CLiC Dickens Documentation

Release 1.4

J. de Joode

August 08, 2016

CONTENTS

Contents:

CHAPTER

ONE

CLIC FOR END-USERS

This section of the documentation is work in progress.

1.1 Definitions

suspension A narratorial interruption of character speech that does not end with sentence final punctuation.

non-quote Any textual unit that is not a quote.

quote A textual unit that starts and ends with respectively single or double quotation marks. It can represent speech, writing, or thought.

Rosa has been crying and is yet in distress. On her coming in, the ironmaster leaves his chair, takes her arm in his, and remains with her near the door ready to depart.

"You are taken charge of, you see," says my Lady in her weary manner, "and are going away well protected. I have mentioned that you are a very good girl, and you have nothing to cry for."

"She seems after all," observes Mr. Tulkinghorn, loitering a little forward with his hands behind him, "as if she were crying at going away."

1.2 The underlying annotation

1.3 The corpora

1.4 How to use ...

1.4.1 the concordance

1.4.2 clusters

1.4.3 keywords

1.4.4 subsets

1.4.5 patterns

1.4.6 user annotation

CHAPTER

TWO

CLIC FOR ADMINISTRATORS

For the deployment of CLiC we rely on Docker. This has several benefits: it packages all the dependencies of CLiC together in a simple image and it makes a deployment much faster and platform-independent.

2.1 Installing CLiC on your own server

2.1.1 Step 1: Install Docker on a vanilla Ubuntu server

```
# Install Docker in a way that can easily be upgraded
sudo apt-get update

sudo apt-key adv --keyserver hkp://pgp.mit.edu:80 --recv-keys
58118e89f3a912897c070adbf76221572c52609d

# open the /etc/apt/sources.list.d/docker.list file in your favorite
editor. if the file doesn't exist, create it. and add:

  # this is not a command, but something that needs to be pasted in the
  file opened

  deb https://apt.dockerproject.org/repo ubuntu-trusty main

sudo apt-get update

# verify that apt is pulling from the right repository.

sudo apt-cache policy docker-engine

sudo apt-get install linux-image-generic-lts-trusty

# you need to reboot, this can cause some issues where the firewall
settings would not be saved, this was solved by loading the firewall
explicitly at the restart

sudo reboot

sudo apt-get update

sudo apt-get install docker-engine

sudo docker run hello-world
```

2.1.2 Step 2: Configure Docker

```
# set UFW's forwarding policy appropriately.
# Open /etc/default/ufw file for editing.
sudo vim /etc/default/ufw

# Set the DEFAULT\_FORWARD\_POLICY policy to:
DEFAULT\_FORWARD\_POLICY="ACCEPT"

sudo ufw reload

#TODO check if this is persistent

# Allow incoming connections on the Docker port.
$ sudo ufw allow 2375/tcp

# Configure Docker to start on boot
# DOCS SAY $ sudo systemctl enable docker
# but I think:

sudo update-rc.d docker defaults
```

2.1.3 Excursus: Quick Docker command guide

You might have to prepend `sudo` to the commands below, depending on your environment.

```
# is docker installed
docker info

# activate docker
docker-machine start default
eval "$(docker-machine env default)"

# to find localhost ping
docker-machine env default

# to build
cd ~/ImagesDocker/clic-docker/
docker build -t jdejoode/clic:v0 .

# list images
docker images

# run a docker image
docker run -d -P -i -t --name apache11 jdejoode/clic:v0 docker ps

# find info on container
docker port apache11
docker logs a5a665d32

# for live updates a la Flask
docker logs -f a6516a51sd f651

# stop all docker containers
docker stop $(docker ps -a -q)

# remove them
```

```
docker rm $(docker ps -a -\ **q**\ )

# ssh into container
docker exec -i -t fbd8112 bash

# command on actual deploy is slightly different for caching and
rounting purposes:
# -v /path/on/host:/path/in/container
docker run -p 80:8080 -v /tmp/cache:/tmp/cache ...

docker run -d -P --name clic2 -v
/bin:/clic-project/clic/dbs/dickens/indexes jdejoode/clic:v0

# to see what processes run in the container
docker top clic0

# remove untagged images
docker rmi $(docker images \| grep "^<none>" \| awk '{print $3}')

# deploy
# on macbook
# https://docs.docker.com/docker-hub/repos/
docker login
docker push jdejoode/clic:latest


# You can also bind Docker containers to specific ports using the-p flag,
for example:
$ docker run -d -p 80:5000 training/webapp python app.py
```

2.1.4 Step 3: Get CLiC's Docker image

```
# you can upload the Docker image found on the shared drive
`mahlbema-01` in the folder `CLiC Live Server Data and Image`

Do load the image on the server, run:

docker load -i path-to-uploaded-CliCLive.tar

For instance:

sudo docker load -i CLiCLive.tar
```

2.1.5 Step 4: Get the indexes, stores, and code

The indexes can be downloaded from the same shared drive (*mahlbema-01*). They need to be uploaded to the server as they need to be included in the Docker container as volumes when initialising the Docker container.

```
git clone https://github.com/CentreForCorpusResearch/clic
```

2.1.6 Step 5: Run the Docker container

The `path-to` elements in the following snippets need to be replaced with the actual path to your indexes, stores, configs, and code.

```
sudo docker run -d -p 80:8080 -p 5000:5000
-v /home/clicman/clic:/clic-project/clic
-v /home/clicman/indexes:/clic-project/clic/dbs/dickens/indexes
-v /home/clicman/stores:/clic-project/clic/dbs/dickens/stores
-v /home/clicman/db_annotation_2016_05_10_at_12_00.tar:/clic-project/clic/db_annotation.tar
-v /home/clicman/config.xml:/clic-project/clic/dbs/dickens/config.xml
-v /home/clicman/textfiles:/clic-project/clic/clic/textfiles
--name clic jdejoode/clic:latest
```

What the above command does:

- Run the CLiC Docker container called with the latest tag as the exact version
- The –d is used to run docker as a daemon (to keep it running, otherwise it only runs a single command)
- -p 80:8080 tells the host to forward port 80 to 8080
- -v host:docker mounts two different folders (the indexes and the stores) which are essential to clic. These are not included in the Docker image as they are volatile and as they are too big.

This enables you to update and release new code that does not change the database as the code and the configs are mounted.

Get the database up and running:

```
# move into the container
docker exec -it clic bash

# run the following commands
dropdb db_annotation
dropuser clic-dickens
sudo -u postgres createuser -P -d -r -s clic-dickens
createdb -O clic-dickens db_annotation --password
# db_annotation.tar is the db.tar that was mounted earlier
pg_restore --dbname=db_annotation --verbose /clic-project/clic/db_annotation.tar

# update to the latest, more advanced caching framework
pip install --upgrade Beaker pandas

# restart uwsgi and postgres
supervisorctl restart all
```

This should get CLiC up and running on your server/computer. Make sure to check whether the forms actually work before considering the installation a success.

There are functional tests in clic/tests/functional/main.py. Read that document for more information.

Before destroying a container, on has to export the postgres database. For instance:

```
pg_dump -U clic-dickens -h localhost -W -F t db_annotation > /clic-project/clic/db_annotation.tar
```

2.1.7 Enjoy, unless ...

If the server knows a power outage, Docker will be shut down without proper warning. This means that the Docker container has to be restarted on rebooting the server.

To do so:

```
# Is the container running?
sudo docker ps
```

```
# If not: check if the container is still available
sudo docker ps -a

# You can now restart that image, for instance, clic18 by running
sudo docker start clic18

# The power outage could have caused other settings to be forgotten:
# IF NEEDED, the following commands can be used to bring them back up

sudo ufw reload
sudo ufw restart
sudo service docker restart

# If the issue still is not solved, consider the above documentation on the installation
# of Docker on a vanilla server
```

To troubleshoot the container:

```
# Is the container running?
sudo docker ps

# Are the right processes running on the container?
sudo docker top clic  # (where clic is the name of the container)

# Did the volumes mount correctly?
Exec into the clic container to check manually whether the volumes are mounted

# What do the logs say?
sudo docker logs afcb70

# What services are listening on what ports?
sudo netstat -peanut

# Do you get "ACCEPT: iptables: No chain/target/match by that name"
sudo service docker restart

# In very rare cases, a container might go down without prior notice. In that case
# check whether it is up:
sudo docker ps
# if it is not up restart it
sudo docker start clic (where clic is the container name)
```

2.2 Installing CLiC on your own computer

Because of the CLiC is released as a Docker image, you can also install CLiC on your own computer (Mac, Windows, or Linux) by simply installing Docker and following the system-specific install instructions in the Docker docs.

2.3 Run the tests

To run the tests:

```
BASE_URL='http://live/' py.test main.py  (in
clic/tests/functional tests/)
```

CHAPTER

THREE

CLIC FOR DEVELOPERS

3.1 Cheshire3

3.1.1 The data-model

3.2 CLiC Concordance

CLiC Concordance based on cheshire3 indexes.

class concordance.**Concordance**

This concordance takes terms, index names, book selections, and search type as input values and returns json with the search term, ten words to the left and ten to the right, and location information.

This can be used in an ajax api.

build_and_run_query(*terms*, *idxName*, *Materials*, *selectWords*)

Builds a cheshire query and runs it.

Its output is a tuple of which the first element is a resultset and the second element is number of search terms in the query.

create_concordance(*terms*, *idxName*, *Materials*, *selectWords*)

main concordance method create a list of lists containing each three contexts left - node -right, and a list within those contexts containing each word. Add two separate lists containing metadata information: [[left context - word 1, word 2, etc.], [node - word 1, word 2, etc], [right context - word 1, etc], [chapter metadata], [book metadata]], etc.

3.3 CLiC Clusters

Tool to create wordlists based on the entries in an index.

class clusters.**Clusters**

Class that does all the heavy weighting. It makes the connection with cheshire3, uses the input parameters (indexname and subcorpus/Materials) to return a list of words and their total number of occurrences.

For instance,

```
the 98021

to 78465

...
```

or `he said 8937`

`she said 6732`

...

`list_clusters`(*idxName*, *Materials*)
Build a list of clusters and their occurrences.

Limit the list to the first 3000 entries.

3.4 CLiC Keywords

Module to compute keywords (words that are used significantly more frequently in one corpus than they are in a reference corpus).

The statistical measure used is Log Likelihood as explained by Rayson and Garside:

Rayson, P. and Garside, R. (2000). Comparing corpora using frequency profiling. In proceedings of the workshop on Comparing Corpora, held in conjunction with the 38th annual meeting of the Association for Computational Linguistics (ACL 2000). 1-8 October 2000, Hong Kong, pp. 1 - 6 Available at: http://ucrel.lancs.ac.uk/people/paul/publications/rg_acl2000.pdf

3.4.1 Usage as follows:

analysis = pd.DataFrame([('a', 2500), ('the', 25000)], columns=('Type', 'Count')) reference = pd.DataFrame([('a', 10), ('the', 10)], columns=('Type', 'Count'))

result = extract_keywords(analysis, reference, tokencount_analysis=20, tokencount_reference=100, round_values=True, limit_rows=3)

result

keywords_to_json(result)

3.4.2 Or using text files as input:

from collections import Counter

with open('~/data/input/DNov/BH.txt') as inputfile: bh = inputfile.read().split() bh = Counter(bh)

with open('~/data/input/DNov/OT.txt') as inputfile: ot = inputfile.read().split() ot = Counter(ot)

bh_df = pd.DataFrame(bh.items(), columns=['Type', 'Count']) ot_df = pd.DataFrame(ot.items(), columns=['Type', 'Count'])

extract_keywords(bh_df, ot_df, tokencount_analysis=bh_df.Count.sum(), tokencount_reference=ot_df.Count.sum(), round_values=True,)

`keywords.`**`extract_keywords`**(*wordlist_analysis*, *wordlist_reference*, *tokencount_analysis*, *tokencount_reference*, *p_value=0.0001*, *exclude_underused=True*, *freq_cut_off=5*, *round_values=True*, *limit_rows=False*)
This is the core method for keyword extraction. It provides a number of handles to select sections of the data and/or adapt the input for the formula.

Input = Two dataframes with columns 'Type', and 'Count' and two total tokencounts

Output = An aligned dataframe which is sorted on the LL value and maybe filter

using the following handles:

- **p_value: limits the keywords based on their converted p_value. A p_value of 0.0001** will select keywords that have 0.0001 *or less* as their p_value. It is a cut-off. One can choose one out of four values: 0.0001, 0.001, 0.01, or 0.05. If any other value is chosen, it is ignored and no filtering on p_value is done.
- **exclude_underused: if True (default) it filters the result by excluding tokens that are** statistically underused.
- **freq_cut_off: limits the wordlist_analysis to words that have the freq_cut_off (inclusive)** as minimal frequency. 5 is a sane default for Log Likelihood. If one does not want frequency-based filtering, set a value of 0.
- **round_values: if True (default) it rounds the columns with expected frequencies and LL** to 2 decimals.
- **limit_rows: if a number (for instance, 100), it limits the result to the number of rows** specified. If false, rows are not limited.

The defaults are reasonably sane:

- a token needs to occur at least 5 times in the corpus of analysis
- a high p-value is set
- no filtering of the rows takes place
- the underused tokens are excluded
- rounding is active

For more information on the algorithm, cf. log_likelihood().

The first column contains the indeces for the original merged dataframe. It does not display a rank and it should be ignored for keyword analysis (to be more precise: it displays the frequency rank of the token in the corpus of analysis).

keywords.**format_wordlist**(*types*, *counts*, *total=None*)

Helper function to convert lists with information into a dataframe.

keywords.**keywords_to_json**(*keywords*)

Transforms a keywords table into a json array. It is specifically tailored at the CLiC Dickens interface.

keywords.**log_likelihood**(*counts*)

This function uses vector calculations to compute LL values.

Input: dataframe that is formatted as follows:

Type, Count_analysis, Total_analysis, Count_ref, Total_ref

Output: dataframe that is formatted as follows:

Type, Count_analysis, Total_analysis, Count_ref, Total_ref, Expected_count_analysis, Expected_count_ref LL

Hapax legomena in the Count_analysis are not deleted. It is expected that Count_analysis and Expected_count_analysis are not zero.

3.5 CLiC Chapter Repository

Display the texts available in the cheshire3 database. Also highlight specific items that were previously retrieved with a concordance.

class chapter_repository.**ChapterRepository**
Responsible for providing access to chapter resources within Cheshire.

get_book_title(*book*)
Gets the title of a book from the json file booklist.json

book – string - the book id/accronym e.g. BH

get_chapter(*chapter_number*, *book*)
Returns transformed XML for given chapter & book

chapter_number – integer book – string - the book id/accronym e.g. BH

get_chapter_with_highlighted_search_term(*chapter_number*, *book*, *wid*, *search_term*)
Returns transformed XML for given chapter & book with the search highlighted.

We create the transformer directly so that we can pass extra parameters to it at runtime. In this case the search term.

chapter_number – integer book – string - the book id/accronym e.g. BH wid – integer - word index search_term – string - term to highlight

get_raw_chapter(*chapter_number*, *book*)
Returns raw chapter XML for given chapter & book

chapter_number – integer book – string - the book id/accronym e.g. BH

3.6 CLiC KWICgrouper

A module to look for patterns in concordances.

class kwicgrouper.**Concordance**(*term*, *text*, *word_boundaries=True*, *length=50*, *keep_punctuation=True*, *keep_line_breaks=False*)
This is a simple concordance for a text file. The input text should a string that is cleaned, for instance:

text.replace(“

”, ” ”).replace(” ”, ” ”)

This function has two argument: the search term and the text to be searched.

The length should be an integer

classmethod from_multiple_line_file(*term*, *input_li*)
Construct a concordance that respect line breaks (rather than one that treats the text as one large string)

TODO

list_concordance()
List the actual concordance.

print_concordance()
Print the lines of a concordance. For debugging purposes.

single_line_conc()
Build a basic concordance based on a single string of text.

class kwicgrouper.**KWICgrouper**(*concordance*)

This starts from a concordance and transforms it into a pandas dataframe (here called textframe) that has five words to the left and right of the search term in separate columns. These columns can then be searched for and sorted.

Input: A nested list of lists looking like:

[['sessed of that very useful appendage a ', 'voice', ' for a much longer space of time than t'],

...

Each pattern needs *its own* instantiation of the KWICgrouper object because the self.textframe variable is changed in the filter method.

args_to_dict(*L5=None*, *L4=None*, *L3=None*, *L2=None*, *L1=None*, *R1=None*, *R2=None*, *R3=None*, *R4=None*, *R5=None*)

Helper function to use L1="a" type of functions

conc_to_df()

Turns a list of dictionaries with L1-R5 values into a dataframe which can be used as a kwicgrouper.

filter_textframe(*kwdict*)

Construct a dataframe slice and selector on the fly. This is no longer meta-programming as it does not use the eval function anymore.

This returns None if there is no textframe

split_nodes()

Splits the words into nodes that can be fed into a dataframe.

kwicgrouper.**clean_punkt**(*text*)

Delete punktuation from a text.

Problem: turns CAN'T into CA NT

kwicgrouper.**clean_text**(*text*)

Clean a text so that it can be used in a concordance. This includes:

- all text to lowercase
- deleting line-breaks
- tokenizing and detokenizing

kwicgrouper.**concordance_for_line_by_line_file**(*input_file*, *term*)

Takes a file that has different line breaks that cannot be ignored (for instance a file with a list of things) and makes it into a concordance

kwicgrouper.**old_clean_punkt**(*text*)

This ignores apostrophes and punctuation marks attached to the word * an alternative way would be to replace-delete the punctuation from the text

3.7 CLiC Normalizer

Defines normalizers that can be used in the cheshire3 indexing workflow.

3.8 CLiC Query Builder

Future module to handle the construction of cheshire3 CQL queries.

3.9 CLiC Web app

3.9.1 Index

This is the most important file for the web app. It contains the various routes that end users can use.

For instance

@app.route('/about/', methods=['GET']) def about():

    return render_template("info/about.html")

Where /about/ is the link.

3.9.2 API

This file is an extension of index.py. It generates the raw json API that the keywords, cluster, and concordances use(d).

It needs to be refactored.

3.9.3 Models

models.py defines the SQL tables that CLiC uses. These classes provide a python interface to the SQL database so that you can write python code that automatically queries the database.

This is heavily dependent on Flask-SQLAlchmey and SQLAlchemy

3.10 How to run the local development server?::

    ~/projects/clic/clic/clic/web$ PYTHONPATH=~/projects/clic/clic python manage.py runserver

3.11 How to run new indexes?::

    ~/projects/clic/clic/dbs/dickens/indexes$ CLIC_SETTINGS=local PYTHONPATH=~/projects/clic/clic python run.py –ntc ~/projects/clic/clic/dbs/dickens/indexes$ CLIC_SETTINGS=local PYTHONPATH=~/projects/clic/clic python run.py –dickens

CHAPTER

FOUR

REPORTING ISSUES

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

c

k

n

q

w