pressagio Documentation

Release 0.1.3

Peter Bouda

September 24, 2014

Contents

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

API documentation

1.1 pressagio Package

1.1.1 pressagio.callback

Base class for callbacks.

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

Methods

`future_stream`()	
`past_stream`()	
`update`(character)	

`__init__`()

1.1.2 pressagio.character

1.1.3 pressagio.combiner

Combiner classes to merge results from several predictors.

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

Methods

`combine`()	
`filter`(prediction)	

`__init__`()

1.1.4 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(index, change)	
future_stream()	
is_completion_valid(completion)	
past_stream()	
prefix()	
token(index)	
update_context()	

__init__(*config*, *predictor_registry*, *callback*)

1.1.5 pressagio.dbconnector

Classes to connect to databases.

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

Methods

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

__init__(*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()
Creates a table for n-grams of cardinality 2.

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

Parameters **cardinality** : int

The cardinality to create a index for.

create_ngram_table(*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()
Creates a table for n-grams of cardinality 3.

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

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

Parameters **cardinality** : int

The cardinality of the index to delete.

delete_ngram_table(*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(*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(*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(*with_counts=False*)
Returns all ngrams that are in the table.

Parameters **None**

Returns **ngrams** : generator

A generator for ngram tuples.

remove_ngram(*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(*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=u'localhost'*, *port=5432*, *user=u'postgres'*, *password=None*, *connection=None*)

Database connector for postgres databases.

Methods

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

Continued on next page

Table 1.5 – continued from previous page

remove_ngram(ngram)	Removes a given ngram from the databae.
reset_database()	Re-create an empty database.
unigram_counts_sum()	
update_ngram(ngram, count)	Updates a given ngram in the database.

__init__(*dbname*, *cardinality=1*, *host=u'localhost'*, *port=5432*, *user=u'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()

Closes the sqlite database.

commit()

Sends a commit to the database.

create_database()

Creates an empty database if not exists.

create_index(*cardinality*)

Create an index for the table with the given cardinality.

Parameters **cardinality** : int

The cardinality to create a index for.

delete_index(*cardinality*)

Delete index for the table with the given cardinality.

Parameters **cardinality** : int

The cardinality of the index to delete.

execute_sql(*query*)

Executes a given query string on an open postgres database.

open_database()

Opens the sqlite database.

reset_database()
Re-create an empty database.

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

Methods

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

__init__(*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()
Closes the sqlite database.

commit()
Sends a commit to the database.

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

open_database()
Opens the sqlite database.

1.1.6 pressagio.predictor

Classes for predictors and to handle suggestions and predictions.

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

Methods

add_suggestion(suggestion)	
append	L.append(object) – append object to end
count(...)	
extend	L.extend(iterable) – extend list by appending elements from the iterable
index((value, [start, ...)	Raises ValueError if the value is not present.
insert	L.insert(index, object) – insert object before index
pop(...)	Raises IndexError if list is empty or index is out of range.
remove	L.remove(value) – remove first occurrence of value.
reverse	L.reverse() – reverse *IN PLACE*
sort	L.sort(cmp=None, key=None, reverse=False) – stable sort *IN PLACE*;
suggestion_for_token(token)	

__init__()

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

Methods

token_satifies_filter(token, prefix, ...)

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

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([multiplier, prediction_filter])

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

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

add_predictor(predictor_name)	
append	L.append(object) – append object to end
close_database()	
count(...)	
extend	L.extend(iterable) – extend list by appending elements from the iterable
index((value, [start, ...)	Raises ValueError if the value is not present.
insert	L.insert(index, object) – insert object before index
pop(...)	Raises IndexError if list is empty or index is out of range.
remove	L.remove(value) – remove first occurrence of value.
reverse	L.reverse() – reverse *IN PLACE*
set_predictors()	
sort	L.sort(cmp=None, key=None, reverse=False) – stable sort *IN PLACE*;

__init__(*config*, *dbconnection=None*)

context_tracker

The context_tracker property.

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(ngram)	
predict(max_partial_prediction_size, filter)	
token_satifies_filter(token, prefix, ...)	

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

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__(*word*, *probability*)

probability
The probability property.

1.1.7 pressagio.tokenizer

Several classes to tokenize text.

class pressagio.tokenizer.**Tokenizer**(*stream*, *blankspaces=u' x0cnrtx0bx85'*, *separators=u'`~!@#$%^&*()_-+=\|]}[{'";:/?.>*, *<u2020u201eu201cu0964u0965u05d5u2013xb4u2019u2018u201au05d9012345678*
Base class for all tokenizers.

Methods

count_characters()	
count_tokens()	
has_more_tokens()	
is_blankspace(char)	Test if a character is a blankspace.
is_separator(char)	Test if a character is a separator.
next_token()	
progress()	
reset_stream()	

__init__(*stream*, *blankspaces=u' x0cnrtx0bx85'*, *separators=u'‘~!@#$%^&*()_-+=\|]}[{‘";:/?.>*, *<u2020u201eu201cu0964u0965u05d5u2013xb4u2019u2018u201au05d90123456789u0903'*)

Constructor of the Tokenizer base class.

Parameters **stream** : str or io.IOBase

The stream to tokenize. Can be a filename or any open IO stream.

blankspaces : str

The characters that represent empty spaces.

separators : str

The characters that separate token units (e.g. word boundaries).

is_blankspace(*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(*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.

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*

p