pressagio Documentation

Release 0.1.3

Peter Bouda

Nov 18, 2019

Contents

1 Example Usage **3**

2 Running the tests **5**

3 API documentation **7**
3.1 pressagio Package 7

4 Indices and tables **21**

Python Module Index **23**

Index **25**

Pressagio is a library that predicts text based on n-gram models. For example, you can send a string and the library will return the most likely word completions for the last token in the string.

CHAPTER 1

Example Usage

The repository contains two example scripts in the folder `example` to demonstrate how to build a language model and use the model for prediction. You can check the code of those two scripts how to use pressagio in your own projects. Here is how to use the two scripts to predict the next word in a phrase.

First, you have to build a languange model. We will use the script example/text2ngram.py to add 1-, 2- and 3-grams of a given text to a sqlite database. For demonstration purposes we will use a simple text file that comes with pressagio's tests. You have to run the script three times to create a table for each of the n-grams:

```
$ python example/text2ngram.py -n 1 -o test.sqlite tests/test_data/der_linksdenker.txt
$ python example/text2ngram.py -n 2 -o test.sqlite tests/test_data/der_linksdenker.txt
$ python example/text2ngram.py -n 3 -o test.sqlite tests/test_data/der_linksdenker.txt
```

This will create a file `test.sqlite` in the current directory. We can now use this database to get a prediction for a phrase. We will use the script example/predict.py which uses the configuration file example/example_profile.ini. Note that you will always need a configuration file if you want to use the built-in predictor. To get a prediction call:

```
$ python example/predict.py
['warm', 'der', 'und', 'die', 'nicht']
```

The script will just output a list of predictions.

CHAPTER 2

Running the tests

```
$ python -m unittest discover
```

CHAPTER 3

API documentation

3.1 pressagio Package

3.1.1 pressagio.callback

Base class for callbacks.

class pressagio.callback.**Callback**
Base class for callbacks.

Methods

future_stream	
past_stream	
update	

__init__(*self*)
Initialize self. See help(type(self)) for accurate signature.

3.1.2 pressagio.character

3.1.3 pressagio.combiner

Combiner classes to merge results from several predictors.

class pressagio.combiner.**Combiner**
Base class for all combiners

Methods

combine	
filter	

`__init__`(*self*)
Initialize self. See help(type(self)) for accurate signature.

`class` `pressagio.combiner.`**`MeritocracyCombiner`**

Methods

combine	
filter	

`__init__`(*self*)
Initialize self. See help(type(self)) for accurate signature.

3.1.4 pressagio.context_tracker

Class for context tracker.

`class` `pressagio.context_tracker.`**`ContextTracker`**(*config*, *predictor_registry*, *callback*)
Tracks the current context.

Methods

context_change	
extra_token_to_learn	
future_stream	
is_completion_valid	
past_stream	
prefix	
token	
update_context	

`__init__`(*self*, *config*, *predictor_registry*, *callback*)
Initialize self. See help(type(self)) for accurate signature.

`exception` `pressagio.context_tracker.`**`InvalidCallbackException`**

3.1.5 pressagio.dbconnector

Classes to connect to databases.

`class` `pressagio.dbconnector.`**`DatabaseConnector`**(*dbname*, *cardinality=1*)
Base class for all database connectors.

Methods

create_bigram_table(self)	Creates a table for n-grams of cardinality 2.
create_index(self, cardinality)	Create an index for the table with the given cardinality.
create_ngram_table(self, cardinality)	Creates a table for n-gram of a give cardinality.
create_trigram_table(self)	Creates a table for n-grams of cardinality 3.
create_unigram_table(self)	Creates a table for n-grams of cardinality 1.
delete_index(self, cardinality)	Delete index for the table with the given cardinality.
delete_ngram_table(self, cardinality)	Deletes the table for n-gram of a give cardinality.
insert_ngram(self, ngram, count)	Inserts a given n-gram with count into the database.
ngram_count(self, ngram)	Gets the count for a given ngram from the database.
ngrams(self[, with_counts])	Returns all ngrams that are in the table.
remove_ngram(self, ngram)	Removes a given ngram from the databae.
update_ngram(self, ngram, count)	Updates a given ngram in the database.

close_database	
execute_sql	
increment_ngram_count	
ngram_like_table	
ngram_like_table_filtered	
open_database	
unigram_counts_sum	

`__init__`(*self*, *dbname*, *cardinality=1*)
Constructor of the base class DababaseConnector.

Parameters

dbname [str] path to the database file or database name

cardinality [int] default cardinality for n-grams

`create_bigram_table`(*self*)
Creates a table for n-grams of cardinality 2.

`create_index`(*self*, *cardinality*)
Create an index for the table with the given cardinality.

Parameters

cardinality [int] The cardinality to create a index for.

`create_ngram_table`(*self*, *cardinality*)
Creates a table for n-gram of a give cardinality. The table name is constructed from this parameter, for example for cardinality *2* there will be a table *_2_gram* created.

Parameters

cardinality [int] The cardinality to create a table for.

`create_trigram_table`(*self*)
Creates a table for n-grams of cardinality 3.

`create_unigram_table`(*self*)
Creates a table for n-grams of cardinality 1.

`delete_index`(*self*, *cardinality*)
Delete index for the table with the given cardinality.

Parameters

cardinality [int] The cardinality of the index to delete.

`delete_ngram_table`(*self*, *cardinality*)
Deletes the table for n-gram of a give cardinality. The table name is constructed from this parameter, for example for cardinality *2* there will be a table *_2_gram* deleted.

Parameters

cardinality [int] The cardinality of the table to delete.

`insert_ngram`(*self*, *ngram*, *count*)
Inserts a given n-gram with count into the database.

Parameters

ngram [iterable of str] A list, set or tuple of strings.

count [int] The count for the given n-gram.

`ngram_count`(*self*, *ngram*)
Gets the count for a given ngram from the database.

Parameters

ngram [iterable of str] A list, set or tuple of strings.

Returns

count [int] The count of the ngram.

`ngrams`(*self*, *with_counts=False*)
Returns all ngrams that are in the table.

Parameters

None

Returns

ngrams [generator] A generator for ngram tuples.

`remove_ngram`(*self*, *ngram*)
Removes a given ngram from the databae. The ngram has to be in the database, otherwise this method will stop with an error.

Parameters

ngram [iterable of str] A list, set or tuple of strings.

`update_ngram`(*self*, *ngram*, *count*)
Updates a given ngram in the database. The ngram has to be in the database, otherwise this method will stop with an error.

Parameters

ngram [iterable of str] A list, set or tuple of strings.

count [int] The count for the given n-gram.

class pressagio.dbconnector.**PostgresDatabaseConnector**(*dbname*, *cardinality=1*, *host='localhost'*, *port=5432*, *user='postgres'*, *password=None*, *connection=None*)

Database connector for postgres databases.

Methods

close_database(self)	Closes the sqlite database.
commit(self)	Sends a commit to the database.
create_bigram_table(self)	Creates a table for n-grams of cardinality 2.
create_database(self)	Creates an empty database if not exists.
create_index(self, cardinality)	Create an index for the table with the given cardinality.
create_ngram_table(self, cardinality)	Creates a table for n-gram of a give cardinality.
create_trigram_table(self)	Creates a table for n-grams of cardinality 3.
create_unigram_table(self)	Creates a table for n-grams of cardinality 1.
delete_index(self, cardinality)	Delete index for the table with the given cardinality.
delete_ngram_table(self, cardinality)	Deletes the table for n-gram of a give cardinality.
execute_sql(self, query)	Executes a given query string on an open postgres database.
insert_ngram(self, ngram, count)	Inserts a given n-gram with count into the database.
ngram_count(self, ngram)	Gets the count for a given ngram from the database.
ngrams(self[, with_counts])	Returns all ngrams that are in the table.
open_database(self)	Opens the sqlite database.
remove_ngram(self, ngram)	Removes a given ngram from the databae.
reset_database(self)	Re-create an empty database.
update_ngram(self, ngram, count)	Updates a given ngram in the database.

increment_ngram_count	
ngram_like_table	
ngram_like_table_filtered	
unigram_counts_sum	

__init__(*self*, *dbname*, *cardinality=1*, *host='localhost'*, *port=5432*, *user='postgres'*, *password=None*, *connection=None*)

Constructor for the postgres database connector.

Parameters

dbname [str] the database name

cardinality [int] default cardinality for n-grams

host [str] hostname of the postgres database

port [int] port number of the postgres database

user [str] user name for the postgres database

password: str user password for the postgres database

connection [connection] an open database connection

close_database (*self*)
Closes the sqlite database.

commit (*self*)
Sends a commit to the database.

create_database (*self*)
Creates an empty database if not exists.

create_index (*self*, *cardinality*)
Create an index for the table with the given cardinality.

Parameters

cardinality [int] The cardinality to create a index for.

delete_index (*self*, *cardinality*)
Delete index for the table with the given cardinality.

Parameters

cardinality [int] The cardinality of the index to delete.

execute_sql (*self*, *query*)
Executes a given query string on an open postgres database.

open_database (*self*)
Opens the sqlite database.

reset_database (*self*)
Re-create an empty database.

class pressagio.dbconnector.**SqliteDatabaseConnector** (*dbname*, *cardinality=1*)
Database connector for sqlite databases.

Methods

close_database(self)	Closes the sqlite database.
commit(self)	Sends a commit to the database.
create_bigram_table(self)	Creates a table for n-grams of cardinality 2.
create_index(self, cardinality)	Create an index for the table with the given cardinality.
create_ngram_table(self, cardinality)	Creates a table for n-gram of a give cardinality.
create_trigram_table(self)	Creates a table for n-grams of cardinality 3.
create_unigram_table(self)	Creates a table for n-grams of cardinality 1.
delete_index(self, cardinality)	Delete index for the table with the given cardinality.
delete_ngram_table(self, cardinality)	Deletes the table for n-gram of a give cardinality.
execute_sql(self, query)	Executes a given query string on an open sqlite database.
insert_ngram(self, ngram, count)	Inserts a given n-gram with count into the database.
ngram_count(self, ngram)	Gets the count for a given ngram from the database.
ngrams(self[, with_counts])	Returns all ngrams that are in the table.
open_database(self)	Opens the sqlite database.
remove_ngram(self, ngram)	Removes a given ngram from the databae.
update_ngram(self, ngram, count)	Updates a given ngram in the database.

increment_ngram_count	
ngram_like_table	
ngram_like_table_filtered	
unigram_counts_sum	

`__init__`(*self*, *dbname*, *cardinality=1*)
Constructor for the sqlite database connector.

Parameters

dbname [str] path to the database file

cardinality [int] default cardinality for n-grams

`close_database`(*self*)
Closes the sqlite database.

`commit`(*self*)
Sends a commit to the database.

`execute_sql`(*self*, *query*)
Executes a given query string on an open sqlite database.

`open_database`(*self*)
Opens the sqlite database.

3.1.6 pressagio.predictor

Classes for predictors and to handle suggestions and predictions.

class `pressagio.predictor.`**`Prediction`**
Class for predictions from predictors.

Methods

`append`(self, object, /)	Append object to the end of the list.
`clear`(self, /)	Remove all items from list.
`copy`(self, /)	Return a shallow copy of the list.
`count`(self, value, /)	Return number of occurrences of value.
`extend`(self, iterable, /)	Extend list by appending elements from the iterable.
`index`(self, value[, start, stop])	Return first index of value.
`insert`(self, index, object, /)	Insert object before index.
`pop`(self[, index])	Remove and return item at index (default last).
`remove`(self, value, /)	Remove first occurrence of value.
`reverse`(self, /)	Reverse *IN PLACE*.
`sort`(self, /, *[, key, reverse])	Stable sort *IN PLACE*.

add_suggestion	
suggestion_for_token	

`__init__`(*self*)
Initialize self. See help(type(self)) for accurate signature.

class `pressagio.predictor.`**`Predictor`**(*config*, *context_tracker*, *predictor_name*, *short_desc=None*, *long_desc=None*)

Base class for predictors.

Methods

token_satifies_filter	

`__init__`(*self*, *config*, *context_tracker*, *predictor_name*, *short_desc=None*, *long_desc=None*)

Initialize self. See help(type(self)) for accurate signature.

class `pressagio.predictor.`**`PredictorActivator`**(*config*, *registry*, *context_tracker*)

PredictorActivator starts the execution of the active predictors, monitors their execution and collects the predictions returned, or terminates a predictor's execution if it execedes its maximum prediction time.

The predictions returned by the individual predictors are combined into a single prediction by the active Combiner.

Attributes

`combination_policy`	The combination_policy property.

Methods

predict	

`__init__`(*self*, *config*, *registry*, *context_tracker*)

Initialize self. See help(type(self)) for accurate signature.

`combination_policy`

The combination_policy property.

class `pressagio.predictor.`**`PredictorRegistry`**(*config*, *dbconnection=None*)

Manages instantiation and iteration through predictors and aids in generating predictions and learning.

PredictorRegitry class holds the active predictors and provides the interface required to obtain an iterator to the predictors.

The standard use case is: Predictor obtains an iterator from PredictorRegistry and invokes the predict() or learn() method on each Predictor pointed to by the iterator.

Predictor registry should eventually just be a simple wrapper around plump.

Attributes

`context_tracker`	The context_tracker property.

Methods

`append`(self, object, /)	Append object to the end of the list.
`clear`(self, /)	Remove all items from list.
`copy`(self, /)	Return a shallow copy of the list.
`count`(self, value, /)	Return number of occurrences of value.
`extend`(self, iterable, /)	Extend list by appending elements from the iterable.

Continued on next page

Table 5 – continued from previous page

`index`(self, value[, start, stop])	Return first index of value.
`insert`(self, index, object, /)	Insert object before index.
`pop`(self[, index])	Remove and return item at index (default last).
`remove`(self, value, /)	Remove first occurrence of value.
`reverse`(self, /)	Reverse *IN PLACE*.
`sort`(self, /, *[, key, reverse])	Stable sort *IN PLACE*.

add_predictor	
close_database	
set_predictors	

`__init__`(*self, config, dbconnection=None*)
Initialize self. See help(type(self)) for accurate signature.

`context_tracker`
The context_tracker property.

exception `pressagio.predictor.`**`PredictorRegistryException`**

class `pressagio.predictor.`**`SmoothedNgramPredictor`**(*config, context_tracker, predictor_name, short_desc=None, long_desc=None, dbconnection=None*)

Calculates prediction from n-gram model in sqlite database. You have to create a database with the script *text2ngram* first.

Attributes

`database` The database property.

`deltas` The deltas property.

`learn_mode` The learn_mode property.

Methods

close_database	
init_database_connector_if_ready	
ngram_to_string	
predict	
token_satifies_filter	

`__init__`(*self, config, context_tracker, predictor_name, short_desc=None, long_desc=None, dbconnection=None*)
Initialize self. See help(type(self)) for accurate signature.

`database`
The database property.

`deltas`
The deltas property.

`learn_mode`
The learn_mode property.

class `pressagio.predictor.`**`Suggestion`**(*word*, *probability*)
Class for a simple suggestion, consists of a string and a probility for that string.

Attributes

`probability`	The probability property.

`__init__`(*self*, *word*, *probability*)
Initialize self. See help(type(self)) for accurate signature.

`probability`
The probability property.

exception `pressagio.predictor.`**`SuggestionException`**

exception `pressagio.predictor.`**`UnknownCombinerException`**

3.1.7 pressagio.tokenizer

Several classes to tokenize text.

class `pressagio.tokenizer.`**`ForwardTokenizer`**(*text*, *blankspaces=' x0cnrtx0bx85xa0u2009'*, *separators=''‘~!@#$%^&*()_+=\|]}[{";:/?.>, <¡¿†¨„“”«»——´’‘,0123456789'*)

Methods

`count_characters`(self)	Counts the number of unicode characters in the IO stream.
`is_blankspace`(self, char)	Test if a character is a blankspace.
`is_separator`(self, char)	Test if a character is a separator.

count_tokens	
has_more_tokens	
next_token	
progress	
reset_stream	

`__init__`(*self*, *text*, *blankspaces=' x0cnrtx0bx85xa0u2009'*, *separators=''‘~!@#$%^&*()_+=\|]}[{";:/?.>, <¡¿†¨„“”«»——´’‘,0123456789'*)
Constructor of the Tokenizer base class.

Parameters

text [str] The text to tokenize.

blankspaces [str] The characters that represent empty spaces.

separators [str] The characters that separate token units (e.g. word boundaries).

`count_characters`(*self*)
Counts the number of unicode characters in the IO stream.

class `pressagio.tokenizer.`**`NgramMap`**
A memory efficient store for ngrams.

This class is optimized for memory consumption, it might be slower than other ngram stores. It is also optimized for a three step process:

1) Add all ngrams.
2) Perform a cutoff opertation (optional).
3) Read list of ngrams.

It might not perform well for other use cases.

Methods

`add`(self, ngram_indices)	Add an ngram to the store.
`add_token`(self, token)	Add a token to the internal string store.
`cutoff`(self, cutoff)	Perform a cutoff on the ngram store.
`items`(self)	Get the ngrams from the store.

`__init__`(*self*)
Initialize internal data stores.

`add`(*self*, *ngram_indices*)
Add an ngram to the store.

This will add a list of strings as an ngram to the ngram store. In our standard use case the strings are the indices of the strings, you can get those from the *add_token()* method.

Parameters

list of str The indices of the ngram strings as string.

`add_token`(*self*, *token*)
Add a token to the internal string store.

This will only add the token to the internal strings store. It will return an index that you can use to create your ngram.

The ngrams a are represented as strings of the indices, so we will return a string here so that the consumer does not have to do the conversion.

Parameters

token [str] The token to add to the string store.

Returns

str The index of the token as a string.

`cutoff`(*self*, *cutoff*)
Perform a cutoff on the ngram store.

This will remove all ngrams that have a frequency with the given cutoff or lower.

Parameters

cutoff [int] The cutoff value, we will remove all items with a frequency of the cutoff or lower.

`items`(*self*)
Get the ngrams from the store.

Returns

iterable of tokens, count The tokens are a list of strings, the real tokens that you added to the store via *add_token()*. The count is the the count value for that ngram.

class pressagio.tokenizer.**ReverseTokenizer**(text, blankspaces=’ x0cnrtx0bx85xa0u2009’, separators=’‘~!@#$%^&*()_+=\|]}[{";:/?.>, <¡¿†¨„“”«»———´’‘,0123456789’)

Methods

count_characters(self)	Counts the number of unicode characters in the IO stream.
is_blankspace(self, char)	Test if a character is a blankspace.
is_separator(self, char)	Test if a character is a separator.

count_tokens	
has_more_tokens	
next_token	
progress	
reset_stream	

__init__(self, text, blankspaces=’ x0cnrtx0bx85xa0u2009’, separators=’‘~!@#$%^&*()_+=\|]}[{";:/?.>, <¡¿†¨„“”«»———´’‘,0123456789’)
Constructor of the Tokenizer base class.

Parameters

text [str] The text to tokenize.

blankspaces [str] The characters that represent empty spaces.

separators [str] The characters that separate token units (e.g. word boundaries).

count_characters(self)
Counts the number of unicode characters in the IO stream.

class pressagio.tokenizer.**Tokenizer**(text, blankspaces=’ x0cnrtx0bx85xa0u2009’, separators=’‘~!@#$%^&*()_+=\|]}[{";:/?.>, <¡¿†¨„“”«»———´’‘,0123456789’)
Base class for all tokenizers.

Methods

is_blankspace(self, char)	Test if a character is a blankspace.
is_separator(self, char)	Test if a character is a separator.

count_characters	
count_tokens	
has_more_tokens	
next_token	
progress	
reset_stream	

__init__(*self*, *text*, *blankspaces=' x0cnrtx0bx85xa0u2009'*, *separators=''~!@#$%^&*()_+=\|]}[{";:/?.>, <¡¿†¨„“”«»——´’‘,0123456789'*)

Constructor of the Tokenizer base class.

Parameters

text [str] The text to tokenize.

blankspaces [str] The characters that represent empty spaces.

separators [str] The characters that separate token units (e.g. word boundaries).

is_blankspace(*self*, *char*)

Test if a character is a blankspace.

Parameters

char [str] The character to test.

Returns

ret [bool] True if character is a blankspace, False otherwise.

is_separator(*self*, *char*)

Test if a character is a separator.

Parameters

char [str] The character to test.

Returns

ret [bool] True if character is a separator, False otherwise.

pressagio.tokenizer.**forward_tokenize_file**(*infile: str*, *ngram_size: int*, *lowercase: bool = False*, *cutoff: int = 0*, *ngram_map: pressagio.tokenizer.NgramMap = None*)

Tokenize a file and return an ngram store.

Parameters

infile [str] The file to parse.

ngram_size [int] The size of the ngrams to generate.

lowercase [bool] Whether or not to lowercase all tokens.

cutoff [int] Perform a cutoff after parsing. We will only return ngrams that have a frequency higher than the cutoff.

ngram_map [NgramMap] Pass an existing NgramMap if you want to add the ngrams of the given file to the store. Will create a new NgramMap if *None*.

Returns

NgramMap The ngram map that allows you to iterate over the ngrams.

pressagio.tokenizer.**forward_tokenize_files**(*infiles: List[str], ngram_size: int, lowercase: bool = False, cutoff: int = 0*)

Tokenize a list of file and return an ngram store.

Parameters

infile [str] The file to parse.

ngram_size [int] The size of the ngrams to generate.

lowercase [bool] Whether or not to lowercase all tokens.

cutoff [int] Perform a cutoff after parsing. We will only return ngrams that have a frequency higher than the cutoff.

Returns

NgramMap The ngram map that allows you to iterate over the ngrams.

CHAPTER 4

Indices and tables

- genindex
- modindex

Python Module Index

p

pressagio.callback, 7
pressagio.character, 7
pressagio.combiner, 7
pressagio.context_tracker, 8
pressagio.dbconnector, 8
pressagio.predictor, 13
pressagio.tokenizer, 16

Index

Symbols

__init__() (*pressagio.callback.Callback method*), 7
__init__() (*pressagio.combiner.Combiner method*), 8
__init__() (*pressagio.combiner.MeritocracyCombiner method*), 8
__init__() (*pressagio.context_tracker.ContextTracker method*), 8
__init__() (*pressagio.dbconnector.DatabaseConnector method*), 9
__init__() (*pressagio.dbconnector.PostgresDatabaseConnector method*), 11
__init__() (*pressagio.dbconnector.SqliteDatabaseConnector method*), 13
__init__() (*pressagio.predictor.Prediction method*), 13
__init__() (*pressagio.predictor.Predictor method*), 14
__init__() (*pressagio.predictor.PredictorActivator method*), 14
__init__() (*pressagio.predictor.PredictorRegistry method*), 15
__init__() (*pressagio.predictor.SmoothedNgramPredictor method*), 15
__init__() (*pressagio.predictor.Suggestion method*), 16
__init__() (*pressagio.tokenizer.ForwardTokenizer method*), 16
__init__() (*pressagio.tokenizer.NgramMap method*), 17
__init__() (*pressagio.tokenizer.ReverseTokenizer method*), 18
__init__() (*pressagio.tokenizer.Tokenizer method*), 18

A

add() (*pressagio.tokenizer.NgramMap method*), 17
add_token() (*pressagio.tokenizer.NgramMap method*), 17

C

Callback (*class in pressagio.callback*), 7
close_database() (*pressagio.dbconnector.PostgresDatabaseConnector method*), 11
close_database() (*pressagio.dbconnector.SqliteDatabaseConnector method*), 13
combination_policy (*pressagio.predictor.PredictorActivator attribute*), 14
Combiner (*class in pressagio.combiner*), 7
commit() (*pressagio.dbconnector.PostgresDatabaseConnector method*), 12
commit() (*pressagio.dbconnector.SqliteDatabaseConnector method*), 13
context_tracker (*pressagio.predictor.PredictorRegistry attribute*), 15
ContextTracker (*class in pressagio.context_tracker*), 8
count_characters() (*pressagio.tokenizer.ForwardTokenizer method*), 16
count_characters() (*pressagio.tokenizer.ReverseTokenizer method*), 18
create_bigram_table() (*pressagio.dbconnector.DatabaseConnector method*), 9
create_database() (*pressagio.dbconnector.PostgresDatabaseConnector method*), 12
create_index() (*pressa-*

gio.dbconnector.DatabaseConnector method), 9
create_index() (*pressagio.dbconnector.PostgresDatabaseConnector method*), 12
create_ngram_table() (*pressagio.dbconnector.DatabaseConnector method*), 9
create_trigram_table() (*pressagio.dbconnector.DatabaseConnector method*), 9
create_unigram_table() (*pressagio.dbconnector.DatabaseConnector method*), 9
cutoff() (*pressagio.tokenizer.NgramMap method*), 17

D

database (*pressagio.predictor.SmoothedNgramPredictor attribute*), 15
DatabaseConnector (*class in pressagio.dbconnector*), 8
delete_index() (*pressagio.dbconnector.DatabaseConnector method*), 9
delete_index() (*pressagio.dbconnector.PostgresDatabaseConnector method*), 12
delete_ngram_table() (*pressagio.dbconnector.DatabaseConnector method*), 10
deltas (*pressagio.predictor.SmoothedNgramPredictor attribute*), 15

E

execute_sql() (*pressagio.dbconnector.PostgresDatabaseConnector method*), 12
execute_sql() (*pressagio.dbconnector.SqliteDatabaseConnector method*), 13

F

forward_tokenize_file() (*in module pressagio.tokenizer*), 19
forward_tokenize_files() (*in module pressagio.tokenizer*), 19
ForwardTokenizer (*class in pressagio.tokenizer*), 16

I

insert_ngram() (*pressagio.dbconnector.DatabaseConnector method*), 10
InvalidCallbackException, 8
is_blankspace() (*pressagio.tokenizer.Tokenizer method*), 19
is_separator() (*pressagio.tokenizer.Tokenizer method*), 19
items() (*pressagio.tokenizer.NgramMap method*), 17

L

learn_mode (*pressagio.predictor.SmoothedNgramPredictor attribute*), 15

M

MeritocracyCombiner (*class in pressagio.combiner*), 8

N

ngram_count() (*pressagio.dbconnector.DatabaseConnector method*), 10
NgramMap (*class in pressagio.tokenizer*), 16
ngrams() (*pressagio.dbconnector.DatabaseConnector method*), 10

O

open_database() (*pressagio.dbconnector.PostgresDatabaseConnector method*), 12
open_database() (*pressagio.dbconnector.SqliteDatabaseConnector method*), 13

P

PostgresDatabaseConnector (*class in pressagio.dbconnector*), 10
Prediction (*class in pressagio.predictor*), 13
Predictor (*class in pressagio.predictor*), 13
PredictorActivator (*class in pressagio.predictor*), 14
PredictorRegistry (*class in pressagio.predictor*), 14
PredictorRegistryException, 15
pressagio.callback (*module*), 7
pressagio.character (*module*), 7
pressagio.combiner (*module*), 7
pressagio.context_tracker (*module*), 8
pressagio.dbconnector (*module*), 8
pressagio.predictor (*module*), 13
pressagio.tokenizer (*module*), 16
probability (*pressagio.predictor.Suggestion attribute*), 16

R

remove_ngram() (*pressagio.dbconnector.DatabaseConnector method*), 10

`reset_database()` (*pressagio.dbconnector.PostgresDatabaseConnector method*), 12

`ReverseTokenizer` (*class in pressagio.tokenizer*), 18

S

`SmoothedNgramPredictor` (*class in pressagio.predictor*), 15

`SqliteDatabaseConnector` (*class in pressagio.dbconnector*), 12

`Suggestion` (*class in pressagio.predictor*), 15

`SuggestionException`, 16

T

`Tokenizer` (*class in pressagio.tokenizer*), 18

U

`UnknownCombinerException`, 16

`update_ngram()` (*pressagio.dbconnector.DatabaseConnector method*), 10