Package ‘ShapDoE’

May 7, 2026

Type Package

Title Approximation of the Shapley Values Based on Experimental Designs

Version 1.0.0

Maintainer Liuqing Yang <yliuqing0714@163.com>

Description Estimating the Shapley values using the algorithm in the paper Liuqing Yang, Yongdao Zhou, Haoda Fu, Min-Qian Liu and Wei Zheng (2024) <doi:10.1080/01621459.2023.2257364> ``Fast Approximation of the Shapley Values Based on Order-of-Addition Experimental Designs''. You provide the data and define the value function, it retures the estimated Shapley values based on sampling methods or experimental designs.

License MIT + file LICENSE

Encoding UTF-8

Imports gtools

RoxygenNote 7.2.3

NeedsCompilation no

Author Liuqing Yang [aut, cre, cph]

Repository CRAN

Date/Publication 2024-04-30 11:42:47 UTC

Contents

`est.sh` *The main algorithm for estimating the Shapley value*

Description

The main algorithm for estimating the Shapley value

Usage

```
est.sh(method, d, n, val, ..., p = NA, f_d = NA)
```

Arguments

`method`	the method used for estimating,'SRS' meas simple random sampling, 'StrRS' means structured simple random sampling, 'LS' means Latin square and 'COA' means component orthogonal array.
`d`	an integer, the number of players.
`n`	an integer, the sample size.
`val`	the predefined value function.
`...`	other parameters used in val(sets,...).
`p`	a prime, the bottom number of d.
`f_d`	a vector represents the coefficients of primative polynomial on GF(d). For example the primative polynomial on GF(3^2) is x^2+x+2, then let f_d=c(1,1,2).

Value

a vector including estimated Shapley values of all players.

Examples

```
temp_adjacent<-matrix(0,nrow=8,ncol=8)
temp_adjacent[1,6:8]<-1;temp_adjacent[2,7]<-1;temp_adjacent[c(4,6,7),8]<-1;
temp_adjacent<-temp_adjacent+t(temp_adjacent)
temp_val<-function(sets,adjacent){
  if(length(sets)==1) val<-0
  else{
    subadjacent<-adjacent[sets,sets]
    nsets<-length(sets)
    A<-diag(1,nsets); B<-matrix(0,nsets,nsets)
```

```
    for(l in 1:(nsets-1)){
      A<-A%*%subadjacent
      B<-B+A
    }
    val<-ifelse(sum(B==0)>nsets,0,1)
  }
  return(val)
}
est.sh('SRS',8,112,temp_val,temp_adjacent)
est.sh('StrRS',8,112,temp_val,temp_adjacent)
est.sh('LS',8,112,temp_val,temp_adjacent)
est.sh('COA',8,112,temp_val,temp_adjacent,p=2,f_d=c(1,0,1,1))
```

`est.shcoa` *Estimating the Shapley value based on component orthogonal array (COA) with a prime power d*

Description

Estimating the Shapley value based on component orthogonal array (COA) with a prime power d

Usage

```
est.shcoa(d, n, val, p, f_d, ...)
```

Arguments

`d`	a power of prime p, the number of players.
`n`	an integer, the sample size.
`val`	the predefined value function.
`p`	a prime, the bottom number of d.
`f_d`	a vector represents the coefficients of primative polynomial on GF(d). For example the primative polynomial on GF(3^2) is x^2+x+2, then let f_d=c(1,1,2).
`...`	other parameters used in val(sets,...).

Value

a vector including estimated Shapley values of all players based on COA.

Examples

```
temp_adjacent<-matrix(0,nrow=8,ncol=8)
temp_adjacent[1,6:8]<-1;temp_adjacent[2,7]<-1;temp_adjacent[c(4,6,7),8]<-1;
temp_adjacent<-temp_adjacent+t(temp_adjacent)
temp_val<-function(sets,adjacent){
  if(length(sets)==1) val<-0
  else{
    subadjacent<-adjacent[sets,sets]
```

```
    nsets<-length(sets)
    A<-diag(1,nsets); B<-matrix(0,nsets,nsets)
    for(l in 1:(nsets-1)){
      A<-A%*%subadjacent
      B<-B+A
    }
    val<-ifelse(sum(B==0)>nsets,0,1)
  }
  return(val)
}
est.shcoa(8,112,temp_val,2,c(1,0,1,1),temp_adjacent)
```

---

est.shcoa.prime *Estimating the Shapley value based on component orthogonal array (COA) with a prime d*

Description

Estimating the Shapley value based on component orthogonal array (COA) with a prime d

Usage

```
est.shcoa.prime(d, n, val, ...)
```

Arguments

d	a prime, the number of players.
n	an integer, the sample size.
val	the predefined value function.
...	other parameters used in val(sets,...).

Value

a vector including estimated Shapley values of all players based on COA.

Examples

```
temp_adjacent<-matrix(0,nrow=5,ncol=5)
temp_adjacent[1,c(2,3,5)]<-1;temp_adjacent[2,4]<-1;temp_adjacent[3,5]<-1;
temp_adjacent<-temp_adjacent+t(temp_adjacent)
temp_val<-function(sets,adjacent){
  if(length(sets)==1) val<-0
  else{
    subadjacent<-adjacent[sets,sets]
    nsets<-length(sets)
    A<-diag(1,nsets); B<-matrix(0,nsets,nsets)
    for(l in 1:(nsets-1)){
      A<-A%*%subadjacent
```

```
      B<-B+A
    }
    val<-ifelse(sum(B==0)>nsets,0,1)
  }
  return(val)
}
est.shcoa.prime(5,20,temp_val,temp_adjacent)
```

---

est.shls *Estimating the Shapley value based on Latin square (LS)*

Description

Estimating the Shapley value based on Latin square (LS)

Usage

```
est.shls(d, n, val, ...)
```

Arguments

d	an integer, the number of players.
n	an integer, the sample size.
val	the predefined value function.
...	other parameters used in val(sets,...).

Value

a vector including estimated Shapley values of all players based on LS.

Examples

```
temp_adjacent<-matrix(0,nrow=8,ncol=8)
temp_adjacent[1,6:8]<-1;temp_adjacent[2,7]<-1;temp_adjacent[c(4,6,7),8]<-1;
temp_adjacent<-temp_adjacent+t(temp_adjacent)
temp_val<-function(sets,adjacent){
  if(length(sets)==1) val<-0
  else{
    subadjacent<-adjacent[sets,sets]
    nsets<-length(sets)
    A<-diag(1,nsets); B<-matrix(0,nsets,nsets)
    for(l in 1:(nsets-1)){
      A<-A%*%subadjacent
      B<-B+A
    }
    val<-ifelse(sum(B==0)>nsets,0,1)
  }
  return(val)
}
est.shls(8,56,temp_val,temp_adjacent)
```

est.shsrs *Estimating the Shapley value based on simple random sampling (SRS)*

Description

Estimating the Shapley value based on simple random sampling (SRS)

Usage

```
est.shsrs(d, n, val, ...)
```

Arguments

d	an integer, the number of players.
n	an integer, the sample size.
val	the predefined value function.
...	other parameters used in val(sets,...).

Value

a vector including estimated Shapley values of all players based on SRS.

Examples

```
temp_adjacent<-matrix(0,nrow=8,ncol=8)
temp_adjacent[1,6:8]<-1;temp_adjacent[2,7]<-1;temp_adjacent[c(4,6,7),8]<-1;
temp_adjacent<-temp_adjacent+t(temp_adjacent)
temp_val<-function(sets,adjacent){
  if(length(sets)==1) val<-0
  else{
    subadjacent<-adjacent[sets,sets]
    nsets<-length(sets)
    A<-diag(1,nsets); B<-matrix(0,nsets,nsets)
    for(l in 1:(nsets-1)){
      A<-A%*%subadjacent
      B<-B+A
    }
    val<-ifelse(sum(B==0)>nsets,0,1)
  }
  return(val)
}
est.shsrs(8,112,temp_val,temp_adjacent)
```

est.shstrrs — *Estimating the Shapley value based on structured simple random sampling (StrRS)*

Description

Estimating the Shapley value based on structured simple random sampling (StrRS)

Usage

```
est.shstrrs(d, n, val, ...)
```

Arguments

`d`	an integer, the number of players.
`n`	an integer, the sample size.
`val`	the predefined value function.
`...`	other parameters used in val(sets,...).

Value

a vector including estimated Shapley values of all players based on StrRS.

Examples

```
temp_adjacent<-matrix(0,nrow=8,ncol=8)
temp_adjacent[1,6:8]<-1;temp_adjacent[2,7]<-1;temp_adjacent[c(4,6,7),8]<-1;
temp_adjacent<-temp_adjacent+t(temp_adjacent)
temp_val<-function(sets,adjacent){
  if(length(sets)==1) val<-0
  else{
    subadjacent<-adjacent[sets,sets]
    nsets<-length(sets)
    A<-diag(1,nsets); B<-matrix(0,nsets,nsets)
    for(l in 1:(nsets-1)){
      A<-A%*%subadjacent
      B<-B+A
    }
    val<-ifelse(sum(B==0)>nsets,0,1)
  }
  return(val)
}
est.shstrrs(8,112,temp_val,temp_adjacent)
```

`gfpoly.add` *Polynomial additive defined on GF(s) with a prime s*

Description

Polynomial additive defined on GF(s) with a prime s

Usage

```
gfpoly.add(f1, f2, s)
```

Arguments

`f1`	a vector represents the coefficients of the first addend polynomial. For example, if the dividend is x^5+2x^3+1, then f1=c(1,0,2,0,0,1).
`f2`	a vector represents the coefficients of the second addend polynomial. For example, if the divisor is x^4+2, then f2=c(1,0,0,0,2).
`s`	a prime, the order of the Galois filed (GF).

Value

a vector represents the coefficients of the resulting polynomial. For example, the result c(1, 1, 2, 0, 0, 0) represents x^5+x^4+2x^3.

Examples

```
gfpoly.add(c(1,0,2,0,0,1),c(1,0,0,0,2),3)
```

`gfpoly.div` *Polynomial division defined on GF(s) with a prime s*

Description

Polynomial division defined on GF(s) with a prime s

Usage

```
gfpoly.div(f1, f2, s)
```

Arguments

`f1`	a vector represents the coefficients of the dividend polynomial. For example, if the dividend is x^5+2x^3+1, then f1=c(1,0,2,0,0,1).
`f2`	a vector represents the coefficients of the dividend polynomial. For example, if the divisor is x^4+2, then f2=c(1,0,0,0,2).
`s`	a prime, the order of the Galois filed (GF).

Value

a vector represents the coefficients of the resulting polynomial. For example, the result c(2,0,1,1) represents 2x^3+x+1.

Examples

```
gfpoly.div(c(1,0,2,0,0,1),c(1,0,0,0,2),3)
```

`gfpoly.multi` *Polynomial mutiplication defined on GF(s) with a prime s*

Description

Polynomial mutiplication defined on GF(s) with a prime s

Usage

```
gfpoly.multi(f1, f2, s)
```

Arguments

`f1`	a vector represents the coefficients of the first multiplier polynomial. For example, if the dividend is x^5+2x^3+1, then f1=c(1,0,2,0,0,1).
`f2`	a vector represents the coefficients of the second multiplier polynomial. For example, if the divisor is x^4+2, then f2=c(1,0,0,0,2).
`s`	a prime, the order of the Galois filed (GF).

Value

a vector represents the coefficients of the resulting polynomial. For example, the result c(1,0,2,0,2,1,1,0,0,2) represents x^9+2x^7+2x^5+x^4+x^3+2.

Examples

```
gfpoly.multi(c(1,0,2,0,0,1),c(1,0,0,0,2),3)
```

`is.prime` *Determine whether an integer is a prime*

Description

Determine whether an integer is a prime

Usage

```
is.prime(x)
```

Arguments

x	the integer to be determined.

Value

the result: TRUE (x is a prime) or FALSE (x is not a prime).

Examples

```
is.prime(7)
is.prime(8)
```

`onecoa` *Generate a component orthogonal array (COA) with a prime power d*

Description

Generate a component orthogonal array (COA) with a prime power d

Usage

```
onecoa(d, p, f_d)
```

Arguments

d	a power of prime p, the column of the resulting COA.
p	a prime, the bottom number of d.
f_d	a vector represents the coefficients of primative polynomial on GF(d). For example the primative polynomial on GF(3^2) is x^2+x+2, then let f_d=c(1,1,2).

Value

a COA with d(d-1) rows and d columns.

Examples

```
onecoa(9,3,c(1,1,2))
```

`onecoa.prime` *Generate a component orthogonal array (COA) with a prime d*

Description

Generate a component orthogonal array (COA) with a prime d

Usage

```
onecoa.prime(d)
```

Arguments

`d`	a prime, the column of the resulting COA.

Value

a COA with d(d-1) rows and d columns.

Examples

```
onecoa.prime(5)
```

`onels` *Generate an Latin square (LS)*

Description

Generate an Latin square (LS)

Usage

```
onels(d)
```

Arguments

`d`	an integer, the run size of the resulting LS.

Value

an LS with d rows and d columns.

Examples

```
onels(5)
```

`poly.div` *Polynomial division*

Description

Polynomial division

Usage

```
poly.div(f1, f2)
```

Arguments

`f1`	a vector represents the coefficients of the dividend polynomial. For example, if the dividend is x^5+2x^3+1, then f1=c(1,0,2,0,0,1).
`f2`	a vector represents the coefficients of the dividend polynomial. For example, if the divisor is x^4+2, then f2=c(1,0,0,0,2).

Value

a vector represents the coefficients of the resulting polynomial. For example, the result c(2,0,-2,1) represents 2x^3-2x+1.

Examples

```
poly.div(c(1,0,2,0,0,1),c(1,0,0,0,2))
```

`structed.perm` *Generate the structured samples of simple random samples*

Description

Generate the structured samples of simple random samples

Usage

```
structed.perm(permatrix, jcom, d)
```

Arguments

`permatrix`	a matrix, each row is a permutation.
`jcom`	an integer, represents the target component. Hope that the component jcom appears the same number of at each position.
`d`	the number of components.

Value

a matrix represents the structured samples.

Examples

```
temp_samples<-matrix(nrow=10,ncol=5)
for(i in 1:10){temp_samples[i,]<-sample(1:5,5)}
structed.perm(temp_samples,3,5)
```

Index