Package ‘CRF’

February 19, 2015

Version 0.3-8
Title CRF - Conditional Random Fields
Description This package implements modeling and computational tools for conditional random fields model (CRF) as well as other probabilistic undirected graphical models of discrete data with pairwise and unary potentials.
ByteCompile TRUE
Depends R (>= 2.12.0)
Imports Rglpk (>= 0.3-5), Matrix (>= 1.1-2)
License GPL (>= 2)
URL http://r-forge.r-project.org/projects/crf/
Author Ling-Yun Wu [aut, cre]
Maintainer Ling-Yun Wu <wulingyun@gmail.com>
Repository CRAN
Repository/R-Forge/Project crf
Repository/R-Forge/Revision 27
Repository/R-Forge/DateTimeStamp 2014-07-30 10:49:44
Date/Publication 2014-07-30 15:58:38

R topics documented:

- CRF-package .................................................. 3
- Chain ..................................................... 5
- clamp.crf ................................................... 6
- clamp.reset ............................................... 7
- Clique .................................................... 8
- crf.nll .................................................... 8
- crf.update ............................................... 9
- decode.block ............................................ 10
R topics documented:

decode.chain ................................. 11
decode.conditional ........................... 12
decode.cutset ................................. 12
decode.exact ...................... ............................ 13
decode.greedy ................................. 14
decode.icm ................................. 14
decode.ilp ................................. 15
decode.junction ................................. 16
decode.lbp ................................. 16
decode.marginal ................................. 17
decode.sample ................................. 18
decode.trbp ................................. 18
decode.tree ................................. 19
duplicate.crf ................................. 20
get.logPotential ................................. 20
get.potential ................................. 21
infer.chain ................................. 22
infer.conditional ................................. 22
infer.cutset ................................. 23
infer.exact ................................. 24
infer.junction ................................. 25
infer.lbp ................................. 26
infer.sample ................................. 27
infer.trbp ................................. 28
infer.tree ................................. 29
Loop ................................. 29
make.crf ................................. 30
make.features ................................. 32
make.par ................................. 33
mrf.nll ................................. 33
mrf.stat ................................. 34
mrf.update ................................. 35
Rain ................................. 35
sample.chain ................................. 36
sample.conditional ................................. 37
sample.cutset ................................. 37
sample.exact ................................. 38
sample.gibbs ................................. 39
sample.junction ................................. 40
sample.tree ................................. 40
Small ................................. 41
sub.crf ................................. 42
train.crf ................................. 43
train.mrf ................................. 44
Tree ................................. 44

Index 46
CRF-package

CRF - Conditional Random Fields

Description

Library of Conditional Random Fields model

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.conditiona` Conditional decoding (takes another decoding method as input)
- `decode.cutset` Exact decoding for graphs with a small cutset using cutset conditioning
- `decode.junction` Exact decoding for low-treewidth graphs using junction trees
- `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.1bp` Approximate decoding using max-product loopy belief propagation
- `decode.trbp` Approximate decoding using max-product tree-rewighted 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
- `decode.ilp` Exact decoding with an integer linear programming formulation and approximate using LP relaxation

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.contitional` Conditional inference (takes another inference method as input)
• `infer.cutset` Exact inference for graphs with a small cutset using cutset conditioning
• `infer.junction` Exact decoding for low-treewidth graphs using junction trees
• `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.contitional` Conditional sampling (takes another sampling method as input)
• `sample.cutset` Exact sampling for graphs with a small cutset using cutset conditioning
• `sample.junction` Exact sampling for low-treewidth graphs using junction trees
• `sample.gibbs` Approximate sampling using a single-site Gibbs sampler

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

• `train.crf` Train CRF model
• `train.mrf` Train MRF model

Tools: Tools for building and manipulating CRF data

• `make.crf` Generate CRF from the adjacent matrix
• `make.features` Make the data structure of CRF features
• `make.par` Make the data structure of CRF parameters
• `duplicate.crf` Duplicate an existing CRF
• `clamp.crf` Generate clamped CRF by fixing the states of some nodes
• `clamp.reset` Reset clamped CRF by changing the states of clamped nodes
• `sub.crf` Generate sub CRF by selecting some nodes
• `mrf.update` Update node and edge potentials of MRF model
• `crf.update` Update node and edge potentials of CRF model

Author(s)

Ling-Yun Wu <wulingyun@gmail.com>
Chain CRF example

Description

This data set gives a chain CRF example

Usage

data(Chain)

Format

A list containing two elements:

- crf The CRF
- answer A list of 4 elements:
  - decode The most likely configuration
  - node.bel The node belief
  - edge.bel The edge belief
  - logZ The logarithmic value of CRF normalization factor Z

References


Examples

library(CRF)
data(Small)
decode.exact(Small$crf)
infer.exact(Small$crf)
sample.exact(Small$crf, 100)
clamp.crf  

Make clamped CRF

Description

Generate clamped CRF by fixing the states of some nodes

Usage

clamp.crf(crf, clamped)

Arguments

crf  
The CRF generated by make.crf
clamped  
The vector of fixed states of nodes

Details

The function will generate a clamped CRF from a given CRF 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 new CRF with additional components:

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

See Also

make.crf, sub.crf, clamp.reset

Examples

library(CRF)
data(Small)
crf <- clamp.crf(Small$crf, c(0, 0, 1, 1))
clamp.reset

Reset clamped CRF

Description

Reset clamped CRF by changing the states of clamped nodes

Usage

clamp.reset(crf, clamped)

Arguments

crf            The clamped CRF generated by clamp.crf
clamped        The vector of fixed states of nodes

Details

The function will reset a clamped CRF by changing the states of fixed 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 the same clamped CRF.

See Also

make.crf, clamp.crf

Examples

library(CRF)
data(Small)
crf <- clamp.crf(Small$crf, c(0, 0, 1, 1))
clamp.reset(crf, c(0,0,2,2))
Clique  

_Clique CRF example_

**Description**

This data set gives a clique CRF example

**Usage**

data(Clique)

**Format**

A list containing two elements:

- **crf** The CRF
- **answer** A list of 4 elements:
  - **decode** The most likely configuration
  - **node.bel** The node belief
  - **edge.bel** The edge belief
  - **logZ** The logarithmic value of CRF normalization factor Z

**crf.nll**  

_Calculate CRF negative log likelihood_

**Description**

Calculate the negative log likelihood of CRF model

**Usage**

crf.nll(par, crf, instances, node.fea = NULL, edge.fea = NULL, node.ext = NULL, edge.ext = NULL, infer.method = infer.chain, ...)

**Arguments**

- **crf**  
  The CRF
- **par**  
  The parameter vector of CRF
- **instances**  
  The training data matrix of CRF model
- **node.fea**  
  The list of node features
- **edge.fea**  
  The list of edge features
- **node.ext**  
  The list of extended information of node features
- **edge.ext**  
  The list of extended information of edge features
- **infer.method**  
  The inference method used to compute the likelihood
- **...**  
  Other parameters need by the inference method
**crf.update**

**Details**

This function calculates the negative log likelihood of CRF model as well as the gradient. This function is intended to be called by optimization algorithm in training process.

In the training data matrix instances, each row is an instance and each column corresponds a node in CRF. The variables node.fea, edge.fea, node.ext, edge.ext are lists of length equal to the number of instances, and their elements are defined as in crf.update respectively.

**Value**

This function will return the value of CRF negative log-likelihood.

**See Also**

crf.update, train.crf

---

<table>
<thead>
<tr>
<th>crf.update</th>
<th>Update CRF potentials</th>
</tr>
</thead>
</table>

**Description**

Update node and edge potentials of CRF model

**Usage**

```r
crf.update(crf, node.fea = NULL, edge.fea = NULL, node.ext = NULL, edge.ext = NULL)
```

**Arguments**

- **crf**  
  The CRF

- **node.fea**  
  The node features matrix with dimension (n.nf, n.nodes)

- **edge.fea**  
  The edge features matrix with dimension (n.ef, n.edges)

- **node.ext**  
  The extended information of node features

- **edge.ext**  
  The extended information of edge features

**Details**

This function updates node.pot and edge.pot of CRF model by using the current values of parameters and features.

There are two ways to model the relationship between parameters and features. The first one exploits the special structure of features to reduce the memory usage. However it may not suitable for all circumstances. The other one is more straightforward by explicitly specifying the coefficients of each parameter to calculate the potentials, and may use much more memory. Two approaches can be used together.
The first way uses the objects `node.par` and `edge.par` to define the structure of features and provides the feature information in variables `node.fea` and `edge.fea`. The second way directly provides the feature information in variables `node.ext` and `edge.ext` without any prior assumption on feature structure. `node.ext` is a list and each element has the same structure as `node.pot`. `edge.ext` is a list and each element has the same structure as `edge.pot`.

In detail, the node potential is updated as follows:

\[
node.pot[n, i] = \sum_f par[node.par[n, i, f]] * node.fea[f, n] + \sum_k par[k] * node.ext[[k]][n, i]
\]

and the edge potential is updated as follows:

\[
edge.pot[[e]][i, j] = \sum_f par[edge.par[[e]][i, j, f]] * edge.fea[f, e] + \sum_k par[k] * edge.ext[[k]][[e]][i, j]
\]

**Value**

This function will directly modify the CRF and return the same CRF.

**See Also**

`crf.nll`, `train.crf`

---

**decode.block**

Decoding method using block iterated conditional modes algorithm

**Description**

Computing the most likely configuration for CRF

**Usage**

`decode.block(crf, blocks, decode.method = decode.tree, restart = 0, start = apply(crf$node.pot, 1, which.max), ...)`

**Arguments**

- `crf` The CRF
- `blocks` A list of vectors, each vector containing the nodes in a block
- `decode.method` The decoding method to solve the clamped CRF
- `restart` Non-negative integer to control how many restart iterations are repeated
- `start` An initial configuration, a good start will significantly reduce the searching time
- `...` The parameters for `decode.method`
decode.chain

Details

Approximate decoding with the block iterated conditional modes algorithm

Value

This function will return the most likely configuration, which is a vector of length crf$n.nodes.

Examples

```r
library(CRF)
data(Small)
d <- decode.block(Small$crf, list(c(1,3), c(2,4)))
```

---

decode.chain

Decoding method for chain-structured graphs

Description

Computing the most likely configuration for CRF

Usage

```r
decode.chain(crf)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crf</td>
<td>The CRF</td>
</tr>
</tbody>
</table>

Details

Exact decoding for chain-structured graphs with the Viterbi algorithm.

Value

This function will return the most likely configuration, which is a vector of length crf$n.nodes.

Examples

```r
library(CRF)
data(Small)
d <- decode.chain(Small$crf)
```
decode.conditionald  Conditional decoding method

Description
Computing the most likely configuration for CRF

Usage
decode.conditional(crf, clamped, decode.method, ...)

Arguments
- crf: The CRF
- clamped: The vector of fixed values for clamped nodes, 0 for unfixed nodes
- decode.method: The decoding method to solve clamped CRF
- ...: The parameters for decode.method

Details
Conditional decoding (takes another decoding method as input)

Value
This function will return the most likely configuration, which is a vector of length crf$n.nodes.

Examples
library(CRF)
data(small)
d <- decode.conditional(small$crf, c(0,1,0,0), decode.exact)

deceive.cutset Decoding method for graphs with a small cutset

Description
Computing the most likely configuration for CRF

Usage
decode.cutset(crf, cutset, engine = "default", start = apply(crf$node.pot, 1, which.max))
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crf</td>
<td>The CRF</td>
</tr>
<tr>
<td>cutset</td>
<td>A vector of nodes in the cutset</td>
</tr>
<tr>
<td>engine</td>
<td>The underlying engine for cutset decoding, possible values are &quot;default&quot;, &quot;none&quot;, &quot;exact&quot;, &quot;chain&quot;, and &quot;tree&quot;.</td>
</tr>
<tr>
<td>start</td>
<td>An initial configuration, a good start will significantly reduce the searching time</td>
</tr>
</tbody>
</table>

Details

Exact decoding for graphs with a small cutset using cutset conditioning

Value

This function will return the most likely configuration, which is a vector of length `crf$n.nodes`.

Examples

```r
crtn <- library(CRF)
data(Small)
d <- decode.cutset(Small$crf, c(2))
```

---

Description

Computing the most likely configuration for CRF

Usage

```r
decode.exact(crf)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crf</td>
<td>The CRF</td>
</tr>
</tbody>
</table>

Details

Exact decoding for small graphs with brute-force search

Value

This function will return the most likely configuration, which is a vector of length `crf$n.nodes`.

Examples

```r
crtn <- library(CRF)
data(Small)
d <- decode.exact(Small$crf)
```
decode.greedy  Decoding method using greedy algorithm

Description
Computing the most likely configuration for CRF

Usage
decode.greedy(crf, restart = 0, start = apply(crf$node.pot, 1, which.max))

Arguments
- crf: The CRF
- restart: Non-negative integer to control how many restart iterations are repeated
- start: An initial configuration, a good start will significantly reduce the searching time

Details
Approximate decoding with greedy algorithm

Value
This function will return the most likely configuration, which is a vector of length crf$n.nodes.

Examples
library(CRF)
data(Small)
d <- decode.greedy(Small$crf)

decode.icm  Decoding method using iterated conditional modes algorithm

Description
Computing the most likely configuration for CRF

Usage
decode.icm(crf, restart = 0, start = apply(crf$node.pot, 1, which.max))

Arguments
- crf: The CRF
- restart: Non-negative integer to control how many restart iterations are repeated
- start: An initial configuration, a good start will significantly reduce the searching time
Details

Approximate decoding with the iterated conditional modes algorithm

Value

This function will return the most likely configuration, which is a vector of length \( \text{crf}$n.\ nodes \).

Examples

```r
library(CRF)
data(Small)
d <- decode.icm(Small$crf)
```

---

**decode.ilp**  
Decoding method using integer linear programming

Description

Computing the most likely configuration for CRF

Usage

```r
decode.ilp(crf, lp.rounding = FALSE)
```

Arguments

- **crf**  The CRF
- **lp.rounding**  Boolean variable to indicate whether LP rounding is need.

Details

Exact decoding with an integer linear programming formulation and approximate using LP relaxation

Value

This function will return the most likely configuration, which is a vector of length \( \text{crf}$n.\ nodes \).

Examples

```r
library(CRF)
data(Small)
d <- decode.ilp(Small$crf)
```
**decode.junction**  
*Decoding method for low-treewidth graphs*

**Description**
Computing the most likely configuration for CRF

**Usage**
```r
decode.junction(crf)
```

**Arguments**
- `crf`  The CRF

**Details**
Exact decoding for low-treewidth graphs using junction trees

**Value**
This function will return the most likely configuration, which is a vector of length `crf$n_nodes`.

**Examples**
```r
library(crf)
data(small)
d <- decode.junction(small$crf)
```

---

**decode.lbp**  
*Decoding method using loopy belief propagation*

**Description**
Computing the most likely configuration for CRF

**Usage**
```r
decode.lbp(crf, max.iter = 10000, cutoff = 1e-04, verbose = 0)
```

**Arguments**
- `crf`  The CRF
- `max.iter`  The maximum allowed iterations of termination criteria
- `cutoff`  The convergence cutoff of termination criteria
- `verbose`  Non-negative integer to control the tracing information in algorithm
Details

Approximate decoding using max-product loopy belief propagation

Value

This function will return the most likely configuration, which is a vector of length `crf$n.nodes`.

Examples

```r
library(CRF)
data(Small)
d <- decode.lbp(Small$crf)
```

---

### decode.marginal

**Decoding method using inference**

**Description**

Computing the most likely configuration for CRF

**Usage**

```r
decode.marginal(crf, infer.method, ...)
```

**Arguments**

- `crf`: The CRF
- `infer.method`: The inference method
- `...`: The parameters for `infer.method`

**Details**

Approximate decoding using inference (takes an inference method as input)

**Value**

This function will return the most likely configuration, which is a vector of length `crf$n.nodes`.

**Examples**

```r
library(CRF)
data(Small)
d <- decode.marginal(Small$crf, infer.exact)
```
decode.sample  Decoding method using sampling

Description
Computing the most likely configuration for CRF

Usage
decode.sample(crf, sample.method, ...)

Arguments
- crf: The CRF
- sample.method: The sampling method
- ...: The parameters for sample.method

Details
Approximate decoding using sampling (takes a sampling method as input)

Value
This function will return the most likely configuration, which is a vector of length crf$n.nodes.

Examples
library(CRF)
data(Small)
d <- decode.sample(Small$crf, sample.exact, 10000)

decode.trbp  Decoding method using tree-reweighted belief propagation

Description
Computing the most likely configuration for CRF

Usage
decode.trbp(crf, max.iter = 10000, cutoff = 1e-04, verbose = 0)
Arguments

crf       The CRF
max.iter  The maximum allowed iterations of termination criteria
cutoff    The convergence cutoff of termination criteria
verbose   Non-negative integer to control the tracing information in algorithm

Details

Approximate decoding using max-product tree-reweighted belief propagation

Value

This function will return the most likely configuration, which is a vector of length crf$n.nodes.

Examples

library(crf)
data(Small)
d <- decode.trbp(Small$crf)

decode.tree  Decoding method for tree- and forest-structured graphs

Description

Computing the most likely configuration for CRF

Usage

decode.tree(crf)

Arguments

crf       The CRF

Details

Exact decoding for tree- and forest-structured graphs with max-product belief propagation

Value

This function will return the most likely configuration, which is a vector of length crf$n.nodes.

Examples

library(CRF)
data(Small)
d <- decode.tree(Small$crf)
**duplicate.crf**  
*Duplicate CRF*

**Description**

Duplicate an existing CRF

**Usage**

duplicate.crf(crf)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crf</td>
<td>The existing CRF</td>
</tr>
</tbody>
</table>

**Details**

This function will duplicate an existing CRF. Since CRF is implemented as an environment, normal assignment will only copy the pointer instead of the real data. This function will generate a new CRF and really copy all data.

**Value**

The function will return a new CRF with copied data

**See Also**

make.crf

---

**get.logPotential**  
*Calculate the log-potential of CRF*

**Description**

Calculate the logarithmic potential of a CRF with given configuration

**Usage**

get.logPotential(crf, configuration)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crf</td>
<td>The CRF</td>
</tr>
<tr>
<td>configuration</td>
<td>The vector of states of nodes</td>
</tr>
</tbody>
</table>
get.potential

Details

The function will calculate the logarithmic potential of a CRF with given configuration, i.e., the assigned states of nodes in the CRF.

Value

The function will return the log-potential of CRF with given configuration

See Also

get.potential

get.potential  Calculate the potential of CRF

Description

Calculate the potential of a CRF with given configuration

Usage

get.potential(crf, configuration)

Arguments

crf  The CRF
configuration  The vector of states of nodes

Details

The function will calculate the potential of a CRF with given configuration, i.e., the assigned states of nodes in the CRF.

Value

The function will return the potential of CRF with given configuration

See Also

get.logPotential
infer.chain

Inference method for chain-structured graphs

Description
Computing the partition function and marginal probabilities

Usage
infer.chain(crf)

Arguments

- **crf**  The CRF

Details
Exact inference for chain-structured graphs with the forward-backward algorithm

Value

This function will return a list with components:

- **node.bel**  Node belief. It is a matrix with `crf$n.nodes` rows and `crf$max.state` columns.
- **edge.bel**  Edge belief. It is a list of matrices. The size of list is `crf$n.edges` and the matrix `i` has `crf$n.states[crf$edges[i,1]]` rows and `crf$n.states[crf$edges[i,2]]` columns.
- **logZ**  The logarithmic value of CRF normalization factor Z.

Examples

```r
library(CRF)
data(Small)
i <- infer.chain(Small$crf)
```

infer.conditional

Conditional inference method

Description
Computing the partition function and marginal probabilities

Usage

infer.conditional(crf, clamped, infer.method, ...)

---

**infer.chain**  

Inference method for chain-structured graphs

**Description**

Computing the partition function and marginal probabilities

**Usage**

```
infer.chain(crf)
```

**Arguments**

- **crf**  The CRF

**Details**

Exact inference for chain-structured graphs with the forward-backward algorithm

**Value**

This function will return a list with components:

- **node.bel**  Node belief. It is a matrix with `crf$n.nodes` rows and `crf$max.state` columns.
- **edge.bel**  Edge belief. It is a list of matrices. The size of list is `crf$n.edges` and the matrix `i` has `crf$n.states[crf$edges[i,1]]` rows and `crf$n.states[crf$edges[i,2]]` columns.
- **logZ**  The logarithmic value of CRF normalization factor Z.

**Examples**

```r
library(CRF)
data(Small)
i <- infer.chain(Small$crf)
```

**infer.conditional**  

Conditional inference method

**Description**

Computing the partition function and marginal probabilities

**Usage**

```
infer.conditional(crf, clamped, infer.method, ...)
```
Arguments

\begin{itemize}
  \item \texttt{crf} \hspace{1cm} The CRF
  \item \texttt{clamped} \hspace{1cm} The vector of fixed values for clamped nodes, 0 for unfixed nodes
  \item \texttt{infer.method} \hspace{1cm} The inference method to solve the clamped CRF
  \item \ldots \hspace{1cm} The parameters for \texttt{infer.method}
\end{itemize}

Details

Conditional inference (takes another inference method as input)

Value

This function will return a list with components:

\begin{itemize}
  \item \texttt{node.bel} \hspace{1cm} Node belief. It is a matrix with \texttt{crf$n.nodes} rows and \texttt{crf$max.state} columns.
  \item \texttt{edge.bel} \hspace{1cm} Edge belief. It is a list of matrices. The size of list is \texttt{crf$n.edges} and the matrix \texttt{\texttt{\texttt{i}}} has \texttt{crf$n.states[crf$edges[i,1]]} rows and \texttt{crf$n.states[crf$edges[i,2]]} columns.
  \item \texttt{logZ} \hspace{1cm} The logarithmic value of CRF normalization factor $Z$.
\end{itemize}

Examples

\begin{verbatim}
library(CRF)
data(Small)
i <- infer.cutset(Small$crf, c(0,1,0,0), infer.exact)
\end{verbatim}

\begin{tabular}{ll}
\hline
\textbf{infer.cutset} & \textit{Inference method for graphs with a small cutset} \\
\hline
\end{tabular}

Description

Computing the partition function and marginal probabilities

Usage

\begin{verbatim}
infer.cutset(crf, cutset, engine = "default")
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{crf} \hspace{1cm} The CRF
  \item \texttt{cutset} \hspace{1cm} A vector of nodes in the cutset
  \item \texttt{engine} \hspace{1cm} The underlying engine for cutset decoding, possible values are "default", "none", "exact", "chain", and "tree".
\end{itemize}
Details

Exact inference for graphs with a small cutset using cutset conditioning

Value

This function will return a list with components:

- **node.bel**: Node belief. It is a matrix with \( \text{crf$n$nodes} \) rows and \( \text{crf$max.state} \) columns.
- **edge.bel**: Edge belief. It is a list of matrices. The size of list is \( \text{crf$n$edges} \) and the matrix \( i \) has \( \text{crf$n.states[crf$edges[i,1]]} \) rows and \( \text{crf$n.states[crf$edges[i,2]]} \) columns.
- **logZ**: The logarithmic value of CRF normalization factor \( Z \).

Examples

```r
library(crf)
data(small)
i <- infer.cutset(small$crf, c(R))
```

---

**infer.exact**  
Inference method for small graphs

Description

Computing the partition function and marginal probabilities

Usage

`infer.exact(crf)`

Arguments

- **crf**: The CRF

Details

Exact inference for small graphs with brute-force counting

Value

This function will return a list with components:

- **node.bel**: Node belief. It is a matrix with \( \text{crf$n$nodes} \) rows and \( \text{crf$max.state} \) columns.
- **edge.bel**: Edge belief. It is a list of matrices. The size of list is \( \text{crf$n$edges} \) and the matrix \( i \) has \( \text{crf$n.states[crf$edges[i,1]]} \) rows and \( \text{crf$n.states[crf$edges[i,2]]} \) columns.
- **logZ**: The logarithmic value of CRF normalization factor \( Z \).
Examples

```r
library(crf)
data(Small)
i <- infer.exact(Small$crf)
```

infer.junction  
*Inference method for low-treewidth graphs*

**Description**

Computing the partition function and marginal probabilities

**Usage**

```r
infer.junction(crf)
```

**Arguments**

- `crf`  
The CRF

**Details**

Exact decoding for low-treewidth graphs using junction trees

**Value**

This function will return a list with components:

- `node.bel`  
  Node belief. It is a matrix with `crf$n.nodes` rows and `crf$max.state` columns.

- `edge.bel`  
  Edge belief. It is a list of matrices. The size of list is `crf$n.edges` and the matrix `i` has `crf$n.states[crf$edges[i,1]]` rows and `crf$n.states[crf$edges[i,2]]` columns.

- `logZ`  
  The logarithmic value of CRF normalization factor $Z$.

**Examples**

```r
library(crf)
data(Small)
i <- infer.junction(Small$crf)
```
infer.lbp

Inference method using loopy belief propagation

Description

Computing the partition function and marginal probabilities

Usage

infer.lbp(crf, max.iter = 10000, cutoff = 1e-04, verbose = 0)

Arguments

crf
The CRF

max.iter
The maximum allowed iterations of termination criteria

cutoff
The convergence cutoff of termination criteria

verbose
Non-negative integer to control the tracing information in algorithm

Details

Approximate inference using sum-product loopy belief propagation

Value

This function will return a list with components:

node.bel
Node belief. It is a matrix with crf$n.nodes rows and crf$max.state columns.

edge.bel
Edge belief. It is a list of matrices. The size of list is crf$n.edges and the matrix i has crf$n.states[crf$edges[i,1]] rows and crf$n.states[crf$edges[i,2]] columns.

logZ
The logarithmic value of CRF normalization factor Z.

Examples

library(CRF)
data(Small)
i <- infer.lbp(Small$crf)
infer.sample

Inference method using sampling

Description
Computing the partition function and marginal probabilities

Usage
infer.sample(crf, sample.method, ...)

Arguments
- crf: The CRF
- sample.method: The sampling method
- ...: The parameters for sample.method

Details
Approximate inference using sampling (takes a sampling method as input)

Value
This function will return a list with components:

- node.bel: Node belief. It is a matrix with crf$n.nodes rows and crf$max.state columns.
- edge.bel: Edge belief. It is a list of matrices. The size of list is crf$n.edges and the matrix i has crf$n.states[crf$edges[i,1]] rows and crf$n.states[crf$edges[i,2]] columns.
- logZ: The logarithmic value of CRF normalization factor Z.

Examples
library(CRF)
data(Small)
i <- infer.sample(Small$crf, sample.exact, 10000)
infer.trbp  

Inference method using tree-reweighted belief propagation

Description
Computing the partition function and marginal probabilities

Usage
infer.trbp(crfL maxNiter ] 1PPPPL cutoff ] 1eMPTL verbose ] P)

Arguments
crf  The CRF
max.iter  The maximum allowed iterations of termination criteria
cutoff  The convergence cutoff of termination criteria
verbose  Non-negative integer to control the tracing information in algorithm

Details
Approximate inference using sum-product tree-reweighted belief propagation

Value
This function will return a list with components:

node.bel  Node belief. It is a matrix with crf$n.nodes rows and crf$max.state columns.
edge.bel  Edge belief. It is a list of matrices. The size of list is crf$n.edges and the matrix $ has crf$n.states[crf$edges[i,1]] rows and crf$n.states[crf$edges[i,2]] columns.
logZ  The logarithmic value of CRF normalization factor Z.

Examples
library(CRF)
data(Small)
i <- infer.trbp(Small$crf)
inference method for tree- and forest-structured graphs

Description
Computing the partition function and marginal probabilities

Usage
infer.tree(crf)

Arguments

- crf: The CRF

Details
Exact inference for tree- and forest-structured graphs with sum-product belief propagation

Value
This function will return a list with components:

- node.bel: Node belief. It is a matrix with `crf$n.nodes` rows and `crf$max.state` columns.
- edge.bel: Edge belief. It is a list of matrices. The size of list is `crf$n.edges` and the matrix `i` has `crf$n.states[crf$edges[i,1]]` rows and `crf$n.states[crf$edges[i,2]]` columns.
- logZ: The logarithmic value of CRF normalization factor Z.

Examples

```r
library(CRF)
data(Small)
i <- infer.tree(Small$crf)
```

Loop CRF example

Description
This data set gives a loop CRF example

Usage
data(Loop)
Format

A list containing two elements:

- crf The CRF
- answer A list of 4 elements:
  - decode The most likely configuration
  - node.bel The node belief
  - edge.bel The edge belief
  - logz The logarithmic value of CRF normalization factor Z

Description

Generate CRF from the adjacent matrix

Usage

make.crf(adj.matrix = NULL, n.states = 2, n.nodes = 2)

Arguments

adj.matrix The adjacent matrix of CRF network.
n.states The state numbers of nodes.
n.nodes The number of nodes, which is only used to generate linear chain CRF when adj.matrix is NULL.

Details

The function will generate an empty CRF 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.

Since the CRF data are often very huge, CRF is implemented as an environment. The assignment of environments will only copy the addresses instead of real data, therefore the variables using normal assignment will refer to the exactly same CRF. For complete duplication of the data, please use duplicate.crfr.

Value

The function will return a new CRF, which is an environment 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 similar 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.

duplicateNcrf, clampNcrf, subNcrf

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[1][i][1,] <- c(2, 1)  
  crf$edge.pot[1][i][2,] <- c(1, 2)  
}
**make.features**  
*Make CRF features*

---

**Description**

Make the data structure of CRF features

**Usage**

```r
make.features(crf, n.nf = 1, n.ef = 1)
```

**Arguments**

- `crf` The CRF
- `n.nf` The number of node features
- `n.ef` The number of edge features

**Details**

This function makes the data structure of features need for modeling and training CRF. The parameters `n.nf` and `n.ef` specify the number of node and edge features, respectively.

The objects `node.par` and `edge.par` define the corresponding parameters used with each feature. `node.par` is a 3-dimensional arrays, and element `node.par[n,i,f]` is the index of parameter associated with the corresponding node potential `node.pot[n,i]` and node feature `f`. `edge.par` is a list of 3-dimensional arrays, and element `edge.par[[e]][i,j,f]` is the index of parameter associated with the corresponding edge potential `edge.pot[[e]][i,j]` and edge feature `f`. The value 0 is used to indicate the corresponding node or edge potential does not depend on that feature.

For detail of calculation of node and edge potentials from features and parameters, please see `crf.update`.

**Value**

This function will directly modify the CRF and return the same CRF.

**See Also**

`crf.update, make.par, make.crf`
make.par

Make CRF parameters

Description
Make the data structure of CRF parameters

Usage
make.par(crf, n.par = 1)

Arguments
- crf: The CRF
- n.par: The number of parameters

Details
This function makes the data structure of parameters need for modeling and training CRF. The parameters are stored in par, which is a numeric vector of length n.par.

Value
This function will directly modify the CRF and return the same CRF.

See Also
crf.update, make.features, make.crf

mrf.nll
Calculate MRF negative log-likelihood

Description
Calculate the negative log-likelihood of MRF model

Usage
mrf.nll(par, crf, instances, infer.method = infer.chain, ...)

Arguments
- crf: The CRF
- par: The parameter vector of CRF
- instances: The training data matrix of MRF model
- infer.method: The inference method used to compute the likelihood
- ...: Other parameters need by the inference method
Details

This function calculates the negative log-likelihood of MRF model as well as the gradient. This function is intended to be called by optimization algorithm in training process. Before calling this function, the MRF sufficient statistics must be calculated and stored in object par.stat of CRF. In the training data matrix instances, each row is an instance and each column corresponds a node in CRF.

Value

This function will return the value of MRF negative log-likelihood.

See Also

mrf.stat, mrf.update, train.mrf
mrf.update  

Update MRF potentials

Description
Update node and edge potentials of MRF model

Usage
mrf.update(crf)

Arguments
crf  The CRF

Details
The function updates node.pot and edge.pot of MRF model.

Value
This function will directly modify the CRF and return the same CRF.

See Also
mrf.nll, train.mrf

---

Rain  

Rain data

Description
This data set gives an example of rain data used to train CRF and MRF models

Usage
data(Rain)

Format
A list containing two elements:

- rain A matrix of 28 columns containing raining data (1: rain, 2: sunny). Each row is an instance of 28 days for one month.
- months A vector containing the months of each instance.
References


sample.chain       Sampling method for chain-structured graphs

Description

Generating samples from the distribution

Usage

sample.chain(crf, size)

Arguments

crf       The CRF
size      The sample size

Details

Exact sampling for chain-structured graphs with the forward-filter backward-sample algorithm

Value

This function will return a matrix with size rows and crf$n nodes columns, in which each row is a sampled configuration.

Examples

library(CRF)
data(Small)
s <- sample.chain(Small$crf, 100)
**sample.conditional**  
*Conditional sampling method*

**Description**
Generating samples from the distribution

**Usage**
```r
sample.conditional(crf, size, clamped, sample.method, ...)
```

**Arguments**
- `crf`: The CRF
- `size`: The sample size
- `clamped`: The vector of fixed values for clamped nodes, 0 for unfixed nodes
- `sample.method`: The sampling method to solve the clamped CRF
- `...`: The parameters for `sample.method`

**Details**
Conditional sampling (takes another sampling method as input)

**Value**
This function will return a matrix with `size` rows and `crf$n.nodes` columns, in which each row is a sampled configuration.

**Examples**
```r
library(CRF)
data(Small)
s <- sample.conditional(Small$crf, 100, c(0,1,0,0), sample.exact)
```

---

**sample.cutset**  
*Sampling method for graphs with a small cutset*

**Description**
Generating samples from the distribution

**Usage**
```r
sample.cutset(crf, size, cutset, engine = "default")
```
Arguments

- crf: The CRF
- size: The sample size
- cutset: A vector of nodes in the cutset
- engine: The underlying engine for cutset sampling, possible values are "default", "none", "exact", "chain", and "tree".

Details

Exact sampling for graphs with a small cutset using cutset conditioning

Value

This function will return a matrix with size rows and crf$n.nodes columns, in which each row is a sampled configuration.

Examples

```r
library(crf)
data(small)
s <- sample.cutset(small$crf, 100, c(2))
```

---

### sample.exact

**Sampling method for small graphs**

Description

Generating samples from the distribution

Usage

```r
sample.exact(crf, size)
```

Arguments

- crf: The CRF
- size: The sample size

Details

Exact sampling for small graphs with brute-force inverse cumulative distribution

Value

This function will return a matrix with size rows and crf$n.nodes columns, in which each row is a sampled configuration.
Examples

```r
library(CRF)
data(Small)
s <- sample.exact(Small$crf, 100)
```

---

**sample.gibbs**

*Sampling method using single-site Gibbs sampler*

**Description**

Generating samples from the distribution

**Usage**

```r
sample.gibbs(crf, size, burn.in = 1000, start = apply(crf$node.pot, 1, which.max))
```

**Arguments**

- `crf` The CRF
- `size` The sample size
- `burn.in` The number of samples at the beginning that will be discarded
- `start` An initial configuration

**Details**

Approximate sampling using a single-site Gibbs sampler

**Value**

This function will return a matrix with `size` rows and `crf$n.nodes` columns, in which each row is a sampled configuration.

**Examples**

```r
library(CRF)
data(Small)
s <- sample.gibbs(Small$crf, 100)
```
**sample.junction**  
**Sampling method for low-treewidth graphs**

**Description**

Generating samples from the distribution

**Usage**

```r
sample.junction(crf, size)
```

**Arguments**

- `crf`  
  The CRF
- `size`  
  The sample size

**Details**

Exact sampling for low-treewidth graphs using junction trees

**Value**

This function will return a matrix with `size` rows and `crf$n` nodes columns, in which each row is a sampled configuration.

**Examples**

```r
library(CRF)
data(Small)
s <- sample.junction(Small$crf, 100)
```

---

**sample.tree**  
**Sampling method for tree- and forest-structured graphs**

**Description**

Generating samples from the distribution

**Usage**

```r
sample.tree(crf, size)
```

**Arguments**

- `crf`  
  The CRF
- `size`  
  The sample size
Details

Exact sampling for tree- and forest-structured graphs with sum-product belief propagation and backward-sampling.

Value

This function will return a matrix with size rows and crf$n$ nodes columns, in which each row is a sampled configuration.

Examples

```r
library(crf)
data(Small)
s <- sample.tree(Small$crf, 100)
```

Description

This data set gives a small CRF example.

Usage

data(Small)

Format

A list containing two elements:

- crf The CRF
- answer A list of 4 elements:
  - decode The most likely configuration
  - node.bel The node belief
  - edge.bel The edge belief
  - logz The logarithmic value of CRF normalization factor Z
**sub.crf**  
*Make sub CRF*

**Description**
Generate sub CRF by selecting some nodes

**Usage**
`sub.crf(crf, subset)`

**Arguments**
- `crf` The CRF generated by `make.crf`
- `subset` The vector of selected node ids

**Details**
The function will generate a new CRF from a given CRF by selecting some nodes. The vector `subset` contains the node ids selected to generate the new CRF. Unlike `clamp.crf`, the potentials of remaining nodes and edges are untouched.

**Value**
The function will return a new CRF with additional components:
- `original` The original CRF data.
- `node.id` The vector of the original node ids for nodes in the new CRF.
- `node.map` The vector of the new node ids for nodes in the original CRF.
- `edge.id` The vector of the original edge ids for edges in the new CRF.
- `edge.map` The vector of the new edge ids for edges in the original CRF.

**See Also**
`make.crf, clamp.crf`

**Examples**
```
library(CRF)
data(Small)
crf <- sub.crf(Small$crf, c(2, 3))
```
train.crf

Train CRF model

Description

Train the CRF model to estimate the parameters

Usage

train.crf(crf, instances, node.fea = NULL, edge.fea = NULL,
          node.ext = NULL, edge.ext = NULL, nll = crf.nll, trace = 0)

Arguments

crf  The CRF
instances  The training data matrix of CRF model
node.fea  The list of node features
dge.fea  The list of edge features
node.ext  The list of extended information of node features
dge.ext  The list of extended information of edge features
nll  The function to calculate negative log likelihood
trace  Non-negative integer to control the tracing information of the optimization process

Details

This function train the CRF model.

In the training data matrix instances, each row is an instance and each column corresponds a node in CRF. The variables node.fea, edge.fea, node.ext, edge.ext are lists of length equal to the number of instances, and their elements are defined as in crf.update respectively.

Value

This function will directly modify the CRF and return the same CRF.

See Also

crf.update, crf.nll, make.crf
train.mrf  

Train MRF model

Description

Train the MRF model to estimate the parameters.

Usage

train.mrf(crf, instances, nll = mrf.nll, trace = 0)

Arguments

crf          The CRF
instances     The training data matrix of CRF model
nll           The function to calculate negative log likelihood
trace         Non-negative integer to control the tracing information of the optimization process

Details

This function trains the Markov Random Fields (MRF) model, which is a simple variant of CRF model.
In the training data matrix instances, each row is an instance and each column corresponds a node in CRF.

Value

This function will directly modify the CRF and return the same CRF.

See Also

mrf.update, mrf.stat, mrf.nll, make.crf

Tree  

Tree CRF example

Description

This data set gives a tree CRF example.

Usage

data(Tree)
**Format**

A list containing two elements:

- `crf` The CRF
- `answer` A list of 4 elements:
  - `decode` The most likely configuration
  - `node.bel` The node belief
  - `edge.bel` The edge belief
  - `logZ` The logarithmic value of CRF normalization factor Z
Index

*Topic **datasets**
  Chain, 5
  Clique, 8
  Loop, 29
  Rain, 35
  Small, 41
  Tree, 44

*Topic **package**
  CRF-package, 3

Chain, 5
clamp.crf, 4, 6, 7, 31, 42
clamp.reset, 4, 6, 7
Clique, 8
CRF (CRF-package), 3
CRF-package, 3
crf.nll, 8, 10, 43
crf.update, 4, 9, 9, 32, 33, 43

decode.block, 3, 10
decode.chain, 3, 11
decode.conditional, 3, 12
decode.cutset, 3, 12
decode.exact, 3, 13
decode.greedy, 3, 14
decode.icm, 3, 14
decode.ilp, 3, 15
decode.junction, 3, 16
decode.lbp, 3, 16
decode.marginal, 3, 17
decode.sample, 3, 18
decode.trbp, 3, 18
decode.tree, 3, 19
duplicate.crf, 4, 20, 30, 31

get.logPotential, 20, 21
get.potential, 21, 21

infer.chain, 3, 22
infer.conditional, 4, 22
infer.cutset, 4, 23
infer.exact, 3, 24
infer.junction, 4, 25
infer.lbp, 4, 26
infer.sample, 4, 27
infer.trbp, 4, 28
infer.tree, 3, 29

Rain, 35
make.crf, 4, 6, 7, 20, 30, 32, 33, 42–44
make.features, 4, 32, 33
make.par, 4, 32, 33
mrf.nll, 33, 34, 35, 44
mrf.stat, 34, 34, 44
mrf.update, 4, 34, 35, 44

Small, 41
sub.crf, 4, 6, 31, 42

train.crf, 4, 9, 10, 43
train.mrf, 4, 34, 35, 44
Tree, 44