ticclat Documentation

Release 0.2.2

Janneke van der Zwaan, Patrick Bos, Tom Klaver

Jan 15, 2020

Contents:

CHAPTER 1

API Reference

1.1 ticclat package

Documentation about TICCLAT

1.1.1 Subpackages

ticclat.flask_app package

Subpackages

ticclat.flask_app.plots package

Submodules

ticclat.flask_app.plots.blueprint module

ticclat.flask_app.plots.corpus_size module

ticclat.flask_app.plots.lexicon_size module

ticclat.flask_app.plots.paradigm_size module

ticclat.flask_app.plots.word_count_per_year module

Submodules

1.1.2 Submodules

ticclat.dbutils module

Collection of database access functions.

ticclat.dbutils.**add_lexicon**(*session*, *lexicon_name*, *vocabulary*, *wfs*, *preprocess_wfs=True*)
wfs is pandas DataFrame with the same column names as the database table, in this case just "wordform"

ticclat.dbutils.**add_lexicon_with_links**(*session*, *lexicon_name*, *vocabulary*, *wfs*, *from_column*, *to_column*, *from_correct*, *to_correct*, *batch_size=50000*, *preprocess_wfs=True*, *to_add=None*)
Add wordforms from a lexicon with links to the database.

Lexica with links contain wordform pairs that are linked. The *wfs* dataframe must contain two columns: the *from_column* and the *to_column*, which contains the two words of each pair (per row). Using the arguments *from_correct* and *to_correct*, you can indicate whether the columns of this dataframe contain correct words or not (boolean). Typically, there are two types of linked lexica: True + True, meaning it links correct wordforms (e.g. morphological variants) or True + False, meaning it links correct wordforms to incorrect ones (e.g. a spelling correction list).

ticclat.dbutils.**add_morphological_paradigms**(*session*, *in_file*)
Add morphological paradigms to database from CSV file.

ticclat.dbutils.**add_ticcl_variants**(*session*, *name*, *df*, ***kwargs*)
Add TICCL variants as a linked lexicon.

ticclat.dbutils.**bulk_add_anahashes**(*session*, *anahashes*, *tqdm_factory=None*, *batch_size=10000*)
anahashes is pandas dataframe with the column wordform (index), anahash

ticclat.dbutils.**bulk_add_wordforms**(*session*, *wfs*, *preprocess_wfs=True*)
wfs is pandas DataFrame with the same column names as the database table, in this case just "wordform"

ticclat.dbutils.**connect_anahashes_to_wordforms**(*session*, *anahashes*, *df*, *batch_size=50000*)
Create the relation between wordforms and anahashes in the database.

Given *anahashes*, a dataframe with wordforms and corresponding anahashes, create the relations between the two in the wordforms and anahashes tables by setting the anahash_id foreign key in the wordforms table.

ticclat.dbutils.**create_ticclat_database**(*delete_existing=False*)
Create the TICCLAT database.

Sets the proper encoding settings and uses the schema to create tables.

ticclat.dbutils.**create_wf_frequencies_table**(*session*)
Create wordform_frequencies table in the database.

The text_attestations frequencies are summed and stored in this table. This can be used to save time when needing total-database frequencies.

ticclat.dbutils.**empty_table**(*session*, *table_class*)
Empty a database table.

- table_class: the ticclat_schema class corresponding to the table

ticclat.dbutils.**get_anahashes**(*session*, *anahashes*, *wf_mapping*, *batch_size=50000*)
Generator of dictionaries with anahash ID and wordform ID pairs.

Given *anahashes*, a dataframe with wordforms and corresponding anahashes, yield dictionaries containing two entries each: key 'a_id' has the value of the anahash ID in the database, key 'wf_id' has the value of the wordform ID in the database.

ticclat.dbutils.**get_db_name**()
Get the database name from the DATABASE_URL environment variable.

ticclat.dbutils.**get_engine**(*without_database=False*)
Create an sqlalchemy engine using the DATABASE_URL environment variable.

ticclat.dbutils.**get_or_create_wordform**(*session*, *wordform*, *has_analysis=False*, *wordform_id=None*)
Get a Wordform object of wordform.

The Wordform object is an sqlalchemy field defined in the ticclat schema. It is coupled to the entry of the given wordform in the wordforms database table.

ticclat.dbutils.**get_session**()
Return an sqlalchemy session object using a sessionmaker from get_session_maker().

ticclat.dbutils.**get_session_maker**()
Return an sqlalchemy sessionmaker object using an engine from get_engine().

ticclat.dbutils.**get_wf_mapping**(*session*, *lexicon=None*, *lexicon_id=None*)
Create a dictionary with a mapping of wordforms to wordform_id.

The keys of the dictionary are wordforms, the values are the IDs of those wordforms in the database wordforms table.

ticclat.dbutils.**get_word_frequency_df**(*session*, *add_ids=False*)
Can be used as input for ticcl-anahash.

Returns

Pandas DataFrame containing wordforms as index and a frequency value as column, or None if all wordforms in the database already are connected to an anahash value

ticclat.dbutils.**session_scope**(*session_maker*)
Provide a transactional scope around a series of operations.

ticclat.dbutils.**update_anahashes**(*session*, *alphabet_file*, *tqdm_factory=None*, *batch_size=50000*)
Add anahashes for all wordforms that do not have an anahash value yet.

Requires ticcl to be installed!

Inputs: session: SQLAlchemy session object. alphabet_file (str): the path to the alphabet file for ticcl.

ticclat.dbutils.**update_anahashes_new**(*session*, *alphabet_file*)
Add anahashes for all wordforms that do not have an anahash value yet.

Requires ticcl to be installed!

Inputs: session: SQLAlchemy session object. alphabet_file (str): the path to the alphabet file for ticcl.

ticclat.dbutils.**write_wf_links_data**(*session*, *wf_mapping*, *links_df*, *wf_from_name*, *wf_to_name*, *lexicon_id*, *wf_from_correct*, *wf_to_correct*, *links_file*, *sources_file*, *add_columns=None*)
Write wordform links (obtained from lexica) to JSON files for later processing.

Two JSON files will be written to: *links_file* and *sources_file*. The links file contains only the wordform links and corresponds to the wordform_links database table. The sources file contains the source lexicon of each link and also whether either wordform is considered a "correct" form or not, which is defined by the lexicon (whether it is a "dictionary" with only correct words or a correction list with correct words in one column and incorrect ones in the other).

ticclat.dev_utils module

Utilities used while developing TICCLAT

ticclat.dev_utils.**delete_lexicon**(*session*, *lexicon_id*)
Delete a lexicon from the database.

ticclat.sacoreutils module

SQLAlchemy core utility functionality

Functionality for faster bulk inserts without using the ORM. More info: https://docs.sqlalchemy.org/en/latest/faq/performance.html

ticclat.sacoreutils.**add_corpus_core**(*session*, *corpus_matrix*, *vectorizer*, *corpus_name*, *document_metadata=Empty DataFrame Columns: [] Index: []*, *batch_size=50000*)
Add a corpus to the database.

A corpus is a collection of documents, which is a collection of words. This function adds all words as wordforms to the database, records their "attestation" (the fact that they occur in a certain document and with what frequency), adds the documents they belong to, adds the corpus and adds the corpus ID to the documents.

Inputs: session: SQLAlchemy session (e.g. from *dbutils.get_session*) corpus_matrix: the dense corpus term-document matrix, like from

tokenize.terms_documents_matrix_ticcl_frequency

vectorizer: the terms in the term-document matrix, as given by *tokenize.terms_documents_matrix_ticcl_frequency*

corpus_name: the name of the corpus in the database document_metadata: see *ticclat_schema.Document* for all the possible

metadata. Make sure the index of this dataframe matches with the document identifiers in the term- document matrix, which can be easily achieved by resetting the index for a Pandas dataframe.

batch_size: batch handling of wordforms to avoid memory issues.

ticclat.sacoreutils.**bulk_add_anahashes_core**(*engine*, *iterator*, ***kwargs*)
Insert anahashes in *iterator* in batches into anahashes database table.

Convenience wrapper around *sql_insert_batches* for anagram hashes. Take care: no session is used, so relationships can't be added automatically.

ticclat.sacoreutils.**bulk_add_textattestations_core**(*engine*, *iterator*, ***kwargs*)
Insert text attestations in *iterator* in batches into text_attestations database table.

Convenience wrapper around *sql_insert_batches* for text attestations. Take care: no session is used, so relationships can't be added automatically.

ticclat.sacoreutils.**bulk_add_wordforms_core**(*engine*, *iterator*, ***kwargs*)
Insert wordforms in *iterator* in batches into wordforms database table.

Convenience wrapper around *sql_insert_batches* for wordforms. Take care: no session is used, so relationships can't be added automatically.

ticclat.sacoreutils.**get_engine**(*user*, *password*, *dbname*, *dburl='mysql://{}:{}@localhost/{}?charset=utf8mb4'*)
Returns an engine that can be used for fast bulk inserts

ticclat.sacoreutils.**get_tas**(*corpus*, *doc_ids*, *wf_mapping*, *word_from_tdmatrix_id*)
Get term attestation from wordform frequency matrix.

Term attestation records the occurrence and frequency of a word in a given document.

Inputs:

corpus: the dense corpus term-document matrix, like from *tokenize.terms_documents_matrix_ticcl_frequency*

doc_ids: list of indices of documents in the term-document matrix wf_mapping: dictionary mapping wordforms (key) to database wordform_id word_from_tdmatrix_id: mapping of term-document matrix column index

(key) to wordforms (value)

ticclat.sacoreutils.**sql_insert**(*engine*, *table_object*, *to_insert*)
Insert a list of objects into the database without using a session.

This is a fast way of (mass) inserting objects. However, because no session is used, no relationships can be added automatically. So, use with care!

This function is a simplified version of test_sqlalchemy_core from here: https://docs.sqlalchemy.org/en/13/faq/performance.html#i-m-inserting-400-000-rows-with-the-orm-and-it-s-really-slow

Inputs: engine: SQLAlchemy engine or session table_object: object representing a table in the database (i.e., one

of the objects from ticclat_schema)

to_insert (list of dicts): list containg dictionary representations of the objects (rows) to be inserted

ticclat.sacoreutils.**sql_insert_batches**(*engine*, *table_object*, *iterator*, *total=0*, *batch_size=10000*)
Insert items in *iterator* in batches into database table.

Take care: no session is used, so relationships can't be added automatically.

Inputs:

table_object: the ticclat_schema object corresponding to the database table.

total: used for tqdm, since iterator will often be a generator, which has no predefined length.

ticclat.sacoreutils.**sql_query_batches**(*engine*, *query*, *iterator*, *total=0*, *batch_size=10000*)
Execute *query* on items in *iterator* in batches.

Take care: no session is used, so relationships can't be added automatically.

Inputs:

total: used for tqdm, since iterator will often be a generator, which has no predefined length.

ticclat.ticclat_schema module

SQLAlchemy schema of the TICCLAT database.

Contains all the tables of the database and their connections, defined as SQLAlchemy declarative_base subclasses.

Many of the tables here defined are based on an INT lexicon database created in the IMPACT project (https://ivdnt.org/images/stories/onderzoek_en_onderwijs/publicaties/impact/impact_lexicon_structure.pdf). See https://github.com/TICCLAT/docs/blob/master/database_design.md for more information about the database design.

Based on this, in TICCLAT, we added tables for: - links between wordforms - morphological paradigm groups of wordforms - anagram hashes from TICCL - spelling variants from TICCL - identifiers linking wordforms to external sources like the WNT, MNW, INT.

class ticclat.ticclat_schema.**Anahash**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Table for storing anahashes.

The anahashes in this table have no direct relation to the wordforms, those links are tracked in the wordforms table. This was done so that the anahashes table can be efficiently searched, e.g. for ranges in anahash "space".

anahash

anahash_id

class ticclat.ticclat_schema.**Corpus**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Table for storing corpus metadata.

corpus_documents

corpus_id

name

class ticclat.ticclat_schema.**Document**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Table for storing document metadata.

author

document_corpora

document_id

document_wordforms

editor

encoding

language

other_languages

parent_document

persistent_id

pub_year

publisher

publishing_location

region

spelling

text_type

title

word_count

year_from

year_to

class ticclat.ticclat_schema.**ExternalLink**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Table for storing ids from external sources of wordforms.

Used for linking wordforms to external sources, such as the WNT, MNW, INT.

external_link_id

source_id

source_name

wordform_id

class ticclat.ticclat_schema.**Lexicon**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Table for storing lexicon metadata.

vocabulary (bool): if True, all words in this lexicon are (supposed to be) valid words, if False, some are misspelled

lexicon_id

lexicon_name

lexicon_wordform_links

lexicon_wordforms

vocabulary

class ticclat.ticclat_schema.**MorphologicalParadigm**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Table for storing information about morphological paradigms of wordforms.

The paradigms are determined according to Reynaert's method (to be published).

V

W

X

Y

Z

paradigm_id

word_type_code

word_type_number

wordform_id

class ticclat.ticclat_schema.**TextAttestation**(*document*, *wordform*, *frequency*)
Bases: sqlalchemy.ext.declarative.api.Base

Table for storing text attestations.

A text attestation entry is defined in the INT schema as the occurrence and frequency of wordforms in documents.

attestation_id

document_id

frequency

ta_document

ta_wordform

wordform_id

class ticclat.ticclat_schema.**TicclatVariant**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Contains spelling variants of words, ingested from TICCL

frequency

levenshtein_distance

ticclat_variant_id

wordform

wordform_source

wordform_source_id

class ticclat.ticclat_schema.**Wordform**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Table for storing wordforms and associated anahashes.

anahash

anahash_id

link(*wordform*)
Add WordformLinks between self and another wordfrom and vice versa.

The WordformLinks are added only in the link does not yet exist.

Inputs: wordform (Wordform): Wordform that is related to Wordform self.

link_spelling_correction(*corr*, *lexicon*)
Add a spelling correction WordformLink.

This method sets the booleans that indicate which Wordforms are correct (according to the lexicon).

Inputs: corr (Wordform): A correction candidate of Wordform self lexicon (Lexicon): The Lexicon that contains the WordformLink

link_with_metadata(*wf_to*, *wf_from_correct*, *wf_to_correct*, *lexicon*)
Add WordformLinks with metadata.

Adds a WordformLink between self and another wordfrom, and vice versa, if these links are not yet in the database. And adds a WordformLinkSource, with Lexicon, and information about which Wordforms are correct according to the Lexicon. No duplicate WordformLinkSources are added.

TODO: add Uniqueconstraint on (wf_from (self), wf_to, lexicon)?

Inputs: wf_to (Wordform): Wordform self will be linked to (and vice versa) wf_from_correct (boolean): True if Wordform self is correct

according to the lexicon, False otherwise.

wf_to_correct (boolean): True if Wordform wf_to is correct according to the lexicon, False otherwise.

lexicon (Lexicon): The Lexicon that contains the WordformLink

wf_lexica

wordform

wordform_documents

wordform_id

wordform_lowercase

class ticclat.ticclat_schema.**WordformFrequencies**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Materialized view containing overall frequencies of wordforms

The data in this table can be used to filter wordforms on frequency. This is necessary, because there is a lot of noise in the wordforms table, and this makes aggregating over all wordforms expensive.

frequency

wordform

wordform_id

class ticclat.ticclat_schema.**WordformLink**(*wf1*, *wf2*)
Bases: sqlalchemy.ext.declarative.api.Base

Table for storing links between wordforms.

linked_from

linked_to

wordform_from

wordform_to

class ticclat.ticclat_schema.**WordformLinkSource**(*wflink*, *wf_from_correct*, *wf_to_correct*, *lexicon*)
Bases: sqlalchemy.ext.declarative.api.Base

Table for storing the sources of links between wordforms.

Wordform links are given by lexica (dictionaries, spelling correction lists, etc.). This table records which lexicon a given link between wordforms was originally ingested from.

anahash_difference

ld

lexicon_id

source_x_wordform_link_id

wfls_lexicon

wfls_wflink

wordform_from

wordform_from_correct

wordform_to

wordform_to_correct

ticclat.tokenize module

Generators that produce term-frequency vectors of documents in a corpus.

A document in ticclat is a term-frequency vector (collections.Counter). This module contains generators that return term-frequency vectors for certain types of input data.

ticclat.tokenize.**do_nothing**(*list_of_words*)
Return the argument unchanged.

ticclat.tokenize.**terms_documents_matrix_ticcl_frequency**(*in_files*)
Returns a terms document matrix and related objects of a corpus

A terms document matrix contains frequencies of wordforms, with wordforms along one matrix axis (columns) and documents along the other (rows).

Inputs:

in_files: list of ticcl frequency files (one per document in the corpus)

Returns

a sparse terms documents matrix vocabulary: the vectorizer object containing the vocabulary (i.e., all word forms

in the corpus)

Return type corpus

ticclat.tokenize.**terms_documents_matrix_word_lists**(*word_lists*)
Returns a terms document matrix and related objects of a corpus

A terms document matrix contains frequencies of wordforms, with wordforms along one matrix axis and documents along the other.

Inputs: word_lists: iterator over lists of words

Returns

a sparse terms documents matrix vocabulary: the vectorizer object containing the vocabulary (i.e., all word forms

in the corpus)

Return type corpus

ticclat.tokenize.**ticcl_frequency**(*in_files*, *max_word_length=255*)
Generate word-frequency pairs from TICCL frequency files.

For each file in *in_files*, open it and yield a dictionary with frequencies (value) for each word (key).

ticclat.utils module

Non-database related utility functions for TICCLAT.

ticclat.utils.**anahash_df**(*wfreq*, *alphabet_file*)
Get anahash values for word frequency data.

The result can be used to add anahash values to the database (ticclat.dbutils.bulk_add_anahashes) and connect wordforms to anahash values (ticclat.dbutils.connect_anahases_to_wordforms).

Inputs:

wfreq (pandas DataFrame): Dataframe containing word frequency data (the result of ticcl.dbutils.get_word_frequency_df)

alphabet_file (str): path to the ticcl alphabet file to use

Returns pandas DataFrame containing the word forms as index and anahash values as column.

ticclat.utils.**chunk_df**(*df*, *batch_size=1000*)
Generator that returns about equally size chunks from a pandas DataFrame

Inputs: df (DataFrame): the DataFrame to be chunked batch_size (int, default 10000): the approximate number of records that will

be in each chunk

ticclat.utils.**chunk_json_lines**(*file_handle*, *batch_size=1000*)
Read a JSON file and yield lines in batches.

ticclat.utils.**count_lines**(*file_handle*)
From https://stackoverflow.com/q/845058/1199693

ticclat.utils.**get_temp_file**()
Create a temporary file and its file handle.

Returns File handle of the temporary file.

ticclat.utils.**iterate_wf**(*lst*)
Generator that yields *{'wordform': value}* for all values in *lst*.

ticclat.utils.**json_line**(*obj*)
Convert an object *obj* to a string containing a line of JSON.

ticclat.utils.**morph_iterator**(*morph_paradigms_per_wordform*, *mapping*)
Generator that yields dicts of morphological paradigm code components plus wordform_id in the database.

Inputs:

morph_paradigms_per_wordform: dictionary with wordforms (keys) and lists (values) of dictionaries of code components (return values of *split_component_code*).

mapping: iterable of named tuples / dictionaries that contain the result of a query on the wordforms table, i.e. fields 'wordform' and 'wordform_id'.

ticclat.utils.**preprocess_wordforms**(*wfs*, *columns=None*)
Clean wordforms in dataframe *wfs*.

Strips whitespace, replaces underscores with asterisks (misc character) and spaces with underscores.

ticclat.utils.**read_json_lines**(*file_handle*)
Generator that reads a dictionary per line from a file

This can be used when doing mass inserts (i.e., inserts not using the ORM) into the database. The data that will be inserted is written to file (using `write_json_lines`), so it can be read and inserted into the database without using a lot of memory.

Inputs:

file_handle: File handle of the file containing the data, one dictionary (JSON) object per line

Returns iterator over the lines in the input file

ticclat.utils.**read_ticcl_variants_file**(*fname*)
Return dataframe containing data in TICCL variants file.

ticclat.utils.**set_logger**(*level='INFO'*)
Configure logging format and level.

ticclat.utils.**split_component_code**(*code*, *wordform*)
Split morphological paradigm code into its components.

Morphological paradigm codes in Reynaert's encoding scheme consist of 8 subcomponents. These are returned as separate entries of a dictionary from this function.

ticclat.utils.**timeit**(*method*)
Decorator for timing methods.

Can be used for benchmarking queries.

Source: https://medium.com/pythonhive/fa04cb6bb36d

ticclat.utils.**write_json_lines**(*file_handle*, *generator*)
Write a sequence of dictionaries to file, one dictionary per line

This can be used when doing mass inserts (i.e., inserts not using the ORM) into the database. The data that will be inserted is written to file, so it can be read (using `read_json_lines`) without using a lot of memory.

Inputs: file_handle: File handle of the file to save the data to generator (generator): Generator that produces objects to write to file

Returns the number of records written.

Return type int

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

CHAPTER 3

Readme

3.1 ticclat

TICCLAT is a tool for text-induced corpus correction and lexical assessment.

3.1.1 Installation

To install ticclat, do:

```
git clone https://github.com/ticclat/ticclat.git
cd ticclat
pip install .
```

Run tests (including coverage) with:

```
python setup.py test
```

3.1.2 Setup MySQL

Server security

Run *sudo mysql_secure_installation* with the following choices:

- Validate passwords: no

- Root password: pick one
- Remove anonymous users: yes
- Disallow root login remotely: no
- Remove test database and access to it: yes
- Reload privilege tables now: yes

To allow login as any user with the root password set above, you have to switch the authentication plugin for root to *mysql_native_password*.

```
SELECT plugin from mysql.user where User='root';
```

what plugin you are using currently. If it is auth_socket (default on Ubuntu), you can only login as root if you are running *mysql* as the Unix root user, e.g. by running with *sudo*. To change it to *mysql_native_password*, start *mysql -u root* and run

```
UPDATE mysql.user SET plugin = 'mysql_native_password' WHERE User = 'root';
```

To make this authentication plugin the default, add the following to */etc/my.cnf* (or another *my.cnf* location, run *mysqladmin –help* to see the locations that *mysqld* looks for):

```
[mysqld]
default-authentication-plugin = mysql_native_password
```

Other settings

To run the ingestion script (e.g. the elex lexicon ingestion), the maximum package size has to be high enough. We set it to 41943040 (4194304 was not enough) by setting the following line in */etc/my.cnf*:

```
[mysqld]
max_allowed_packet = 42M
```

To allow for loading CSV files (this is the fastest way of inserting big bulks of records), add:

```
[mysqld]
local_infile=ON
```

This allows you to run queries like this:

```
LOAD DATA LOCAL INFILE '/file.csv' INTO TABLE test FIELDS TERMINATED BY ',' ENCLOSED␣
↪BY '"' ESCAPED BY '\\';
```

This loads the file */file.csv* **from the client**, sends it to the server which inserts it into table *test*. See [MySQL Load Data Documentation](https://dev.mysql.com/doc/refman/8.0/en/load-data.html).

To allow for saving CSV files, add:

```
[mysqld]
secure_file_priv=/data/tmp/mysql
```

Also, add this to */etc/apparmor.d/usr.sbin.mysqld* (restart afterwards: *sudo systemctl reload apparmor*)

```
# Allow /data/tmp/mysql access
/data/tmp/mysql/ rw,
/data/tmp/mysql/** rw,
```

Make sure the directory /data/tmp/mysql exists and is writable by the *mysql* user.

Ubuntu

On Ubuntu 18.04, the default mysqld settings in */etc/mysql/mysql.conf.d/mysqld.cnf* set the socket to a non-standard location that confuses all the default values in MySQLdb. Change it to */tmp/mysql.sock* if you get *OperationError: 2006 . . .* when running *ticclat* tasks like ingesting corpora or lexica.

Changes to the Database Schema

Important note: Alembic stripts were removed. Use most recent database dumps to get the newest version of the database.

To apply changes to the database schema, we use [alembic](https://alembic.sqlalchemy.org/en/latest/index.html).

Alembic is configured to read the information needed to connect to the database database from environment variable *DATABASE_URL*

To migrate the database to the latest database schema run:

```
alembic upgrade head
```

Important note: if you are creating the database from scratch, **do not** use the alembic database migrations. Instead, use SQLAlchemy to create a complete new instance of the database.

3.1.3 Data ingestion

The ticclat package contains scripts for ingesting data into the database. To run the scripts, create an `.env` file as described under *Setup virtual environment*. In the directory where the *.env* file is located, type *python* and then:

```
>>> from ticclat import ingest
>>> ingest.run()
```

You can conigure `run()` by providing arguments:

- `env`: path to the `.env` file (default: `.env`)
- `reset_db`: delete the database and recreate it before ingesting data (default: `False`)
- `alphabet_file`: path to the alphabet file (required for calculating anahashes; default: */data/ALPH/nld.aspell.dict.clip20.lc.LD3.charconfus.clip20.lc.chars*)
- `batch_size`: size of database batches (default: 5000) (We should look into how this is used.)
- `include`: list of data sources to ingest (default: `[]`)
- `exclude`: list of data sources to exclude from ingesting (default: `[]`)
- `ingest`: boolean indicating whether data should be ingested (default: `True`)
- `anahash`: boolean indicating whether anahashes should be calculated (default: `True`)
- `tmpdir`: directory to use for storing temporary data (default: `/data/tmp`)
- `loglevel`: what log messages to show (default: `INFO`)
- `reset_anahashes` boolean indicating whether the anahashes table should be emptied (default: `False`)
- `base_dir`: path to the directory containing the source datafiles

The following sources can be ingested (and added to the `include` and `exclude` lists):

- `twente`: spelling correction lexicon
- `inl`: lexicon
- `SoNaR500`: corpus
- `elex`: lexicon
- `groene boekje`: lexicon
- `OpenTaal`: lexicon
- `sgd`: Staten Generaal Digitaal, corpus
- `edbo`: Early Dutch Books Online, corpus
- `dbnl`: Digitale Bibliotheek voor de Nederlandse letteren
- `morph_par`: Morphological Paradigms
- `wf_freqs`: Generate materialized view (table) containing wordforms and their total frequencies in the corpora
- `sgd_ticcl`: ingest ticcl corrections based on the SDG data (we currently have data for two wordforms: *Amsterdam* and *Binnenlandsche*)

3.1.4 Flask web app

Preparation

Starting from Ubuntu (18.04), setup the MySQL database. Then clone this directory, install dependencies (*conda* & *libmysqlclient-dev* & *build-essential* e.g. https://docs.conda.io/en/latest/miniconda.html and *apt-get update && apt-get install -y libmysqlclient-dev build-essential*).

Setup virtual environment

```
conda create --name ticclat-web
conda activate ticclat-web
conda install pip
```

From ticclat directory, install it:

Create a *.env* file with the following:

```
DATABASE_URL=mysql://[user]:[pass]@[host]:[port]/[db_name]?charset=utf8mb4&local_
↪infile=1

FLASK_APP=ticclat.flask_app.py
FLASK_ENV=production
FLASK_DEBUG=0

#for DEV:
#FLASK_ENV=development
#FLASK_DEBUG=1
```

You can now run a development server using: *flask run*

Or a production server:

```
export $(cat .env | xargs)
gunicorn ticclat.flask_app.wsgi:app --bind 0.0.0.0:8000 --max-requests 100 --workers␣
↪2 --timeout 30
```

where the last three options may not be necessary, but can be tweaked for stability and performance.

3.1.5 Debugger

If the debugger in e.g. PyCharm isn't working correctly, this might be because test coverage is enabled. Disbable this temporarily by commenting *addopts* line in *setup.cfg*:

3.1.6 Documentation

Include a link to your project's full documentation here.

3.1.7 Contributing

If you want to contribute to the development of ticclat, have a look at the contribution guidelines.

3.1.8 License

3.1.9 Credits

This package was created with Cookiecutter and the NLeSC/python-template.

Python Module Index

t

Index

F

G

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z