

Package: crfsuite (via r-universe)

September 7, 2024

Type Package

Title Conditional Random Fields for Labelling Sequential Data in Natural Language Processing

Version 0.4.2

Maintainer Jan Wijffels <jwijffels@bnosac.be>

Description Wraps the 'CRFSuite' library <<https://github.com/chokkan/crfsuite>> allowing users to fit a Conditional Random Field model and to apply it on existing data. The focus of the implementation is in the area of Natural Language Processing where this R package allows you to easily build and apply models for named entity recognition, text chunking, part of speech tagging, intent recognition or classification of any category you have in mind. Next to training, a small web application is included in the package to allow you to easily construct training data.

License BSD_3_clause + file LICENSE

URL <https://github.com/bnosac/crfsuite>

Depends R (>= 2.10)

Imports Rcpp, data.table (>= 1.9.6), utils, tools, stats

Suggests udpipe, knitr, rmarkdown

LinkingTo Rcpp

RoxygenNote 7.1.2

VignetteBuilder knitr

Repository <https://bnosac.r-universe.dev>

RemoteUrl <https://github.com/bnosac/crfsuite>

RemoteRef HEAD

RemoteSha 2638e8e42722142c90f406b36f9b81c9ae43865f

Contents

airbnb	2
airbnb_chunks	3
as.crf	3
crf	4
crf_caretmethod	7
crf_cbind_attributes	8
crf_evaluation	9
crf_options	11
merge.chunkrange	12
ner_download_modeldata	13
predict.crf	15
txt_feature	16
txt_sprintf	17
Index	19

airbnb	<i>Dutch reviews of AirBnB customers on Brussels address locations available at www.insideairbnb.com</i>
--------	--

Description

The data contains 500 reviews in Dutch of people who visited an AirBnB apartment in Brussels. The data frame contains the fields

- doc_id: a unique identifier of the review
- listing_id: the airbnb address identifier
- text: text with the feedback of a customer on his visit in the AirBnB apartment

Source

<http://insideairbnb.com/brussels>: information of 2015-10-03

See Also

[airbnb_chunks](#)

Examples

```
data(airbnb)
str(airbnb)
head(airbnb)
```

airbnb_chunks	<i>Dutch reviews of AirBnB customers on Brussels address locations manually tagged with entities</i>
---------------	--

Description

The `airbnb` dataset was manually annotated with the shiny app inside this R package. The annotation shows chunks of data which have been flagged with the following categories: PERSON, LOCATION, DISTANCE. The dataset is an object of class `chunkrange` and of type `data.frame` which contains the following fields:

- `doc_id`: a unique identifier of the review, which is also available in `airbnb`
- `listing_id`: the airbnb address identifier
- `text`: text with the feedback of a customer on his visit in the AirBnB apartment
- `chunk_id`: a chunk identifier
- `chunk_entity`: a chunk entity label
- `chunk`: the text of the chunk which is a substring of `text`
- `start`: the starting position in `text` where the chunk is found
- `end`: the end position in `text` where the chunk is found

See Also

[airbnb_chunks](#)

Examples

```
data(airbnb_chunks)
str(airbnb_chunks)
head(airbnb_chunks)
```

<code>as.crf</code>	<i>Convert a model built with CRFsuite to an object of class <code>crf</code></i>
---------------------	---

Description

If you have a model built with CRFsuite either by this R package or by another software library which wraps CRFsuite (e.g. Python/Java), you can convert it to an object of class `crf` which this package can use to inspect the model and to use it for prediction (if you can mimic the way the attributes are created).

This is for expert use only.

Usage

```
as.crf(file, ...)
```

Arguments

file the path to a file on disk containing the CRFsuite model
 ... other arguments which can be set except the path to the file, namely method, type, options, attribute_names, log (expert use only)

Value

an object of class crf

crf *Linear-chain Conditional Random Field*

Description

Fits a Linear-chain (first-order Markov) CRF on the provided label sequence and saves it on disk in order to do sequence labelling.

Usage

```
crf(
  x,
  y,
  group,
  method = c("lbfgs", "l2sgd", "averaged-perceptron", "passive-aggressive", "arow"),
  options = crf_options(method)$default,
  embeddings,
  file = "annotator.crfsuite",
  trace = FALSE,
  FUN = identity,
  ...
)
```

Arguments

x a character matrix of data containing attributes about the label sequence y or an object which can be coerced to a character matrix. It is important to note that an attribute which has the same value in a different column is considered the same.

y a character vector with the sequence of labels to model

group an integer or character vector of the same length as y indicating the group the sequence y belongs to (e.g. a document or sentence identifier)

method character string with the type of training method. Either one of:

- lbfgs: L-BFGS with L1/L2 regularization
- l2sgd: SGD with L2-regularization
- averaged-perceptron: Averaged Perceptron
- passive-aggressive: Passive Aggressive

	<ul style="list-style-type: none"> • arow: Adaptive Regularization of Weights (AROW)
options	a list of options to provide to the training algorithm. See crf_options for possible options and the example below on how to provide them.
embeddings	a matrix with the same number of rows as x and in the same order with numeric information used in model building (experimental)
file	a character string with the path to the file on disk where the CRF model will be stored.
trace	a logical indicating to show the trace of the training output. Defaults to FALSE.
FUN	a function which can be applied on raw text in order to obtain the attribute matrix used in <code>predict.crf</code> . Currently not used yet.
...	arguments to FUN. Currently not used yet.

Value

an object of class `crf` which is a list with elements

- `method`: The training method
- `type`: The type of graphical model which is always set `crf1d`: Linear-chain (first-order Markov) CRF
- `labels`: The training labels
- `options`: A `data.frame` with the training options provided to the algorithm
- `file_model`: The path where the CRF model is stored
- `attribute_names`: The column names of x
- `log`: The training log of the algorithm
- `FUN`: The argument passed on to FUN
- `ldots`: A list with the arguments passed on to ...

References

More details about this model is available at <http://www.chokkan.org/software/crfsuite/>.

See Also

[predict.crf](#)

Examples

```
## Download modeldata (conll 2002 shared task in Dutch)

x      <- ner_download_modeldata("conll2002-nl")

# for CRAN only - word on a subset of the data
x <- ner_download_modeldata("conll2002-nl", docs = 10)
if(is.data.frame(x)){
  ##
  ## Build Named Entity Recognition model on conll2002-nl
```

```

##
x$pos      <- txt_sprintf("Parts of Speech: %s", x$pos)
x$token    <- txt_sprintf("Token: %s", x$token)
crf_train  <- subset(x, data == "ned.train")
crf_test   <- subset(x, data == "testa")

model <- crf(y = crf_train$label,
             x = crf_train[, c("token", "pos")],
             group = crf_train$doc_id,
             method = "lbfgs",
             options = list(max_iterations = 3, feature.minfreq = 5,
                           c1 = 0, c2 = 1))

model
weights <- coefficients(model)
head(weights$states, n = 20)
head(weights$transitions, n = 20)
stats <- summary(model, "modeldetails.txt")
stats
plot(stats$iterations$loss)

## Use the CRF model to label a sequence
scores <- predict(model,
                  newdata = crf_test[, c("token", "pos")],
                  group = crf_test$doc_id)

head(scores)
crf_test$label <- scores$label

## cleanup for CRAN
if(file.exists(model$file_model)) file.remove(model$file_model)
if(file.exists("modeldetails.txt")) file.remove("modeldetails.txt")
}

##
## More detailed example where text data was annotated with the webapp in the package
## This data is joined with a tokenised dataset to construct the training data which
## is further enriched with attributes of upos/lemma in the neighbourhood
##

library(udpipe)
data(airbnb_chunks, package = "crfsuite")
udmodel <- udpipe_download_model("dutch-lassysmall")
if(!udmodel$download_failed){
  udmodel <- udpipe_load_model(udmodel$file_model)
  airbnb_tokens <- udpipe(x = unique(airbnb_chunks[, c("doc_id", "text")]),
                        object = udmodel)
  x <- merge(airbnb_chunks, airbnb_tokens)
  x <- crf_cbind_attributes(x, terms = c("upos", "lemma"), by = "doc_id")
  model <- crf(y = x$chunk_entity,
              x = x[, grep("upos|lemma", colnames(x), value = TRUE)],
              group = x$doc_id,
              method = "lbfgs", options = list(max_iterations = 5))
  stats <- summary(model)
  stats

```

```
plot(stats$iterations$loss, type = "b", xlab = "Iteration", ylab = "Loss")
scores <- predict(model,
                  newdata = x[, grep("upos|lemma", colnames(x))],
                  group = x$doc_id)
head(scores)
}
```

crf_caretmethod

Functionality allowing to tune a crfsuite model using caret

Description

The object `crf_caretmethod` contains functionality to tune a crf model using `caret`. Each list element of `crf_caretmethod` is a list of functions which can be passed on to the `method` argument of `caret::train` to tune the hyperparameters of the crfsuite model.

Usage

```
crf_caretmethod
```

Format

see details

Details

If you want to tune the hyperparameters of a crfsuite model (see [crf_options](#) and the `options` argument of [crf](#)), you can use the `caret` package.

In order to facilitate this tuning, an object called `crf_caretmethod` has been made available. The object `crf_caretmethod` is a list with 6 elements, where each of these 6 elements can be used in tuning the CRF hyperparameters by passing it on to the `method` argument of the `train` function of the `caret` package.

The list has elements `'default'`, `'lbfgs'`, `'l2sgd'`, `'averaged_perceptron'`, `'passive_aggressive'` and `'arow'`. Each list element corresponds to arguments that you need to tune for each method as used in [crf](#).

For `crf_caretmethod`

1. `lbfgs`: Tuning across all hyperparameters for method `lbfgs`: L-BFGS with L1/L2 regularization
2. `l2sgd`: Tuning across all hyperparameters for method `l2sgd`: SGD with L2-regularization
3. `averaged_perceptron`: Tuning across all hyperparameters for method `averaged-perceptron`: Averaged Perceptron
4. `passive_aggressive`: Tuning across all hyperparameters for method `passive-aggressive`: Passive Aggressive
5. `arow`: Tuning across all hyperparameters for method `arow`: Adaptive Regularization of Weights (AROW)

6. default: Tune over the hyperparameters `feature.minfreq`, `feature.possible_states`, `feature.possible_transitions`, `max_iterations`. While tuning these, it uses the default hyperparameters for each method. This tuning allows you to compare the 5 methods.

For details on the hyperparameter definitions: see [crf_options](#)

`crf_cbind_attributes` *Enrich a data.frame by adding frequently used CRF attributes*

Description

The CRF attributes which are implemented in this function are merely the neighbouring information of a certain field. For example the previous word, the next word, the combination of the previous 2 words. This function `cbind`s these neighbouring attributes as columns to the provided `data.frame`.

By default it adds the following columns to the `data.frame`

- the term itself (`term[t]`)
- the next term (`term[t+1]`)
- the term after that (`term[t+2]`)
- the previous term (`term[t-1]`)
- the term before the previous term (`term[t-2]`)
- as well as all combinations of these terms (bigrams/trigrams/...) where up to `ngram_max` number of terms are combined.

See the examples.

Usage

```
crf_cbind_attributes(
  data,
  terms,
  by,
  from = -2,
  to = 2,
  ngram_max = 3,
  sep = "-"
)
```

Arguments

<code>data</code>	a <code>data.frame</code> which will be coerced to a <code>data.table</code> (cbinding will be done by reference on the existing <code>data.frame</code>)
<code>terms</code>	a character vector of column names which are part of <code>data</code> for which the function will look to the preceding and following rows in order to <code>cbind</code> this information to the data

by	a character vector of column names which are part of data indicating the fields which define the sequence. Preceding/following terms will be looked for within data of by. Typically this will be a document identifier or sentence identifier in an NLP context.
from	integer, by default set to -2, indicating to look up to 2 terms before the current term
to	integer, by default set to 2, indicating to look up to 2 terms after the current term
ngram_max	integer indicating the maximum number of terms to combine (2 means bigrams, 3 trigrams, ...)
sep	character indicating how to combine the previous/next/current terms. Defaults to '-'.

Examples

```
x <- data.frame(doc_id = sort(sample.int(n = 10, size = 1000, replace = TRUE)))
x$pos <- sample(c("Art", "N", "Prep", "V", "Adv", "Adj", "Conj",
                "Punc", "Num", "Pron", "Int", "Misc"),
              size = nrow(x), replace = TRUE)
x <- crf_cbind_attributes(x, terms = "pos", by = "doc_id",
                        from = -1, to = 1, ngram_max = 3)
head(x)
```

```
## Example on some real data
x <- ner_download_modeldata("conll2002-n1")
x <- crf_cbind_attributes(x, terms = c("token", "pos"),
                        by = c("doc_id", "sentence_id"),
                        ngram_max = 3, sep = "|")
```

crf_evaluation

Basic classification evaluation metrics for multi-class labelling

Description

The accuracy, precision, recall, specificity, F1 measure and support metrics are provided for each label in a one-versus the rest setting.

Usage

```
crf_evaluation(
  pred,
  obs,
  labels = na.exclude(unique(c(as.character(pred), as.character(obs)))),
  labels_overall = setdiff(labels, "0")
)
```

Arguments

pred	a factor with predictions
obs	a factor with gold labels
labels	a character vector of possible values that pred and obs can take. Defaults to the values in the data
labels_overall	a character vector of either labels which is either the same as labels or a subset of labels in order to compute a weighted average of the by-label statistics

Value

a list with 2 elements:

- bylabel: data.frame with the accuracy, precision, recall, specificity, F1 score and support (number of occurrences) for each label
- overall: a vector containing
 - the overall accuracy
 - the metrics precision, recall, specificity and F1 score which are weighted averages of these metrics from list element bylabel, where the weight is the support
 - the metrics precision, recall, specificity and F1 score which are averages of these metrics from list element bylabel giving equal weight to each label

Examples

```

pred <- sample(LETTERS, 1000, replace = TRUE)
gold <- sample(LETTERS, 1000, replace = TRUE)
crf_evaluation(pred = pred, obs = gold, labels = LETTERS)

x <- ner_download_modeldata("conll2002-n1")
x <- crf_cbind_attributes(x, terms = c("token", "pos"),
  by = c("doc_id", "sentence_id"))
crf_train <- subset(x, data == "ned.train")
crf_test <- subset(x, data == "testa")
attributes <- grep("token|pos", colnames(x), value=TRUE)
model <- crf(y = crf_train$label,
  x = crf_train[, attributes],
  group = crf_train$doc_id,
  method = "lbfgs")

## Use the model to score on existing tokenised data
scores <- predict(model,
  newdata = crf_test[, attributes],
  group = crf_test$doc_id)
crf_evaluation(pred = scores$label, obs = crf_test$label)
crf_evaluation(pred = scores$label, obs = crf_test$label,
  labels = c("0",
    "B-ORG", "I-ORG", "B-PER", "I-PER",
    "B-LOC", "I-LOC", "B-MISC", "I-MISC"))

```

```

library(udpipe)
pred <- txt_recode(scores$label,
  from = c("B-ORG", "I-ORG", "B-PER", "I-PER",
           "B-LOC", "I-LOC", "B-MISC", "I-MISC"),
  to = c("ORG", "ORG", "PER", "PER",
         "LOC", "LOC", "MISC", "MISC"))
obs <- txt_recode(crf_test$label,
  from = c("B-ORG", "I-ORG", "B-PER", "I-PER",
           "B-LOC", "I-LOC", "B-MISC", "I-MISC"),
  to = c("ORG", "ORG", "PER", "PER",
         "LOC", "LOC", "MISC", "MISC"))
crf_evaluation(pred = pred, obs = obs,
  labels = c("ORG", "LOC", "PER", "MISC", "O"))

```

crf_options

Conditional Random Fields parameters

Description

Conditional Random Fields parameters

Usage

```

crf_options(
  method = c("lbfgs", "l2sgd", "averaged-perceptron", "passive-aggressive", "arow")
)

```

Arguments

method character string with the type of training method. Either one of:

- lbfgs: L-BFGS with L1/L2 regularization
- l2sgd: SGD with L2-regularization
- averaged-perceptron: Averaged Perceptron
- passive-aggressive: Passive Aggressive
- arow: Adaptive Regularization of Weights (AROW)

Value

a list with elements

- method: The training method
- type: The type of graphical model which is always set crf1d: Linear-chain (first-order Markov) CRF
- params: A data.frame with fields arg, arg_default and description indicating the possible hyperparameters of the algorithm, the default values and the description
- default: A list of default values which can be used to pass on to the options argument of `crf`

Examples

```
# L-BFGS with L1/L2 regularization
opts <- crf_options("lbfgs")
str(opts)

# SGD with L2-regularization
crf_options("l2sgd")

# Averaged Perceptron
crf_options("averaged-perceptron")

# Passive Aggressive
crf_options("passive-aggressive")

# Adaptive Regularization of Weights (AROW)
crf_options("arow")
```

merge.chunkrange	<i>CRF Training data construction: add chunk entity category to a tokenised dataset</i>
------------------	---

Description

Chunks annotated with the shiny app in this R package indicate for a chunk of text of a document the entity that it belongs to. As text chunks can contain several words, we need to have a way in order to add this chunk category to each word of a tokenised dataset. That's what this function is doing.

If you have a tokenised data.frame with one row per token/document which indicates the start and end position where the token is found in the text of the document, this function allows to assign the chunk label to each token of the document.

Usage

```
## S3 method for class 'chunkrange'
merge(x, y, by.x = "doc_id", by.y = "doc_id", default_entity = "0", ...)
```

Arguments

x	an object of class chunkrange. A chunkrange is just a data.frame which contains one row per chunk/doc_id. It should have the columns doc_id, text, chunk_id, chunk_entity, start and end. The fields start and end indicate in the original text where the chunks of words start and where it ends. The chunk_entity is a label you have assigned to the chunk (e.g. ORGANISATION / LOCATION / MONEY / LABELXYZ / ...).
y	a tokenised data.frame containing one row per doc_id/token. It should have the columns doc_id, start and end where the fields start and end indicate the positions in the original text of the doc_id where the token starts and where it ends. See the examples.

by.x	a character string of a column of x which is an identifier which defines the sequence. Defaults to 'doc_id'.
by.y	a character string of a column of y which is an identifier which defines the sequence. Defaults to 'doc_id'.
default_entity	character string with the default chunk_entity to be assigned to the token if the token is not part of any chunk range. Defaults to 'O'.
...	not used

Value

the data.frame y where 2 columns are added, namely:

- chunk_entity: The chunk entity of the token if the token is inside the chunk defined in x. If the token is not part of any chunk, the chunk category will be set to the default value.
- chunk_id: The chunk identifier of the chunk for which the token is inside the chunk.

Examples

```
library(udpipe)
udmodel <- udpipe_download_model("dutch-lassysmall")
if(packageVersion("udpipe") >= "0.7"){
  data(airbnb_chunks, package = "crfsuite")
  airbnb_chunks <- head(airbnb_chunks, 20)
  airbnb_tokens <- unique(airbnb_chunks[, c("doc_id", "text")])

  airbnb_tokens <- udpipe(airbnb_tokens, object = udmodel)
  head(airbnb_tokens)
  head(airbnb_chunks)

  ## Add the entity of the chunk to the tokenised dataset
  x <- merge(airbnb_chunks, airbnb_tokens)
  x[, c("doc_id", "token", "chunk_entity")]
  table(x$chunk_entity)
}

## cleanup for CRAN
file.remove(udmodel$file_model)
```

ner_download_modeldata

CRF Training data: download training data for doing Named Entity Recognition (NER)

Description

Download training data for doing Named Entity Recognition (NER)

Usage

```
ner_download_modeldata(
  type = c("conll2002-nl", "conll2002-es", "GermanNER", "wikiner-de-wp2",
    "wikiner-de-wp3", "wikiner-en-wp2", "wikiner-en-wp3", "wikiner-es-wp2",
    "wikiner-es-wp3", "wikiner-fr-wp2", "wikiner-fr-wp3", "wikiner-it-wp2",
    "wikiner-it-wp3", "wikiner-nl-wp2", "wikiner-nl-wp3", "wikiner-pl-wp3",
    "wikiner-pt-wp3", "wikiner-ru-wp2", "wikiner-ru-wp3"),
  docs = -Inf
)
```

Arguments

type	<p>a character string with the type of data to download. See the function usage for all possible values. These data will be downloaded from either:</p> <ul style="list-style-type: none"> • NLTK-data forked repository: https://github.com/bnosac-dev/nltk_data/blob/gh-pages/packages/corpora/conll2002.zip • FOX forked repository of GermanNER: https://github.com/bnosac-dev/FOX/tree/master/input/GermanNER • FOX forked repository of WikiNER: https://github.com/bnosac-dev/FOX/tree/master/input/Wikiner <p>Please visit the information on these repositories first before you use these data in any commercial product.</p>
docs	<p>integer indicating how many documents to sample from the data (only used for data from the NLTK repository). This is only used to reduce CRAN R CMD check training time in the examples of this R package.</p>

Value

a data.frame with training data for a Named Entity Recognition task or an object of try-error in case of failure of downloading the data

Examples

```
## Not run:
x <- ner_download_modeldata("conll2002-nl")
x <- ner_download_modeldata("conll2002-es")
x <- ner_download_modeldata("GermanNER")
x <- ner_download_modeldata("wikiner-en-wp2")
x <- ner_download_modeldata("wikiner-nl-wp3")
x <- ner_download_modeldata("wikiner-fr-wp3")

## End(Not run)
## reduce number of docs
x <- ner_download_modeldata("conll2002-es", docs = 10)
```

predict.crf	<i>Predict the label sequence based on the Conditional Random Field</i>
-------------	---

Description

Predict the label sequence based on the Conditional Random Field

Usage

```
## S3 method for class 'crf'
predict(
  object,
  newdata,
  embeddings,
  group,
  type = c("marginal", "sequence"),
  trace = FALSE,
  ...
)
```

Arguments

object	an object of class <code>crf</code> as returned by crf
newdata	a character matrix of data containing attributes about the label sequence <code>y</code> or an object which can be coerced to a character matrix. This data should be provided in the same format as was used for training the model
embeddings	a matrix with the same number of rows as <code>x</code> and in the same order with numeric information used to predict
group	an integer or character vector of the same length as <code>nrow(newdata)</code> indicating the group the sequence <code>y</code> belongs to (e.g. a document or sentence identifier)
type	either 'marginal' or 'sequence' to get predictions at the level of <code>newdata</code> or at the level of the sequence group. Defaults to 'marginal'
trace	a logical indicating to show the trace of the labelling output. Defaults to FALSE.
...	not used

Value

If `type` is 'marginal': a data.frame with columns `label` and `marginal` containing the viterbi decoded predicted label and marginal probability.

If `type` is 'sequence': a data.frame with columns `group` and `probability` containing for each sequence group the probability of the sequence.

See Also

[crf](#)

Examples

```

library(udpipe)
data(airbnb_chunks, package = "crfsuite")
udmodel <- udpipe_download_model("dutch-lassysmall")
udmodel <- udpipe_load_model(udmodel$file_model)
airbnb_tokens <- unique(airbnb_chunks[, c("doc_id", "text")])
airbnb_tokens <- udpipe_annotate(udmodel,
                                x = airbnb_tokens$text,
                                doc_id = airbnb_tokens$doc_id)
airbnb_tokens <- as.data.frame(airbnb_tokens)
x <- merge(airbnb_chunks, airbnb_tokens)
x <- crf_cbind_attributes(x, terms = c("upos", "lemma"), by = "doc_id")
model <- crf(y = x$chunk_entity,
             x = x[, grep("upos|lemma", colnames(x))],
             group = x$doc_id,
             method = "lbfgs", options = list(max_iterations = 5))
scores <- predict(model,
                  newdata = x[, grep("upos|lemma", colnames(x))],
                  group = x$doc_id, type = "marginal")
head(scores)
scores <- predict(model,
                  newdata = x[, grep("upos|lemma", colnames(x))],
                  group = x$doc_id, type = "sequence")
head(scores)

## cleanup for CRAN
file.remove(model$file_model)
file.remove("modeldetails.txt")
file.remove(udmodel$file)

```

txt_feature

Extract basic text features which are useful for entity recognition

Description

Extract basic text features which are useful for entity recognition

Usage

```

txt_feature(
  x,
  type = c("is_capitalised", "is_url", "is_email", "is_number", "prefix", "suffix",
           "shape"),
  n = 4
)

```


Arguments

x	a character vector
type	a character string, which can be one of 'is_capitalised', 'is_url', 'is_email', 'is_number', 'prefix', 'suffix', 'shape'
n	for type 'prefix' or 'suffix', the number of characters of the prefix/suffix

Value

For type 'is_capitalised', 'is_url', 'is_email', 'is_number': a logical vector of the same length as x, indicating if x is capitalised, a url, an email or a number

For type 'prefix', 'suffix': a character vector of the same length as x, containing the prefix or suffix n number of characters of x

For type 'shape': a character vector of the same length as x, where lowercased elements are replaced with x and uppercased elements with X

Examples

```
txt_feature("Red Devils", type = "is_capitalised")
txt_feature("red devils", type = "is_capitalised")
txt_feature("http://www.bnosac.be", type = "is_url")
txt_feature("info@google.com", type = "is_email")
txt_feature("hi there", type = "is_email")
txt_feature("1230000", type = "is_number")
txt_feature("123.15", type = "is_number")
txt_feature("123,15", type = "is_number")
txt_feature("123abc", type = "is_number")
txt_feature("abcdefghijklmnopqrstuvwxyz", type = "prefix", n = 3)
txt_feature("abcdefghijklmnopqrstuvwxyz", type = "suffix", n = 3)
txt_feature("Red Devils", type = "shape")
txt_feature("red devils", type = "shape")
```

txt_sprintf

NA friendly version of sprintf

Description

Does the same as the function [sprintf](#) except that if in ... NA values are passed, also NA values are returned instead of being replaced by the character string 'NA'.

Usage

```
txt_sprintf(fmt, ...)
```

Arguments

fmt	a character vector of format strings, which will be fed on to sprintf
...	values to be passed into fmt, the ... will be passed on to sprintf

Value

The same as what `sprintf` returns: a character vector of length that of the longest input in
Except, in case any of the values passed on to . . . are NA, the corresponding returned value will be set to NA for that element of the vector.
See the examples to see the difference with `sprintf`

See Also

`sprintf`

Examples

```
sprintf("(w-1):%s", c("xyz", NA, "abc"))  
txt_sprintf("(w-1):%s", c("xyz", NA, "abc"))  
sprintf("(w-1):%s_%s", c("xyz", NA, "abc"), c(NA, "xyz", "abc"))  
txt_sprintf("(w-1):%s_%s", c("xyz", NA, "abc"), c(NA, "xyz", "abc"))
```

Index

* datasets

crf_caretmethod, [7](#)

airbnb, [2](#), [3](#)

airbnb_chunks, [2](#), [3](#), [3](#)

as.crf, [3](#)

crf, [4](#), [7](#), [11](#), [15](#)

crf_caretmethod, [7](#)

crf_cbind_attributes, [8](#)

crf_evaluation, [9](#)

crf_options, [5](#), [7](#), [8](#), [11](#)

merge.chunkrange, [12](#)

ner_download_modeldata, [13](#)

predict.crf, [5](#), [15](#)

sprintf, [17](#), [18](#)

txt_feature, [16](#)

txt_sprintf, [17](#)