

# Package ‘yap’

January 15, 2020

**Title** Yet Another Probabilistic Neural Network

**Version** 0.1.0

**Author** WenSui Liu

**Maintainer** WenSui Liu <liuwensui@gmail.com>

**Description** Another implementation of probabilistic neural network in R based on Specht (1990) <DOI:10.1016/0893-6080(90)90049-Q>. It is applicable to the pattern recognition with a N-level response, where  $N > 2$ .

**URL** <https://github.com/statcompute/yap>

**Depends** R (>= 3.6.0)

**Imports** stats, randtoolbox, lhs, parallel, datasets

**License** GPL (>= 2)

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.0.2

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2020-01-15 09:50:02 UTC

## R topics documented:

dummies . . . . .	2
folds . . . . .	2
gen_latin . . . . .	3
gen_sobol . . . . .	4
gen_unifm . . . . .	4
logl . . . . .	5
pnn.fit . . . . .	6
pnn.imp . . . . .	6
pnn.optimiz_logl . . . . .	7
pnn.parpred . . . . .	8
pnn.pfi . . . . .	9

pnn.predict . . . . .	9
pnn.predone . . . . .	10
pnn.search_logl . . . . .	11
pnn.x_imp . . . . .	12
pnn.x_pfi . . . . .	12

<b>Index</b>	<b>14</b>
--------------	-----------

---

dummies	<i>Convert a N-category vector to a N-dimension matrix</i>
---------	--

---

### Description

The function dummies converts a N-category vector to a N-dimension matrix

### Usage

```
dummies(x)
```

### Arguments

x                    A N-category vector

### Value

A N-dimension matrix with 0/1 values

### Examples

```
data(iris, package = "datasets")
dummies(iris[, 5])
```

---

folds	<i>Generate a list of index for the n-fold cross-validation</i>
-------	---

---

### Description

The function folds generates a list of index for the n-fold cross-validation

### Usage

```
folds(idx, n, seed = 1)
```

### Arguments

idx                    A vector of index list  
n                        The number of n folds  
seed                    The seed value to generate random n-fold index

**Value**

A list of n-fold index

**Examples**

```
folds(seq(10), 3, 2020)
```

---

gen\_latin

*Generate random numbers of latin hypercube sampling*

---

**Description**

The function `gen_latin` generates a vector of random numbers by latin hypercube sampling

**Usage**

```
gen_latin(min = 0, max = 1, n, seed = 1)
```

**Arguments**

min	The minimum value of random numbers
max	The maximum value of random numbers
n	The number of random numbers to generate
seed	The seed value of random number generation

**Value**

A vector of random numbers bounded by the min and max

**Examples**

```
gen_latin(0, 1, 10, 2020)
```

---

gen_sobol	<i>Generate sobol sequence</i>
-----------	--------------------------------

---

**Description**

The function `gen_sobol` generates a vector of scrambled sobol sequence

**Usage**

```
gen_sobol(min = 0, max = 1, n, seed = 1)
```

**Arguments**

<code>min</code>	The minimum value of random numbers
<code>max</code>	The maximum value of random numbers
<code>n</code>	The number of random numbers to generate
<code>seed</code>	The seed value of random number generation

**Value**

A vector of sobol sequence bounded by the min and max

**Examples**

```
gen_sobol(0, 1, 10, 2020)
```

---

gen_unifm	<i>Generate Uniform random numbers</i>
-----------	--

---

**Description**

The function `gen_unifm` generates a vector of uniform random numbers

**Usage**

```
gen_unifm(min = 0, max = 1, n, seed = 1)
```

**Arguments**

<code>min</code>	The minimum value of random numbers
<code>max</code>	The maximum value of random numbers
<code>n</code>	The number of random numbers to generate
<code>seed</code>	The seed value of random number generation

**Value**

A vector of uniform random numbers bounded by the min and max

**Examples**

```
gen_unifm(0, 1, 10, 2020)
```

---

logl	<i>Calculate the multiclass cross-entropy</i>
------	---

---

**Description**

The function logl calculates the multiclass cross entropy

**Usage**

```
logl(y_true, y_pred)
```

**Arguments**

y_true	A matrix of multiclass 0/1 indicators
y_pred	A matrix of predicted probability of each class

**Value**

The value of multiclass cross entropy

**Examples**

```
data(iris, package = "datasets")
Y <- iris[, 5]
X <- scale(iris[, 1:4])
pnet <- pnn.fit(x = X, y = Y)
logl(y_true = pnet$y.ind, y_pred = pnn.predict(pnet, X))
```

pnn.fit *Create a probabilistic neural network*

---

### Description

The function `pnn.fit` creates a probabilistic neural network (PNN)

### Usage

```
pnn.fit(x, y, sigma = 1)
```

### Arguments

<code>x</code>	A matrix of predictors
<code>y</code>	A vector of N-category factors
<code>sigma</code>	A scalar with the positive value

### Value

A PNN object

### References

Donald Specht. (1990). Probabilistic Neural Networks.

### Examples

```
data(iris, package = "datasets")
Y <- iris[, 5]
X <- scale(iris[, 1:4])
pnet <- pnn.fit(x = X, y = Y)
```

---

pnn.imp *Derive the importance rank of all predictors used in the PNN*

---

### Description

The function `pnn.imp` derives the importance rank of all predictors used in the PNN. It essentially is a wrapper around the function `pnn.x_imp`.

### Usage

```
pnn.imp(net)
```

### Arguments

<code>net</code>	A PNN object generated by <code>pnn.fit()</code>
------------------	--

**Value**

A dataframe with important values of all predictors in the PNN

**See Also**

[pnn.x\\_imp](#)

**Examples**

```
data(iris, package = "datasets")
Y <- iris[, 5]
X <- scale(iris[, 1:4])
pnet <- pnn.fit(x = X, y = Y)

pnn.imp(pnet)
```

---

pnn.optimiz_log1	<i>Optimize the optimal value of PNN smoothing parameter based on the cross entropy</i>
------------------	---

---

**Description**

The function `pnn.optimiz_log1` optimize the optimal value of PNN smoothing parameter by cross-validation.

**Usage**

```
pnn.optimiz_log1(net, lower = 0, upper, nfolds = 4, seed = 1, method = 1)
```

**Arguments**

net	A PNN object generated by <code>pnn.fit()</code>
lower	A scalar for the lower bound of the smoothing parameter, 0 by default
upper	A scalar for the upper bound of the smoothing parameter
nfolds	A scalar for the number of n-fold, 4 by default
seed	The seed value for the n-fold cross-validation, 1 by default
method	A scalar referring to the optimization method, 1 for Golden section searc and 2 for Brent's method

**Value**

The best outcome

**See Also**

[pnn.search\\_log1](#)

## Examples

```
data(iris, package = "datasets")
Y <- iris[, 5]
X <- scale(iris[, 1:4])
pnet <- pnn.fit(x = X, y = Y)

pnn.optimiz_logl(pnet, upper = 1)
```

---

pnn.parpred

*Calculate predicted probabilities of PNN by using parallelism*

---

## Description

The function `pnn.parpred` calculates a matrix of PNN predicted probabilities based on an input matrix

## Usage

```
pnn.parpred(net, x)
```

## Arguments

<code>net</code>	A PNN object generated by <code>pnn.fit()</code>
<code>x</code>	A matrix of input predictors

## Value

A matrix of predicted probabilities

## See Also

[pnn.predict](#)

## Examples

```
data(iris, package = "datasets")
Y <- iris[, 5]
X <- scale(iris[, 1:4])
pnet <- pnn.fit(x = X, y = Y)
pnn.parpred(pnet, X[seq(5), ])
```



---

pnn.pfi	<i>Derive the PFI rank of all predictors used in the PNN</i>
---------	--

---

**Description**

The function `pnn.pfi` derives the PFI rank of all predictors used in the PNN. It essentially is a wrapper around the function `pnn.x_pfi`.

**Usage**

```
pnn.pfi(net, ntry = 1000, seed = 1)
```

**Arguments**

<code>net</code>	A PNN object generated by <code>pnn.fit()</code>
<code>ntry</code>	The number of random permutations to try, 1e3 times by default
<code>seed</code>	The seed value for the random permutation

**Value**

A dataframe with PFI values of all predictors in the PNN

**See Also**

[pnn.x\\_pfi](#)

**Examples**

```
data(iris, package = "datasets")
Y <- iris[, 5]
X <- scale(iris[, 1:4])
pnet <- pnn.fit(x = X, y = Y)

pnn.pfi(pnet)
```

---

pnn.predict	<i>Calculate a matrix of predicted probabilities</i>
-------------	--

---

**Description**

The function `pnn.predict` calculates a matrix of predicted probabilities based on a matrix of predictors

**Usage**

```
pnn.predict(net, x)
```

**Arguments**

net                    The PNN object generated by pnn.fit()  
 x                      The matrix of input predictors

**Value**

A matrix of predicted probabilities for all categories

**See Also**

[pnn.predone](#)

**Examples**

```
data(iris, package = "datasets")
Y <- iris[, 5]
X <- scale(iris[, 1:4])
pnet <- pnn.fit(x = X, y = Y)
pnn.predict(pnet, X[seq(5), ])
```

---

pnn.predone

*Calculate the predicted probability for each category of PNN*

---

**Description**

The function pnn.predone calculates the predicted probability for each category of PNN

**Usage**

```
pnn.predone(net, x)
```

**Arguments**

net                    A PNN object created by pnn.fit()  
 x                      A vector of input predictors

**Value**

A one-row matrix of predicted probabilities

**See Also**

[pnn.fit](#)

**Examples**

```
data(iris, package = "datasets")
Y <- iris[, 5]
X <- scale(iris[, 1:4])
pnet <- pnn.fit(x = X, y = Y)
for (i in seq(5)) print(pnn.predone(pnet, X[i, ]))
```

---

pnn.search_log1	<i>Search for the optimal value of PNN smoothing parameter based on the cross entropy</i>
-----------------	---

---

**Description**

The function `pnn.search_log1` searches for the optimal value of PNN smoothing parameter by cross-validation.

**Usage**

```
pnn.search_log1(net, sigmas, nfolds = 4, seed = 1)
```

**Arguments**

net	A PNN object generated by <code>pnn.fit()</code>
sigmas	A numeric vector to search for the best smoothing parameter
nfolds	A scalar for the number of n-fold, 4 by default
seed	The seed value for the n-fold cross-validation, 1 by default

**Value**

The list of all searching outcomes and the best outcome

**Examples**

```
data(iris, package = "datasets")
Y <- iris[, 5]
X <- scale(iris[, 1:4])
pnet <- pnn.fit(x = X, y = Y)
pnn.search_log1(pnet, c(0.5, 1), nfolds = 2)
```

---

pnn.x\_imp *Derive the importance of a predictor used in the PNN*

---

### Description

The function `pnn.x_imp` derives the importance of a predictor used in the PNN, where the "importance" is measured by the increase in cross entropy after eliminating the impact of the predictor in interest.

### Usage

```
pnn.x_imp(net, i)
```

### Arguments

<code>net</code>	A PNN object generated by <code>pnn.fit()</code>
<code>i</code>	The <i>i</i> th predictor in the PNN

### Value

A vector with the variable name and two values of importance measurements, namely "imp1" and "imp2". The "imp1" measures the increase in cross entropy after replacing all values of the predictor with its mean. The "imp2" measures the increase in cross entropy after dropping the predictor from the PNN.

### See Also

[pnn.x\\_pfi](#)

### Examples

```
data(iris, package = "datasets")
Y <- iris[, 5]
X <- scale(iris[, 1:4])
pnet <- pnn.fit(x = X, y = Y)
pnn.x_imp(pnet, 1)
```

---

pnn.x\_pfi *Derive the permutation feature importance of a predictor used in the PNN*

---

### Description

The function `pnn.x_pfi` derives the permutation feature importance (PFI) of a predictor used in the PNN, where the "importance" is defined by the increase in cross entropy after the predictor is randomly permuted.

**Usage**

```
pnn.x_pfi(net, i, ntry = 1000, seed = 1)
```

**Arguments**

<code>net</code>	A PNN object generated by <code>pnn.fit()</code>
<code>i</code>	The <i>i</i> th predictor in the PNN
<code>ntry</code>	The number of random permutations to try, 1e3 times by default
<code>seed</code>	The seed value for the random permutation

**Value**

A vector with the variable name and the PFI value.

**See Also**

[pnn.x\\_imp](#)

**Examples**

```
data(iris, package = "datasets")
Y <- iris[, 5]
X <- scale(iris[, 1:4])
pnet <- pnn.fit(x = X, y = Y)
pnn.x_pfi(pnet, 1)
```

# Index

dummies, [2](#)

folds, [2](#)

gen\_latin, [3](#)

gen\_sobol, [4](#)

gen\_unifm, [4](#)

logl, [5](#)

pnn.fit, [6](#), [10](#)

pnn.imp, [6](#)

pnn.optimiz\_logl, [7](#)

pnn.parpred, [8](#)

pnn.pfi, [9](#)

pnn.predict, [8](#), [9](#)

pnn.predone, [10](#), [10](#)

pnn.search\_logl, [7](#), [11](#)

pnn.x\_imp, [7](#), [12](#), [13](#)

pnn.x\_pfi, [9](#), [12](#), [12](#)