beta, sigma_p_sqr,
          sigma_a_sqr, max_iter, multi_sample = FALSE)
{
  if (all(input_one_patient$neo_load[!is.na(input_one_patient$cluster_id)] ==
    0)) {
    return(NA)
  }
  input_one_patient = input_one_patient[!is.na(input_one_patient$cluster_id),
```

```

]
if (multi_sample == T) {
  mutations = unlist(sapply(input_one_patient$mutation_id,
    function(x) paste(strsplit(x, " ")[[1]][2], strsplit(x,
      " ")[[1]][3])))
  input_one_patient$neo_load = unlist(sapply(mutations,
    function(x) max(input_one_patient[mutations == x,
      "neo_load"])))
  phi = "1"
  clones = list()
  clones[[id = "1"]] = mutations[paste(input_one_patient$sample_id,
    input_one_patient$cluster_id) == paste(input_one_patient$sample_id,
    input_one_patient$cluster_id)[1]]
  for (each_clone in unique(paste(input_one_patient$sample_id,
    input_one_patient$cluster_id))[-1]) {
    mutations_one_clone = mutations[paste(input_one_patient$sample_id,
      input_one_patient$cluster_id) == each_clone]
    phi_tmp = unlist(sapply(1:length(clones), function(x) {
      uniq_clone = clones[[x]]
      shared_mutations = intersect(uniq_clone, mutations_one_clone)
      if (length(shared_mutations)/length(uniq_clone) >
        0.5 & length(shared_mutations)/length(mutations_one_clone) >
        0.5) {
        return(names(clones)[x])
      }
    })), use.names = FALSE)
    if (!is.null(phi_tmp)) {
      phi = c(phi, phi_tmp)
    }
    else {
      phi_tmp = max(as.numeric(names(clones))) + 1
      phi = c(phi, phi_tmp)
      clones[[id = as.character(phi_tmp)]] = mutations_one_clone
    }
  }
  names(phi) = unique(paste(input_one_patient$sample_id,
    input_one_patient$cluster_id))
}
if (length(unique(input_one_patient$cluster_id)) > 1) {
  if (is.null(sigma_a_sqr)) {
    non_zero_neo_avg = sapply(unique(input_one_patient$cluster_id),
      function(x) mean(input_one_patient[input_one_patient$cluster_id ==
        x & input_one_patient$neo_load != 0, "neo_load"]))
    non_zero_neo_avg[is.nan(non_zero_neo_avg)] = 0
    sigma_a_sqr = sd(log(non_zero_neo_avg + 1))^2 * 10
    if (sigma_a_sqr == 0) {
      sigma_a_sqr = 1
    }
  }
}
else {
  sigma_a_sqr = 1
}
}

```

```

if (sigma_square < 100 * sigma_a_sqr) {
  print("sigma square should be much more larger than sigma a square!")
  stop()
}
if (alpha <= beta) {
  print("alpha should be larger than beta!")
  stop()
}
if (multi_sample == T) {
  max_vaf = unlist(sapply(mutations, function(x) max(input_one_patient[mutations ==
    x, "variant_allele_frequency"])))
  input_one_patient = input_one_patient[max_vaf > 0.05,
    ]
}
else {
  input_one_patient = input_one_patient[input_one_patient$variant_allele_frequency >
    0.05, ]
}
tmp = table(input_one_patient$cluster_id[input_one_patient$neo_load >
  0])
tmp = names(tmp[tmp >= 1])
input_one_patient = input_one_patient[input_one_patient$cluster_id %in%
  tmp, ]
tmp = table(input_one_patient$cluster_id)
tmp = names(tmp[tmp >= 2])
input_one_patient = input_one_patient[input_one_patient$cluster_id %in%
  tmp, ]
if (dim(input_one_patient)[1] == 0) {
  return(NA)
}
if (multi_sample == T) {
  input_one_patient$phi = as.numeric(phi[paste(input_one_patient$sample_id,
    input_one_patient$cluster_id)])
  input_one_patient$cluster_id = as.numeric(factor(paste(input_one_patient$sample_id,
    input_one_patient$cluster_id)))
  phi_cluster = input_one_patient[, c("cluster_id", "phi")]
  phi_cluster = phi_cluster[!duplicated(phi_cluster$cluster_id),
    ]
  rownames(phi_cluster) = as.character(phi_cluster$cluster_id)
  phi_cluster = phi_cluster[as.character(unique(input_one_patient$cluster_id)),
    ]
}
else {
  input_one_patient$cluster_id = as.numeric(factor(input_one_patient$cluster_id))
}
input_one_patient[input_one_patient$neo_load > 150, "neo_load"] = 150
ac = bc = rep(0, length(unique(input_one_patient$cluster_id)))
pi = 0.5
a = 0
zck_list = list()
ac_list = list()
bc_list = list()
acp_rate_ac_list = list()

```

```

acp_rate_bc_list = list()
a_all = c()
pi_all = c()
for (iter in 1:max_iter) {
  if (iter/1000 == round(iter/1000)) {
    cat(paste("Iteration", iter, "\n"))
    print(ac)
    print(ac)
    print(bc)
    print(acp_rate_ac)
    print(acp_rate_bc)
  }
  acp_rate_ac = rep(FALSE, length(unique(input_one_patient$cluster_id)))
  acp_rate_bc = rep(FALSE, length(unique(input_one_patient$cluster_id)))
  zck_df = input_one_patient[, c("mutation_id", "cluster_id")]
  zck_df$zck = 1
  if (multi_sample == T) {
    for (p in 1:length(unique(input_one_patient$phi))) {
      input_each_phi = input_one_patient[input_one_patient$phi ==
        unique(input_one_patient$phi)[p], ]
      for (c in unique(input_each_phi$cluster_id)) {
        input_each_clone = input_each_phi[input_each_phi$cluster_id ==
          c, ]
        vck = input_each_clone$variant_allele_frequency
        lambda = exp(ac[c] * vck + bc[c])
        nck = input_each_clone$neo_load
        r_tmp = pi * (nck == 0)/(pi * (nck == 0) +
          (1 - pi) * dpois(nck, lambda, log = F))
        r_tmp_deno = pi * (nck == 0) + (1 - pi) * dpois(nck,
          lambda, log = F)
        r_tmp[r_tmp_deno == 0] = 0
        zck = 1 * (runif(length(nck), 0, 1) > r_tmp)
        names(zck) = input_each_clone$mutation_id
        zck_df$zck[zck_df$mutation_id %in% names(zck)] = zck
        bc_prim = rnorm(1, bc[c], sqrt(sigma_p_sqr))
        lambda_prim_b = exp(ac[c] * vck + bc_prim)
        lambda = exp(ac[c] * vck + bc[c])
        tmp_prim = sum((zck == 1) * dpois(nck, lambda_prim_b,
          log = T))
        tmp = sum((zck == 1) * dpois(nck, lambda, log = T))
        llhr_b = exp(tmp_prim - bc_prim^2/(2 * sigma_square) -
          tmp + bc[c]^2/(2 * sigma_square))
        acceptance_function_b = min(1, llhr_b)
        u = runif(1, 0, 1)
        if (u <= acceptance_function_b) {
          bc[c] = bc_prim
          acp_rate_bc[c] = TRUE
        }
      }
    }
    input_each_phi$bc = bc[input_each_phi$cluster_id]
    input_each_phi$ac = ac[c]
    vck_phi = input_each_phi$variant_allele_frequency
    lambda_phi = exp(input_each_phi$ac * vck_phi +

```

```

    input_each_phi$bc)
nck_phi = input_each_phi$neo_load
zck_phi = zck_df[input_each_phi$mutation_id,
  "zck"]
ac_prim = rnorm(1, ac[c], sqrt(sigma_p_sqr))
lambda_prim_a = exp(ac_prim * vck_phi + input_each_phi$bc)
tmp_prim = sum((zck_phi == 1) * dpois(nck_phi,
  lambda_prim_a, log = T))
tmp = sum((zck_phi == 1) * dpois(nck_phi, lambda_phi,
  log = T))
if (length(table(input_one_patient$cluster_id)) ==
  1) {
  llhr_a = exp(tmp_prim - ac_prim^2/(2 * sigma_square) -
    tmp + ac[c]^2/(2 * sigma_square))
}
else {
  llhr_a = exp(tmp_prim - (ac_prim - a)^2/(2 *
    sigma_a_sqr) - tmp + (ac[c] - a)^2/(2 * sigma_a_sqr))
}
acceptance_function_a = min(1, llhr_a)
u = runif(1, 0, 1)
if (u <= acceptance_function_a) {
  ac[phi_cluster$phi == unique(input_each_clone$phi)] = ac_prim
  acp_rate_ac[c] = TRUE
}
}
pi = rbeta(1, alpha + sum((zck_df$zck == 0) * (input_one_patient$neo_load ==
  0)), beta + sum(zck_df$zck == 1))
A = 1/sigma_square + length(unique(input_one_patient$phi))/sigma_a_sqr
B = sum(ac[!duplicated(phi_cluster$phi)])/sigma_a_sqr
a = rnorm(1, B/A, sqrt(1/A))
ac_list[[iter]] = ac
bc_list[[iter]] = bc
zck_list[[iter]] = zck_df$zck
acp_rate_ac_list[[iter]] = acp_rate_ac
acp_rate_bc_list[[iter]] = acp_rate_bc
a_all = c(a_all, a)
pi_all = c(pi_all, pi)
}
else {
  for (c in 1:length(unique(input_one_patient$cluster_id))) {
    input_each_clone = input_one_patient[input_one_patient$cluster_id ==
      unique(input_one_patient$cluster_id)[c], ]
    vck = input_each_clone$variant_allele_frequency
    lambda = exp(ac[c] * vck + bc[c])
    nck = input_each_clone$neo_load
    r_tmp = pi * (nck == 0)/(pi * (nck == 0) + (1 -
      pi) * dpois(nck, lambda, log = F))
    r_tmp_deno = pi * (nck == 0) + (1 - pi) * dpois(nck,
      lambda, log = F)
    r_tmp[r_tmp_deno == 0] = 0
    zck = 1 * (runif(length(nck), 0, 1) > r_tmp)
    names(zck) = input_each_clone$mutation_id
  }
}

```

```

zck_df$zck[zck_df$mutation_id %in% names(zck)] = zck
ac_prim = rnorm(1, ac[c], sqrt(sigma_p_sqr))
lambda_prim_a = exp(ac_prim * vck + bc[c])
tmp_prim = sum((zck == 1) * dpois(nck, lambda_prim_a,
  log = T))
tmp = sum((zck == 1) * dpois(nck, lambda, log = T))
if (length(table(input_one_patient$cluster_id)) ==
  1) {
  llhr_a = exp(tmp_prim - ac_prim^2/(2 * sigma_square) -
    tmp + ac[c]^2/(2 * sigma_square))
}
else {
  llhr_a = exp(tmp_prim - (ac_prim - a)^2/(2 *
    sigma_a_sqr) - tmp + (ac[c] - a)^2/(2 * sigma_a_sqr))
}
acceptance_function_a = min(1, llhr_a)
u = runif(1, 0, 1)
if (u <= acceptance_function_a) {
  ac[c] = ac_prim
  acp_rate_ac[c] = TRUE
}
bc_prim = rnorm(1, bc[c], sqrt(sigma_p_sqr))
lambda_prim_b = exp(ac[c] * vck + bc_prim)
lambda = exp(ac[c] * vck + bc[c])
tmp_prim = sum((zck == 1) * dpois(nck, lambda_prim_b,
  log = T))
tmp = sum((zck == 1) * dpois(nck, lambda, log = T))
llhr_b = exp(tmp_prim - bc_prim^2/(2 * sigma_square) -
  tmp + bc[c]^2/(2 * sigma_square))
acceptance_function_b = min(1, llhr_b)
u = runif(1, 0, 1)
if (u <= acceptance_function_b) {
  bc[c] = bc_prim
  acp_rate_bc[c] = TRUE
}
}
pi = rbeta(1, alpha + sum((zck_df$zck == 0) * (input_one_patient$neo_load ==
  0)), beta + sum(zck_df$zck == 1))
A = 1/sigma_square + length(unique(input_one_patient$cluster_id))/sigma_a_sqr
B = sum(ac)/sigma_a_sqr
a = rnorm(1, B/A, sqrt(1/A))
ac_list[[iter]] = ac
bc_list[[iter]] = bc
zck_list[[iter]] = zck_df$zck
acp_rate_ac_list[[iter]] = acp_rate_ac
acp_rate_bc_list[[iter]] = acp_rate_bc
a_all = c(a_all, a)
pi_all = c(pi_all, pi)
}
}
keep = round(max_iter/2):max_iter
ac_final = Reduce("+", ac_list[keep])/length(keep)
bc_final = Reduce("+", bc_list[keep])/length(keep)

```

```
zck_df_final = round(Reduce("+", zck_list[keep])/length(keep))
names(zck_df_final) = zck_df$mutation_id
ac_rate = Reduce("+", acp_rate_ac_list[keep])/length(keep)
bc_rate = Reduce("+", acp_rate_bc_list[keep])/length(keep)
a_final = mean(a_all[keep])
pi_final = mean(pi_all[keep])
if (multi_sample == TRUE) {
  final_parameters = list(zck = data.frame(zck_df_final),
    ac = ac_final, bc = bc_final, acp_rate_ac = ac_rate,
    a = a_final, acp_rate_bc = bc_rate, pi = pi_final,
    phi_cluster = phi_cluster)
}
else {
  final_parameters = list(zck = data.frame(zck_df_final),
    ac = ac_final, bc = bc_final, acp_rate_ac = ac_rate,
    a = a_final, acp_rate_bc = bc_rate, pi = pi_final)
}
all_parameters = list(zck = zck_list, ac = ac_list, bc = bc_list,
  a = a_all, pi = pi_all)
result = list(all_parameters = all_parameters, final_parameters = final_parameters)
return(result)
}
```

Index

- * **datasets**
 - input_data, [3](#)
- * **immune**
 - netie, [3](#)
- * **neoantigen**
 - netie, [3](#)
- * **package neoantigen**
 - netie-package, [1](#)

input_data, [3](#)

netie, [3](#)

netie-package, [1](#)