

# Package ‘CRF’

October 31, 2011

**Version** 0.2-0

**Date** 2011-10-26

**Title** CRF - Conditional Random Fields

**Author** Ling-Yun Wu <wulingyun@gmail.com>

**Author@R** c(person("Ling-Yun", "Wu", email = "wulingyun@gmail.com"))

**Maintainer** Ling-Yun Wu <wulingyun@gmail.com>

**Depends** R (>= 2.12.0)

**Description** Library to decode/infer/sample/train Conditional Random Fields

**License** GPL (>= 2)

## R topics documented:

CRF-package . . . . .	1
clamp.crf . . . . .	3
decode . . . . .	4
infer . . . . .	5
make.crf . . . . .	7
sample . . . . .	8
sub.crf . . . . .	9

<b>Index</b>	<b>11</b>
--------------	-----------

---

CRF-package	<i>CRF - Conditional Random Fields</i>
-------------	--

---

## Description

Library to decode/infer/sample/train Conditional Random Fields

## Details

CRF is R package for various computational tasks of conditional random fields as well as other probabilistic undirected graphical models of discrete data with pairwise (and unary) potentials. The decoding/inference/sampling tasks are implemented for general discrete undirected graphical models with pairwise potentials. The training task is less general, focusing on conditional random fields with log-linear potentials and a fixed structure. The code is written entirely in R and C++. The initial version is ported from UGM written by Mark Schmidt.

Decoding: Computing the most likely configuration

- `decode.exact` Exact decoding for small graphs with brute-force search
- `decode.chain` Exact decoding for chain-structured graphs with the Viterbi algorithm
- `decode.tree` Exact decoding for tree- and forest-structured graphs with max-product belief propagation
- `decode.conditional` Conditional decoding (takes another decoding method as input)
- `decode.cutset` Exact decoding for graphs with a small cutset using cutset conditioning
- `decode.sample` Approximate decoding using sampling (takes a sampling method as input)
- `decode.marginal` Approximate decoding using inference (takes an inference method as input)
- `decode.lbp` Approximate decoding using max-product loopy belief propagation
- `decode.trbp` Approximate decoding using max-product tree-reweighted belief propagation
- `decode.greedy` Approximate decoding with greedy algorithm
- `decode.icm` Approximate decoding with the iterated conditional modes algorithm

Inference: Computing the partition function and marginal probabilities

- `infer.exact` Exact inference for small graphs with brute-force counting
- `infer.chain` Exact inference for chain-structured graphs with the forward-backward algorithm
- `infer.tree` Exact inference for tree- and forest-structured graphs with sum-product belief propagation
- `infer.conditional` Conditional inference (takes another inference method as input)
- `infer.cutset` Exact inference for graphs with a small cutset using cutset conditioning
- `infer.sample` Approximate inference using sampling (takes a sampling method as input)
- `infer.lbp` Approximate inference using sum-product loopy belief propagation
- `infer.trbp` Approximate inference using sum-product tree-reweighted belief propagation

Sampling: Generating samples from the distribution

- `sample.exact` Exact sampling for small graphs with brute-force inverse cumulative distribution
- `sample.chain` Exact sampling for chain-structured graphs with the forward-filter backward-sample algorithm
- `sample.tree` Exact sampling for tree- and forest-structured graphs with sum-product belief propagation and backward-sampling
- `sample.conditional` Conditional sampling (takes another sampling method as input)
- `sample.cutset` Exact sampling for graphs with a small cutset using cutset conditioning
- `sample.gibbs` Approximate sampling using a single-site Gibbs sampler

Training: Given data, computing the most likely estimates of the parameters

**Author(s)**

Ling-Yun Wu <wulingyun@gmail.com>

**References**

- J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *the proceedings of International Conference on Machine Learning (ICML)*, pp. 282-289, 2001.
- Mark Schmidt. UGM: A Matlab toolbox for probabilistic undirected graphical models. <http://www.cs.ubc.ca/~schmidtm/Software/UGM.html>

**See Also**

`make.crf`, `decode`, `infer`, `sample`, `train`

**Examples**

```
library(CRF)
data(Small)
decode.exact(small.crf)
infer.exact(small.crf)
sample.exact(small.crf, 100)
```

---

`clamp.crf`*Make clamped CRF data structure*

---

**Description**

Generate clamped CRF data structure by fixing the states of some nodes

**Usage**

```
clamp.crf(crf, clamped)
```

**Arguments**

<code>crf</code>	The CRF data structure generated by <code>make.crf</code>
<code>clamped</code>	The vector of fixed states of nodes

**Details**

The function will generate a clamped CRF data structure from a given CRF data structure by fixing the states of some nodes. The vector `clamped` contains the desired state for each node while zero means the state is not fixed. The node and edge potentials are updated to the conditional potentials based on the clamped vector.

**Value**

The function will return a CRF data structure with additional components:

<code>original</code>	The original CRF data.
<code>clamped</code>	The vector of fixed states of nodes.
<code>node.id</code>	The vector of the original node ids for nodes in the new CRF data.
<code>node.map</code>	The vector of the new node ids for nodes in the original CRF data.
<code>edge.id</code>	The vector of the original edge ids for edges in the new CRF data.
<code>edge.map</code>	The vector of the new edge ids for edges in the original CRF data.

**See Also**

[make.crf](#), [sub.crf](#)

**Examples**

```
library(CRF)
data(Small)
crf <- clamp.crf(small.crf, c(0, 0, 1, 1))
```

---

decode

*Decoding methods*

---

**Description**

Computing the most likely configuration

**Usage**

```
decode.exact(crf)
decode.chain(crf)
decode.tree(crf)
decode.conditional(crf, clamped, decode.method, ...)
decode.cutset(crf, cutset)
decode.sample(crf, sample.method, ...)
decode.marginal(crf, infer.method, ...)
decode.lbp(crf, max.iter=10000, cutoff=1e-4, verbose=0)
decode.trbp(crf, max.iter=10000, cutoff=1e-4, verbose=0)
decode.greedy(crf, restart = 0, start = apply(crf$node.pot, 1, which.max))
decode.icm(crf, restart = 0, start = apply(crf$node.pot, 1, which.max))
decode.block(crf, blocks, decode.method = decode.tree, restart = 0,
             start = apply(crf$node.pot, 1, which.max), ...)
```

**Arguments**

`crf`

## Details

- `decode.exact` Exact decoding for small graphs with brute-force search
- `decode.chain` Exact decoding for chain-structured graphs with the Viterbi algorithm
- `decode.tree` Exact decoding for tree- and forest-structured graphs with max-product belief propagation
- `decode.conditional` Conditional decoding (takes another decoding method as input)
- `decode.cutset` Exact decoding for graphs with a small cutset using cutset conditioning
- `decode.sample` Approximate decoding using sampling (takes a sampling method as input)
- `decode.marginal` Approximate decoding using inference (takes an inference method as input)
- `decode.lbp` Approximate decoding using max-product loopy belief propagation
- `decode.trbp` Approximate decoding using max-product tree-reweighted belief propagation
- `decode.greedy` Approximate decoding with greedy algorithm
- `decode.icm` Approximate decoding with the iterated conditional modes algorithm
- `decode.block` Approximate decoding with the block iterated conditional modes algorithm

## See Also

`make.crf`, `infer`, `sample`

## Examples

```
library(CRF)
data(Small)
decode.exact(small.crf)
decode.chain(small.crf)
decode.tree(small.crf)
decode.conditional(small.crf, c(0,1,0,0), decode.exact)
decode.cutset(small.crf, c(2))
decode.sample(small.crf, sample.exact, 10000)
decode.marginal(small.crf, infer.exact)
decode.lbp(small.crf)
decode.trbp(small.crf)
decode.greedy(small.crf)
decode.icm(small.crf)
decode.block(small.crf, list(c(1,3), c(2,4)))
```

---

infer

*Inference methods*

---

## Description

Computing the partition function and marginal probabilities

## Usage

```
infer.exact(crf)
infer.chain(crf)
infer.tree(crf)
infer.conditional(crf, clamped, infer.method, ...)
infer.cutset(crf, cutset)
infer.sample(crf, sample.method, ...)
infer.lbp(crf, max.iter = 10000, cutoff = 1e-4, verbose = 0)
infer.trbp(crf, max.iter = 10000, cutoff = 1e-4, verbose = 0)
```

## Arguments

`crf`

## Details

- [infer.exact](#) Exact inference for small graphs with brute-force counting
- [infer.chain](#) Exact inference for chain-structured graphs with the forward-backward algorithm
- [infer.tree](#) Exact inference for tree- and forest-structured graphs with sum-product belief propagation
- [infer.conditional](#) Conditional inference (takes another inference method as input)
- [infer.cutset](#) Exact inference for graphs with a small cutset using cutset conditioning
- [infer.sample](#) Approximate inference using sampling (takes a sampling method as input)
- [infer.lbp](#) Approximate inference using sum-product loopy belief propagation
- [infer.trbp](#) Approximate inference using sum-product tree-reweighted belief propagation

## See Also

[make.crf](#), [decode](#), [sample](#)

## Examples

```
library(CRF)
data(Small)
infer.exact(small.crf)
infer.chain(small.crf)
infer.tree(small.crf)
infer.conditional(small.crf, c(0,1,0,0), infer.exact)
infer.cutset(small.crf, c(2))
infer.sample(small.crf, sample.exact, 10000)
infer.lbp(small.crf)
infer.trbp(small.crf)
```

make.crf

*Make CRF data structure***Description**

Generate CRF data structure from the adjacent matrix

**Usage**

```
make.crf(adj.matrix, nstates)
```

**Arguments**

adj.matrix	The adjacent matrix of CRF network
nstates	The state numbers of nodes

**Details**

The function will generate a empty CRF data structure from a given adjacent matrix. If the length of nstates is less than n.nodes, it will be used repeatedly. All node and edge potentials are initialized as 1.

**Value**

The function will return a CRF data structure, which is a list with components:

n.nodes	The number of nodes.
n.edges	The number of edges.
n.states	The number of states for each node. It is a vector of length n.nodes.
max.state	The maximum number of states. It is equal to max(n.states).
edges	The node pair of each edge. It is a matrix with 2 columns and n.edges rows. Each row denotes one edge. The node with smaller id is put in the first column.
n.adj	The number of adjacent nodes for each node. It is a vector of length n.nodes.
adj.nodes	The list of adjacent nodes for each node. It is a list of length n.nodes and the i-th element is a vector of length n.adj[i].
adj.edges	The list of adjacent edges for each node. It is similiar to adj.nodes while contains the edge ids instead of node ids.
node.pot	The node potentials. It is a matrix with dimension (n.nodes, max.state). Each row node.pot[i, ] denotes the node potentials of the i-th node.
edge.pot	The edge potentials. It is a list of n.edges matrixes. Each matrix edge.pot[[i]], with dimension (n.states[edges[i,1]], n.states[edges[i,2]]), denotes the edge potentials of the i-th edge.

**See Also**

[clamp.crf](#), [sub.crf](#)

## Examples

```
library(CRF)

nNodes <- 4
nStates <- 2

adj <- matrix(0, nrow=nNodes, ncol=nNodes)
for (i in 1:(nNodes-1))
{
  adj[i,i+1] <- 1
  adj[i+1,i] <- 1
}

crf <- make.crf(adj, nStates)

crf$node.pot[1,] <- c(1, 3)
crf$node.pot[2,] <- c(9, 1)
crf$node.pot[3,] <- c(1, 3)
crf$node.pot[4,] <- c(9, 1)

for (i in 1:crf$n.edges)
{
  crf$edge.pot[[i]][1,] <- c(2, 1)
  crf$edge.pot[[i]][2,] <- c(1, 2)
}
```

---

sample

*Sampling methods*

---

## Description

Generating samples from the distribution

## Usage

```
sample.exact(crf, size)
sample.chain(crf, size)
sample.tree(crf, size)
sample.conditional(crf, clamped, sample.method, ...)
sample.cutset(crf, cutset, size)
sample.gibbs(crf, size, burn.in = 1000,
             start = apply(crf$node.pot, 1, which.max), ...)
```

## Arguments

crf

## Details

- [sample.exact](#) Exact sampling for small graphs with brute-force inverse cumulative distribution



- [sample.chain](#) Exact sampling for chain-structured graphs with the forward-filter backward-sample algorithm
- [sample.tree](#) Exact sampling for tree- and forest-structured graphs with sum-product belief propagation and backward-sampling
- [sample.conditional](#) Conditional sampling (takes another sampling method as input)
- [sample.cutset](#) Exact sampling for graphs with a small cutset using cutset conditioning
- [sample.gibbs](#) Approximate sampling using a single-site Gibbs sampler

## See Also

[make.crf](#), [decode](#), [infer](#)

## Examples

```
library(CRF)
data(Small)
sample.exact(small.crf, 100)
sample.chain(small.crf, 100)
sample.tree(small.crf, 100)
sample.conditional(small.crf, c(0,1,0,0), sample.exact, 100)
sample.cutset(small.crf, c(2), 100)
sample.gibbs(small.crf, 100)
```

---

sub.crf

---

*Make sub CRF data structure*


---

## Description

Generate sub CRF data structure by selecting some nodes

## Usage

```
clamp.crf(crf, clamped)
```

## Arguments

crf	The CRF data structure generated by <a href="#">make.crf</a>
subset	The vector of selected node ids

## Details

The function will generate a CRF data structure from a given CRF data structure by selecting some nodes. The vector `subset` contains the node ids selected to generate the new CRF data. Unlike [clamp.crf](#), the potentials of remaining nodes and edges are untouched.

**Value**

The function will return a CRF data structure with additional components:

<code>original</code>	The original CRF data.
<code>node.id</code>	The vector of the original node ids for nodes in the new CRF data.
<code>node.map</code>	The vector of the new node ids for nodes in the original CRF data.
<code>edge.id</code>	The vector of the original edge ids for edges in the new CRF data.
<code>edge.map</code>	The vector of the new edge ids for edges in the original CRF data.

**See Also**

[make.crf](#), [clamp.crf](#)

**Examples**

```
library(CRF)
data(Small)
crf <- sub.crf(small.crf, c(2, 3))
```

# Index

**\*Topic decode**  
  decode, 4

**\*Topic infer**  
  infer, 5

**\*Topic make.crf**  
  clamp.crf, 3  
  make.crf, 7  
  sub.crf, 9

**\*Topic package**  
  CRF-package, 1

**\*Topic sample**  
  sample, 8

clamp.crf, 3, 7, 9, 10  
CRF (*CRF-package*), 1  
CRF-package, 1

decode, 3, 4, 6, 9  
decode.block, 5  
decode.chain, 2, 5  
decode.conditional, 2, 5  
decode.cutset, 2, 5  
decode.exact, 2, 5  
decode.greedy, 2, 5  
decode.icm, 2, 5  
decode.lbp, 2, 5  
decode.marginal, 2, 5  
decode.sample, 2, 5  
decode.trbp, 2, 5  
decode.tree, 2, 5

infer, 3, 5, 5, 9  
infer.chain, 2, 6  
infer.conditional, 2, 6  
infer.cutset, 2, 6  
infer.exact, 2, 6  
infer.lbp, 2, 6  
infer.sample, 2, 6  
infer.trbp, 2, 6  
infer.tree, 2, 6

make.crf, 3–6, 7, 9, 10

sample, 3, 5, 6, 8  
sample.chain, 2, 9  
sample.conditional, 2, 9  
sample.cutset, 2, 9  
sample.exact, 2, 8  
sample.gibbs, 2, 9  
sample.tree, 2, 9  
sub.crf, 4, 7, 9  
train, 3