Package ‘lime’

August 19, 2022

Type Package

Title Local Interpretable Model-Agnostic Explanations

Version 0.5.3

Maintainer Emil Hvitfeldt <emilhhvitfeldt@gmail.com>

Description When building complex models, it is often difficult to explain why
the model should be trusted. While global measures such as accuracy are
useful, they cannot be used for explaining why a model made a specific
prediction. ‘lime’ (a port of the ‘lime’ ‘Python’ package) is a method for
explaining the outcome of black box models by fitting a local model around
the point in question an perturbations of this point. The approach is
described in more detail in the article by Ribeiro et al. (2016)

License MIT + file LICENSE


BugReports https://github.com/thomasp85/lime/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

VignetteBuilder knitr

Imports glmnet, stats, ggplot2, tools, stringi, Matrix, Rcpp,
assertthat, methods, grDevices, gower

Suggests xgboost, testthat, mlr, h2o, text2vec, MASS, covr, knitr,
rmardown, sessioninfo, magick, keras, htmlwidgets, shiny,
shinythemes, ranger

LinkingTo Rcpp, RcppEigen

NeedsCompilation yes

Author Emil Hvitfeldt [aut, cre] (<https://orcid.org/0000-0002-0679-1945>),
Thomas Lin Pedersen [aut] (<https://orcid.org/0000-0002-5147-4711>),
Michaël Benesty [aut]

Repository CRAN

Date/Publication 2022-08-19 08:50:06 UTC
Description

When building complex models, it is often difficult to explain why the model should be trusted. While global measures such as accuracy are useful, they cannot be used for explaining why a model made a specific prediction. 'lime' (a port of the 'lime' 'Python' package) is a method for explaining the outcome of black box models by fitting a local model around the point in question an perturbations of this point. The approach is described in more detail in the article by Ribeiro et al. (2016) arXiv:1602.04938.

Details

This package is a port of the original Python lime package implementing the prediction explanation framework laid out Ribeiro et al. (2016). The package supports models from caret and mlr natively, but see the docs for how to make it work for any model.

Main functions:

Use of lime is mainly through two functions. First you create an explainer object using the lime() function based on the training data and the model, and then you can use the explain() function along with new data and the explainer to create explanations for the model output.

Along with these two functions, lime also provides the plot_features() and plot_text_explanations() function to visualise the explanations directly.
as_classifier

**Author(s)**

**Maintainer:** Emil Hvitfeldt &lt;emilhhvitfeldt@gmail.com&gt; (ORCID)

**Authors:**

- Thomas Lin Pedersen &lt;thomasp85@gmail.com&gt; (ORCID)
- Michaël Benesty &lt;michael@benesty.fr&gt;

**References**


**See Also**

Useful links:

- https://lime.data-imaginist.com
- https://github.com/thomasp85/lime
- Report bugs at https://github.com/thomasp85/lime/issues

---

**Description**

`lime` requires knowledge about the type of model it is dealing with, more specifically whether the model is a regressor or a classifier. If the model class has a `model_type()` method defined `lime` can figure it out on its own but if not, you can wrap your model in either of these functions to indicate what type of model `lime` is dealing with. This can also be used to overwrite the output from `model_type()` if the implementation uses some heuristic that doesn’t work for your particular model (e.g. keras models types are found by checking if the activation in the last layer is linear or not - this is rather crude). In addition `as_classifier` can be used to overwrite the returned class labels - this is handy if the model does not store the labels (again, keras springs to mind).

**Usage**

```r
as_classifier(x, labels = NULL)
```

```r
as_regressor(x)
```

**Arguments**

- `x` The model object
- `labels` An optional character vector giving labels for each class

**Value**

A model augmented with information about the model type and (potentially) the class labels.
### default_tokenize

*Default function to tokenize*

**Description**

This tokenizer uses `stringi::stri_split_boundaries()` to tokenize a character vector. To be used with `[explain.character()]`.

**Usage**

```r
default_tokenize(text)
```

**Arguments**

- **text**: text to tokenize as a character vector

**Value**

a character vector.

**Examples**

```r
data('train_sentences')
default_tokenize(train_sentences$text[1])
```

### explain

*Explain model predictions*

**Description**

Once an explainer has been created using the `lime()` function it can be used to explain the result of the model on new observations. The `explain()` function takes new observation along with the explainer and returns a data.frame with prediction explanations, one observation per row. The returned explanations can then be visualised in a number of ways, e.g. with `plot_features()`.

**Usage**

```r
# S3 method for class 'data.frame'
explain(
  x,
  explainer,
  labels = NULL,
  n_labels = NULL,
  n_features,
  n_permutations = 5000,
)```
explain

feature_select = "auto",
dist_fun = "gower",
kernel_width = NULL,
gower_pow = 1,
...
}

## S3 method for class 'character'
explain(
x, 
explainer, 
labels = NULL, 
n_labels = NULL, 
n_features, 
n_permutations = 5000, 
feature_select = "auto", 
single_explanation = FALSE, 
...
)

explain(
x, 
explainer, 
labels, 
n_labels = NULL, 
n_features, 
n_permutations = 5000, 
feature_select = "auto", 
...
)

## S3 method for class 'imagefile'
explain(
x, 
explainer, 
labels = NULL, 
n_labels = NULL, 
n_features, 
n_permutations = 1000, 
feature_select = "auto", 
n_superpixels = 50, 
weight = 20, 
n_iter = 10, 
p_remove = 0.5, 
batch_size = 10, 
background = "grey", 
...
)
Arguments

- **x**: New observations to explain, of the same format as used when creating the explainer.
- **explainer**: An explainer object to use for explaining the observations.
- **labels**: The specific labels (classes) to explain in case the model is a classifier. For classifiers either this or **n_labels** must be given.
- **n_labels**: The number of labels to explain. If this is given for classifiers the top n_label classes will be explained.
- **n_features**: The number of features to use for each explanation.
- **n_permutations**: The number of permutations to use for each explanation.
- **feature_select**: The algorithm to use for selecting features. One of:
  - "auto": If n_features < 6 use "forward_selection" else use "highest_weights".
  - "none": Ignore n_features and use all features.
  - "forward_selection": Add one feature at a time until n_features is reached, based on quality of a ridge regression model.
  - "highest_weights": Fit a ridge regression and select the n_features with the highest absolute weight.
  - "lasso_path": Fit a lasso model and choose the n_features whose lars path converge to zero the latest.
  - "tree": Fit a tree to select n_features (which needs to be a power of 2). It requires last version of XGBoost.
- **dist_fun**: The distance function to use for calculating the distance from the observation to the permutations. If dist_fun = 'gower' (default) it will use gower::gower_dist(). Otherwise it will be forwarded to stats::dist().
- **kernel_width**: The width of the exponential kernel that will be used to convert the distance to a similarity in case dist_fun != 'gower'.
- **gower_pow**: A modifier for gower distance. The calculated distance will be raised to the power of this value.
- **single_explanation**: A boolean indicating whether to pool all text in x into a single explanation.
- **n_superpixels**: The number of segments an image should be split into.
- **weight**: How high should locality be weighted compared to colour. High values leads to more compact superpixels, while low values follow the image structure more.
- **n_iter**: How many iterations should the segmentation run for.
- **p_remove**: The probability that a superpixel will be removed in each permutation.
- **batch_size**: The number of explanations to handle at a time.
- **background**: The colour to use for blocked out superpixels.
**Value**

A data.frame encoding the explanations one row per explained observation. The columns are:

- **model_type**: The type of the model used for prediction.
- **case**: The case being explained (the rowname in **cases**).
- **model_r2**: The quality of the model used for the explanation
- **model_intercept**: The intercept of the model used for the explanation
- **model_prediction**: The prediction of the observation based on the model used for the explanation.
- **feature**: The feature used for the explanation
- **feature_value**: The value of the feature used
- **feature_weight**: The weight of the feature in the explanation
- **feature_desc**: A human readable description of the feature importance.
- **data**: Original data being explained
- **prediction**: The original prediction from the model

Furthermore classification explanations will also contain:

- **label**: The label being explained
- **label_prob**: The probability of label as predicted by model

**Examples**

```r
# Explaining a model and an explainer for it
library(MASS)
iris_test <- iris[1, 1:4]
iris_train <- iris[-1, 1:4]
iris_lab <- iris[[5]][-1]
model <- lda(iris_train, iris_lab)
explanation <- lime(iris_train, model)

# This can now be used together with the explain method
explain(iris_test, explanation, n_labels = 1, n_features = 2)
```

**Description**

Display text explanation in an interactive way. You can:

- Create an output to insert text explanation plot in Shiny application.
- Render the text explanations in Shiny application.
Usage

```r
interactive_text_explanations(
  explainer,
  window_title = "Text model explainer",
  title = "Local Interpretable Model-agnostic Explanations",
  place_holder = "Put here the text to explain",
  minimum_length = 3,
  minimum_length_error = "Text provided is too short to be explained (>= 3).",
  max_feature_to_select = 20
)
```

text_explanations_output(outputId, width = "100\%", height = "400px")

`render_text_explanations(expr, env = parent.frame(), quoted = FALSE)`

Arguments

- `explainer`: parameters
- `window_title`, `title`, `place_holder`, `minimum_length_error`: text to be displayed on the page
- `minimum_length`: don't update display if text is shorter than this parameter
- `max_feature_to_select`: up limit to the number of words that can be selected
- `outputId`: output variable to read from
- `width`, `height`: Must be a valid CSS unit or a number, which will be coerced to a string and have "px" appended.
- `expr`: An expression that generates an HTML widget
- `env`: The environment in which to evaluate `expr`.
- `quoted`: Is `expr` a quoted expression (with `quote()`)? This is useful if you want to save an expression in a variable.

Details

- send a new sentence
- update the parameters of the explainer

Value

An output function that enables the use of the widget within Shiny applications.

A render function that enables the use of the widget within Shiny applications.

Examples

```r
## Not run:
library(text2vec)
```
library(xgboost)

data(train_sentences)
data(test_sentences)

get_matrix <- function(text) {
  it <- itoken(text, progressbar = FALSE)
  create_dtm(it, vectorizer = hash_vectorizer())
}

dtm_train = get_matrix(train_sentences$text)

xgb_model <- xgb.train(list(max_depth = 7, eta = 0.1, objective = "binary:logistic",
                           eval_metric = "error", nthread = 1),
                       xgb.DMatrix(dtm_train, label = train_sentences$class.text == "OWNX"),
                       nrounds = 50)

sentences <- head(test_sentences[test_sentences$class.text == "OWNX", "text"], 1)
explainer <- lime(train_sentences$text, xgb_model, get_matrix)

# The explainer can now be queried interactively:
interactive_text_explanations(explainer)

## End(Not run)

---

**lime**

Create a model explanation function based on training data

**Description**

This is the main function of the `lime` package. It is a factory function that returns a new function that can be used to explain the predictions made by black box models. This is a generic with methods for the different data types supported by `lime`.

**Usage**

```r
## S3 method for class 'data.frame'
lime(
  x,
  model,
  preprocess = NULL,
  bin_continuous = TRUE,
  n_bins = 4,
  quantile_bins = TRUE,
  use_density = TRUE,
  ...
)

## S3 method for class 'character'
```
lime(x, model, preprocess = NULL, tokenization = default_tokenize, keep_word_position = FALSE, ...)

## S3 method for class 'imagefile'
lime(x, model, preprocess = NULL, ...)
lime(x, model, ...)

Arguments

- **x**: The training data used for training the model that should be explained.
- **model**: The model whose output should be explained.
- **preprocess**: Function to transform a character vector to the format expected from the model.
- **bin_continuous**: Should continuous variables be binned when making the explanation?
- **n_bins**: The number of bins for continuous variables if bin_continuous = TRUE.
- **quantile_bins**: Should the bins be based on n_bins quantiles or spread evenly over the range of the training data.
- **use_density**: If bin_continuous = FALSE should continuous data be sampled using a kernel density estimation. If not, continuous features are expected to follow a normal distribution.
- **...**: Arguments passed on to methods
- **tokenization**: Function used to tokenize text for the permutations.
- **keep_word_position**: Set to TRUE if to keep order of words. Warning: each word will be replaced by word_position.

Value

Return an explainer which can be used together with `explain()` to explain model predictions.

Examples

```r
# Explaining a model based on tabular data
library(MASS)
iris_test <- iris[1, 1:4]
iris_train <- iris[-1, 1:4]
iris_lab <- iris[[5]][-1]
# Create linear discriminant model on iris data
model <- lda(iris_train, iris_lab)
# Create explanation object
```
# This can now be used together with the explain method
define(iris_test, explanation, n_labels = 1, n_features = 2)

## Not run:
# Explaining a model based on text data

# Purpose is to classify sentences from scientific publications
# and find those where the team writes about their own work
# (category OWNX in the provided dataset).

define(text2vec)
define(xgboost)

data(train_sentences)
data(test_sentences)

define(get_matrix)
    it <- itoken(text, progressbar = F)
define(create_dtm(it, vectorizer = hash_vectorizer()))

dtm_train = get_matrix(train_sentences$text)

define(xgb_model)
    xgb.train(list(max_depth = 7, eta = 0.1, objective = "binary:logistic",
                  eval_metric = "error", nthread = 1),
              xgb.DMatrix(dtm_train, label = train_sentences$class.text == "OWNX"),
              nrounds = 50)

sentences <- head(test_sentences[test_sentences$class.text == "OWNX", "text"], 1)
define(explainer)
    lime(train_sentences$text, xgb_model, get_matrix)
define(explanations)
    explain(sentences, explainer, n_labels = 1, n_features = 2)

    # We can see that many explanations are based
    # on the presence of the word 'we' in the sentences
    # which makes sense regarding the task.
define(print(explanations))

## Not run:
# library(keras)
# library(abind)

# get some image
img_path <- system.file('extdata', 'produce.png', package = 'lime')

# load a predefined image classifier
model <- application_vgg16(
    weights = "imagenet",
    include_top = TRUE
)

# create a function that prepare images for the model
img_preprocess <- function(x) {
    
}
arrays <- lapply(x, function(path) {
  img <- image_load(path, target_size = c(224,224))
  x <- image_to_array(img)
  x <- array_reshape(x, c(1, dim(x)))
  x <- imagenet_preprocess_input(x)
})
do.call(abind, c(arrays, list(along = 1)))

# Create an explainer (lime recognise the path as an image)
explainer <- lime(img_path, as_classifier(model, unlist(labels)), img_preprocess)

# Explain the model (can take a long time depending on your system)
explanation <- explain(img_path, explainer, n_labels = 2, n_features = 10, n_superpixels = 70)

## End(Not run)

---

### model_support

#### Methods for extending limes model support

**Description**

In order to have lime support for your model of choice lime needs to be able to get predictions from the model in a standardised way, and it needs to be able to know whether it is a classification or regression model. For the former it calls the predict_model() generic which the user is free to supply methods for without overriding the standard predict() method. For the latter the model must respond to the model_type() generic.

**Usage**

predict_model(x, newdata, type, ...)

model_type(x, ...)

**Arguments**

- **x** A model object
- **newdata** The new observations to predict
- **type** Either 'raw' to indicate predicted values, or 'prob' to indicate class probabilities
- **...** passed on to predict method

**Value**

A data.frame in the case of predict_model(). If type = 'raw' it will contain one column named 'Response' holding the predicted values. If type = 'prob' it will contain a column for each of the possible classes named after the class, each column holding the probability score for class membership. For model_type() a character string. Either 'regression' or 'classification' is currently supported.
Supported Models

Out of the box, lime supports the following model objects:

- train from caret
- WrappedModel from mlr
- xgb.Booster from xgboost
- H2OModel from h2o
- keras.engine.training.Model from keras
- lda from MASS (used for low-dependency examples)

If your model is not one of the above you’ll need to implement support yourself. If the model has a predict interface mimicking that of predict.train() from caret, it will be enough to wrap your model in as_classifier()/as_regressor() to gain support. Otherwise you’ll need need to implement a predict_model() method and potentially a model_type() method (if the latter is omitted the model should be wrapped in as_classifier()/as_regressor(), everytime it is used in lime()).

Examples

# Example of adding support for lda models (already available in lime)
predict_model.lda <- function(x, newdata, type, ...) {
  res <- predict(x, newdata = newdata, ...)
  switch(
    type,
    raw = data.frame(Response = res$class, stringsAsFactors = FALSE),
    prob = as.data.frame(res$posterior, check.names = FALSE)
  )
}
model_type.lda <- function(x, ...) 'classification'

plot_explanations

Plot a condensed overview of all explanations

Description

This function produces a facetted heatmap visualisation of all case/label/feature combinations. Compared to plot_features() it is much more condensed, thus allowing for an overview of many explanations in one plot. On the other hand it is less useful for getting exact numerical statistics of the explanation.

Usage

plot_explanations(explanation, ...)

plot_explanations

Plot a condensed overview of all explanations
Arguments

explanation A data.frame as returned by `explain()`.

... Parameters passed on to `ggplot2::facet_wrap()`

Value

A ggplot object

See Also

Other explanation plots: `plot_features()`, `plot_text_explanations()`

Examples

```r
# Create some explanations
library(MASS)
iris_test <- iris[c(1, 51, 101), 1:4]
iris_train <- iris[-c(1, 51, 101), 1:4]
iris_lab <- iris[[5]][-c(1, 51, 101)]
model <- lda(iris_train, iris_lab)
explanation <- lime(iris_train, model)
explanations <- explain(iris_test, explanation, n_labels = 1, n_features = 2)

# Get an overview with the standard plot
plot_explanations(explanations)
```

Description

This function creates a compact visual representation of the explanations for each case and label combination in an explanation. Each extracted feature is shown with its weight, thus giving the importance of the feature in the label prediction.

Usage

`plot_features(explanation, ncol = 2, cases = NULL)`

Arguments

- explanation A data.frame as returned by `explain()`.
- ncol The number of columns in the faceted plot
- cases An optional vector with case names to plot. `explanation` will be filtered to only include these cases prior to plotting
plot_image_explanation

Value
A ggplot object

See Also
Other explanation plots: plot_explanations(), plot_text_explanations()

Examples

# Create some explanations
library(MASS)
iris_test <- iris[1, 1:4]
iris_train <- iris[-1, 1:4]
iris_lab <- iris[[5]][-1]
model <- lda(iris_train, iris_lab)
explanation <- lime(iris_train, model)
explanations <- explain(iris_test, explanation, n_labels = 1, n_features = 2)

# Get an overview with the standard plot
plot_features(explanations)

plot_image_explanation

Display image explanations as superpixel areas

Description
When classifying images one is often interested in seeing the areas that supports and/or contradicts a classification. plot_image_explanation() will take the result of an image explanation and highlight the areas found relevant to each label in the explanation. The highlighting can either be done by blocking the parts of the image not related to the classification, or by encircling and colouring the areas that influence the explanation.

Usage

plot_image_explanation(
  explanation,
  which = 1,
  threshold = 0.02,
  show_negative = FALSE,
  display = "outline",
  fill_alpha = 0.3,
  outline_col = c("blue", "red"),
  block_col = "grey"
)
plot_superpixels

Arguments

explanation The explanation created with an image_explainer
which The case in explanation to illustrate. plot_image_explanation only supports showing one case at a time.
threshold The lowest absolute weighted superpixels to include
show_negative Should areas that contradict the prediction also be shown
display How should the areas be shown? Either outline or block
fill_alpha In case of display = 'outline' how opaque should the area colour be?
outline_col A vector of length 2 giving the colour for supporting and contradicting areas respectively if display = 'outline'
block_col The colour to use for the unimportant areas if display = 'block'

Value

A ggplot object

Examples

```r
## Not run:
# load precalculated explanation as it takes a long time to create
explanation <- .load_image_example()

# Default
plot_image_explanation(explanation)

# Block out background instead
plot_image_explanation(explanation, display = 'block')

# Show negatively correlated areas as well
plot_image_explanation(explanation, show_negative = TRUE)
```

plot_superpixels Test super pixel segmentation

Description

The segmentation of an image into superpixels are an important step in generating explanations for image models. It is both important that the segmentation is correct and follows meaningful patterns in the picture, but also that the size/number of superpixels are appropriate. If the important features in the image are chopped into too many segments the permutations will probably damage the picture beyond recognition in almost all cases leading to a poor or failing explanation model. As the size of the object of interest is varying it is impossible to set up hard rules for the number of superpixels to
segment into - the larger the object is relative to the size of the image, the fewer superpixels should be generated. Using `plot_superpixels` it is possible to evaluate the superpixel parameters before starting the time consuming explanation function.

Usage

```r
plot_superpixels(
  path,
  n_superpixels = 50,
  weight = 20,
  n_iter = 10,
  colour = "black"
)
```

Arguments

- `path` The path to the image. Must be readable by `magick::image_read()`
- `n_superpixels` The number of superpixels to segment into
- `weight` How high should locality be weighted compared to colour. High values leads to more compact superpixels, while low values follow the image structure more
- `n_iter` How many iterations should the segmentation run for
- `colour` What line colour should be used to show the segment boundaries

Value

A ggplot object

Examples

```r
image <- system.file("extdata", "produce.png", package = "lime")

# plot with default settings
plot_superpixels(image)

# Test different settings
plot_superpixels(image, n_superpixels = 100, colour = "white")
```

Description

Highlight words which explains a prediction.
Usage

plot_text_explanations(explanations, ...)

Arguments

explanations object returned by the lime.character function.
... parameters passed to htmlwidgets::sizingPolicy()

See Also

Other explanation plots: plot_explanations(), plot_features()

Examples

# We load a precalculated explanation set based on the procedure in the ?lime
# examples
explanations <- .load_text_example()

# We see that the explanations are in the expected format
print(explanations)

# We can now get the explanations in the context of the input text
plot_text_explanations(explanations)

---

stop_words_sentences  Stop words list

Description

List of words that can be safely removed from sentences.

Usage

stop_words_sentences

Format

Character vector of stop words

Source

https://archive.ics.uci.edu/ml/datasets/
**Description**

This corpus contains sentences from the abstract and introduction of 30 scientific articles that have been annotated (i.e. labeled or tagged) according to a modified version of the Argumentative Zones annotation scheme.

**Usage**

**test_sentences**

**Format**

2 data frame with 3117 rows and 2 variables:

- **text** the sentences as a character vector
- **class.text** the category of the sentence

**Details**

These 30 scientific articles come from three different domains:

1. PLoS Computational Biology (PLOS)
2. The machine learning repository on arXiv (ARXIV)
3. The psychology journal Judgment and Decision Making (JDM)

There are 10 articles from each domain. In addition to the labeled data, this corpus also contains a corresponding set of unlabeled articles. These unlabeled articles also come from PLOS, ARXIV, and JDM. There are 300 unlabeled articles from each domain (again, only the sentences from the abstract and introduction). These unlabeled articles can be used for unsupervised or semi-supervised approaches to sentence classification which rely on a small set of labeled data and a larger set of unlabeled data.

===== References =====


**Source**

https://archive.ics.uci.edu/ml/datasets/Sentence+Classification
Description

This corpus contains sentences from the abstract and introduction of 30 scientific articles that have been annotated (i.e. labeled or tagged) according to a modified version of the Argumentative Zones annotation scheme.

Usage

train_sentences

Format

2 data frame with 3117 rows and 2 variables:

text the sentences as a character vector

class.text the category of the sentence

Details

These 30 scientific articles come from three different domains:

1. PLoS Computational Biology (PLOS)
2. The machine learning repository on arXiv (ARXIV)
3. The psychology journal Judgment and Decision Making (JDM)

There are 10 articles from each domain. In addition to the labeled data, this corpus also contains a corresponding set of unlabeled articles. These unlabeled articles also come from PLOS, ARXIV, and JDM. There are 300 unlabeled articles from each domain (again, only the sentences from the abstract and introduction). These unlabeled articles can be used for unsupervised or semi-supervised approaches to sentence classification which rely on a small set of labeled data and a larger set of unlabeled data.

===== References =====


Source

https://archive.ics.uci.edu/ml/datasets/Sentence+Classification
Index

* datasets
  stop_words_sentences, 18
  test_sentences, 19
  train_sentences, 20
* explanation plots
  plot_explanations, 13
  plot_features, 14
  plot_text_explanations, 17
_PACKAGE (lime-package), 2
as_classifier, 3
as_classifier(), 13
as_regressor (as_classifier), 3
as_regressor (), 13
default_tokenize, 4
explain, 4
explain(), 2, 10, 14
ggplot2::facet_wrap(), 14
gower::gower_dist(), 6
interactive_text_explanations, 7
lime, 9
lime(), 2, 4, 13
lime-package, 2
lime.character, 18
magick::image_read(), 17
model_support, 12
model_type (model_support), 12
model_type(), 3
plot_explanations, 13, 15, 18
plot_features, 14, 14, 18
plot_features(), 2, 4, 13
plot_image_explanation, 15
plot_superpixels, 16
plot_text_explanations, 14, 15, 17
plot_text_explanations(), 2
predict_model (model_support), 12
quote(), 8
render_text_explanations
  (interactive_text_explanations), 7
stats::dist(), 6
stop_words_sentences, 18
stringi::stri_split_boundaries(), 4
test_sentences, 19
text_explanations_output
  (interactive_text_explanations), 7
the docs, 2
train_sentences, 20