Introduction to the Reproducible Open Coding Kit (ROCK)

Gjalt-Jorn Ygram Peters & Szilvia Zörgő

2021-07-22

The Reproducible Open Coding Kit (ROCK) was developed to facilitate reproducible and open coding, specifically geared towards qualitative research methods. Although it is a general-purpose toolkit, three specific applications have been implemented:

- an interface to the `rENA` package that implements Epistemic Network Analysis (ENA);
- means to process notes from Cognitive Interviews (CIs);
- means to work with a decentralized construct taxonomy (DCT).

In this introduction, first a general overview of the logic behind ROCK will be given, after which each of those three use cases will be discussed.

General background and introduction

Although the availability of tools for reproducible and open quantitative research is quickly increasing, there are few such options for researchers using qualitative methods. This hampers applying Open Science principles, and as a result, it often precludes re-use of data, learning from each other, and detecting errors. The Reproducible Open Coding Kit (ROCK) aims to fill this gap.

ROCK consists of two parts. On the one hand, there is the `rock` R package that makes it easy to work with `.rock` files. This makes it easy to, for example, process coded sources and collapse the most specific levels of hierarchical coding trees into their parents, specify which utterances form stanzas or strophes for Epistemic Network Analysis, or

On the other hand, there is the `.rock` file format: plain text files that can include YAML fragments and follow conventions that allow extracting metadata, deductive and inductive coding trees, and codes as applied to lines of each source. Because these are plain text files, they are easily accessible to other researchers, regardless of the software they use.

`.rock` file format

The `.rock` file format uses a number of conventions to represent codes, metadata, and data structures. Although the functions in the `rock` package have been built to allow flexibility, this vignette uses the defaults (there is something to say for uniformity, after all).

Codes

Codes are by default any string of characters (specifically, lower or uppercase letters, digits, periods, underscores, larger than signs, and dashes) in between two pairs of square brackets (`[[` and `]]`). This is described

by the regular expression `\[\[([a-zA-Z0-9._>-]+)\]\]`; note that the escaping backslashes must be escaped themselves by prepending a second backslash when specifying this regular expression in R. Codes are designated per utterance, or in other words, per line. As many codes can be specified per line as one wishes. For example, see these two lines (utterances):

```
So what went right [[reflection-positive]]
What went wrong [[reflection-negative]]
```

The first line is coded with `reflection-positive`, and the second line with code `reflection-negative`.

Structuring inductive codes

When engaging in inductive coding (i.e. when not working with a prespecified code structure, but instead developing the code structure as one goes along; see the section below re: deductive coding), it can be desirable to structure the codes hierarchically. For example, perhaps a researchers wants to specify a parent code such as `reflection` with two child codes such as `positive` and `negative`. This helps one to identify patterns in the data, and makes it possible to easily extract all utterances coded as any type of reflection. By default, the marker that can be used to structure inductive codes is the greater than sign (specified by the regular expression `>`). For example, see the same fragment but coded in two levels:

```
So what went right [[reflection>positive]]
What went wrong [[reflection>negative]]
```

When this source is parsed by `rock`, it will recognize these deductive codes and their structure, and it will generate the corresponding hierarchical coding structure, as illustrated in the more extensive example below.

Specifying identifiers

It is often desirable to attach specific attributes to utterances. For example, one may want to compare the patterns in codes between different categories of participants, such as those who do and do not own a car, or those that listen to progressive metal versus those that listen to psychedelic trance. Instead of coding all utterances with all relevant attributes, instead, it is possible to specify identifier to easily link utterances to characteristics of the data provision (such as data providers, for example participants, or the moment of data collection, for example daytime or nighttime, or winter or summer, or the location of data collection, such as in a busy place or in a silent office).

This can be done by specifying identifiers. These are again specified using regular expressions. By default, two types of identifiers are specified: case identifiers and stanza identifiers. They are again specified using two pairs of square brackets, but this time, the opening brackets are immediately follow by a string of identifying characters (the 'identifier identifier', so to speak), followed by an equals sign, and then by the unique identifier. This may seem a bit abstract; it will become clearer as we look at the first example.

Case identifiers

Case identifiers can be used to link utterances to data providers, such as participants. Their 'identifier identifier' is `cid`, and by default, their full regular expression is `\[\[cid=([a-zA-Z0-9._-]+)\]\]`. A source excerpt coded with only case identifiers may look like this:

```
CAIAPHAS: No, wait!   We need a more permanent solution to our problem. [[cid=1]]

ANNAS: What then to do about Jesus of Nazareth?   Miracle wonderman, hero of fools. [[cid=2]]
```

```
PRIEST THREE: No riots, no army, no fighting, no slogans. [[cid=3]]

CAIAPHAS: One thing I'll say for him -- Jesus is cool. [[cid=1]]

ANNAS: We dare not leave him to his own devices.   His half-witted fans will get out of control. [[cid=
```

(Note that in this example, the names of the participants were retained; normally, the researcher would anonimyze the transcripts so as to allow publication of the coded transcripts.)

WHen `rock` parses this source, it will know that the first and fourth utterances belong to the same case, as do the second and fifth. The attributes specified for these cases will then be attached to these utterances (see the section about metadata below).

Stanza identifiers

A stanza is a unit of analysis in ENA analysis (see the glossary for the exact definition).

Specifying deductive coding structures

When a researcher works with a prespecified coding structure (i.e. engages in deductive coding), they only use codes that were determined a priori. Like in inductive coding, there are often multiple levels in such a coding structure, with the codes organised hierarchically. To efficiently be able to collapse codes to higher levels, `rock` needs to know the deductive coding structure. This can be specified using YAML fragments in the sources. YAML fragments are, by default, delimited by two lines that each contain only three dashes (`---`). Between those delimiters, YAML (a recursive acronym that stands for 'YAML ain't markup language') can be specified. Specifically, in YAML terminology, each fragment should be a sequence of mappings that is named `codes`.

The coding tree specified in the section on inductive coding, for example, can be efficiently specified as a deductive coding structure like this:

```
---
codes:
  -
    id: reflection
    children:
      -
        id: positive
      -
        id: negative
---
```

If all children of a code are so-called 'leaves' (i.e. in the coding tree, they have no children of their own[1]) they can be specified more efficiently:

```
---
codes:
  -
    id: reflection
    children: ["positive", "negative"]
---
```

[1] This is less sad than it may look; voluntary childlessness is becoming more and more common, and not having children is one of the most effective choices one can make to save the environment.

When `rock` parses the sources, it will collect all such code specifications and combined them into one coding three using each code's identifiers. It is possible to specify a parent in other code specification fragment by adding the field `parentId`. For example, in other sourrce, we could add this fragment:

```
---
codes:
  -
    id: neutral
    parentId: reflection
---
```

This would add `neutral` as a sibling to `positive` and `negative`.

Specifying metadata

Examples

Section breaks

```
So what went right
What went wrong
---paragraph-break---
Was it a story
or was it a song
---paragraph-break---
Was it over night
Or did it take you long
---paragraph-break---
Was knowing your weakness
what made you strong
```

Source excerpt as example of section breaks (lyrics from Smiley Faces by Gnarls Barclay)

Identifiers

```
CAIAPHAS
No, wait!   We need a more permanent solution to our problem.

ANNAS
What then to do about Jesus of Nazareth?   Miracle wonderman, hero of fools.

PRIEST THREE
No riots, no army, no fighting, no slogans.

CAIAPHAS
One thing I'll say for him -- Jesus is cool.

ANNAS
We dare not leave him to his own devices.   His half-witted fans will get out of control.

PRIESTS
But how can we stop him?   His glamour increases   By leaps every moment; he's top of the poll.
```

```
CAIAPHAS
I see bad things arising.   The crowd crown him king; which the Romans would ban.
I see blood and destruction,   Our elimination because of one man.   Blood and destruction because of o

ALL (inside)
Because, because, because of one man.

CAIAPHAS
Our elimination because of one man.

ALL (inside)
Because, because, because of one, 'cause of one, 'cause of one man.

PRIEST THREE
What then to do about this Jesus-mania?

ANNAS
How do we deal with a carpenter king?

PRIESTS
Where do we start with a man who is bigger   Than John was when John did his baptism thing?

CAIAPHAS
Fools, you have no perception!   The stakes we are gambling are frighteningly high!
We must crush him completely,   So like John before him, this Jesus must die.   For the sake of the nat
```

This Jesus Must Die by Andrew Lloyd Webber

Codes

Helper functions

`clean_source` and `clean_sources`

Sometimes, sources are a bit messy.[2] In such cases, it can be efficient to preprocess them and perform some search and replace actions. This can be done for one or multiple source files using `clean_source` (for one file) and `clean_sources` (for multiple files; it basically just calls `clean_transcript` for multiple files).

For example, a researcher will often want every sentence, as transcribed, to be on its own line (as lines correspond to utterances). In fact, this is the basic function of the `clean_source` function: by default, if used without other arguments, they try to (more or less smartly) split a transcript such that each transcribed sentence (as marked by a period (`.`), a question mark (`?`), an exclamation mark (`!`), or an ellipsis (`...`)) ends up on its own line. Before doing this, `clean_source` replaces all occurrences of exactly consecutive periods (`..`) with one period, all occurrences of four or more consecutive periods with three periods, and all occurrences of three or more newlines (`\n`) with two newlines.

But this function can also be used to perform additonal (or other) replacements. For example, imagine that a transcriber used a dash at the beginning of a line, followed by a space, to indicate when a person starts talking. To easily group all utterances by the same person together, it would be convenient if this was expressed in the source file in a way that fits with ROCK's conventions. There are four ways to achieve this.

First, that sequence of characters (actually a newline character (`\n`) followed by a dash (`-`) followed by a whitespace character (`\s`)) can be converted into section break '`---turn-of-talk---`' like this:

[2] Well, they are messy more often than not, unfortunately.

```
rock::clean_source("
- Something said by one speaker
- Something said by another speaker
",
                   replacements=list(c("\\n-\\s", "\n---turn-of-talk---\n")));
```

To also maintain the default replacements, more can be added by specifying them in argument `extraReplacements` instead of `replacements`. For `clean_source`, as the first argument (`input`), either a character vector (like in the example above) or a path to a file can be specified, in which case the files contents will be read. If the second argument (`outputFile`) is specified, the result is saved to that file; if not, it is returned (and printed by R).

A word of caution

If you use this function to clean one or more transcripts, make sure that whenever you edit the `outputFile`, you save it under another name! Otherwise, rerunning the script to clean the transcripts will overwrite your edits.

Glossary

Attribute ***(ROCK term)*** An attribute is a characteristic of a case. FOr example, if the cases are persons, attributes can be characteristics such as sex, age, region of residence, or diagnosis. All attributes of a case are attached to all utterances coded as belonging to that case.

Case ***(ROCK term)*** A case is a data provider, such as a person, a family, an organisation, or any other research unit. In psychological research, cases are almost always persons, but in a qualitative study of organisation policy documents, each organisation would be a case.

Conversation ***(ENA term)*** In ENA terminology, a source is called a conversation (see *source*).

Discourse ***(ENA term)*** In ENA terminology, a discourse is the full set of all utterances (and therefore, all sources) under analysis.

Identifier ***(ROCK term)*** In ROCK, an identifier provides a way to specify metadata (i.e. one or more attributes) about utterances. The most common identifiers are cases, but in ROCK, this concept is generalized, and different types of identifiers can be specified. This makes it possible to quickly attach other attributes to utterances as well. For example, one could use case identifiers to attach personal characteristics such as sex or age to utterances, and interview location identifiers to attach information about the interview context to utterances.

Metadata ***(ENA term)*** Metadata refers to the attributes that can be specified about cases (or more generally, can be attached to identifiers), and so can be attached to utterances. This metadata then makes it possible to group or compare patterns in codes.

Section ***(ROCK term)*** A section is a set of one or more consecutive utterances. Section markers can be used to divide a source into sections. Multiple different section markers can be used, and so a source can simultaneously be divided into sections differently.

Section marker ***(ROCK term)*** A section marker is a sequence of characters that marks a section division. Every section marker encountered in a source signals the ending of the preceding and the beginning of the following section. In the produced dataframe with utterances, this is shown by increasing section numbers.

Source ***(ROCK term)*** A file with content to code. The file can contain, for example, a transcript of an interview or focus group, or it can be something else, like a policy document, a discussion in an online forum, or a list of twitter posts. A source consists of utterances.

Stanza ***(ENA term)*** a set of one or more utterances that occur in close proximity and that are defined by ... (the complicated bit), i.e. the smallest unit of analysis psychologically. Stanzas are a solution for the fact that peoples' sentence length is not closely related to the psychological unity of the matter they discuss. E.g. Alice could discuss her ideas about the cause of the disease in three sentences while Bob would use only one, but both explanations would be the same stanza. Stanzas are what in

written text would be paragraphs. Except that spoken text is more messy, and so 'paragraphs' can be interspersed by unrelated utterances. Therefore, utterances in close proximity can be combined into the same stanza even if they are separated by a small number of unrelated utterances. Note, however, that stanzas remain defined as 'messy paragraphs'; if an interviewee starts referring back to something discussed half an hour earlier, that doesn't justify combining those utterances into the same stanza. (Note, however, that they *could* be combined through stanza sets / strophes, if they happen to be codes by the same codes, *and* if strophes are composed by collapsing stanzas with those codes.)

Strophe ***(ENA term)*** a set of one or more stanzas that are combined based on sharing the same code, identifier, or attribute. Strophes, therefore, are collected over the entire transcript/source or over all transcripts/sources; proximity is irrelevant.

Topic ***(ROCK term)*** A topic is a distinct subject area covered by data in a source. For example, for interviews or focus groups, the topics can be the topics that form the toic list or the interview scheme.

Transcript ***(ROCK term)*** The verbatim text of an interview or focus group in a text file.

Unit ***(ENA term)*** a set of one or more utterances that share a given attribute

Utterance ***(ENA & ROCK term)*** An utterance is the smallest unit of analysis in a study. This is usually one sentence, but in any case it is one line in the `rock` file (a line being defined as zero or more characters ending with a line ending). That is, when reading the sources, `rock` splits each source at the line endings (newline characters).

Using ROCK for Epistemic Network Analysis

In Epistemic Network Analysis (ENA), the data are segmented and co-occurrences of codes in segments called stanzas are determined, after which these co-occurrences are visualised in a network. In ENA, the smallest unit of analysis is that stanza, but stanzas are composed of one or more utterances. An utterance can be, for example, a sentence, but it can also be several sentences or parts of sentences. Stanzas can be entire interviews, or paragraphs within an interview, or sets of two or three utterances. Each of these are determined depending on what is sensible given the type of data and research question at hand. In ENA vocabulary, a conversation defines a set of utterances that can be segmented into stanzas. The role of these conversations is to explicitly specify where co-occurrences can exist (i.e. only within the same conversation). When specifying stanzas 'manually', this is of limited added value, but when specifying stanzas automatically, for example using what is called the moving stanza window in ENA vocabulary, such delineations are used to constrain the possible stanzas.

In addition to this segmentation in stanzas, the data are segmented into units. A unit is defined by a nesting of characteristics. In research in humans, one such characteristic that would make sense to distinguish is a person: designating persons as one level in the unit specification ensures that the dependence between utterances of the same person is taken into account properly. Similarly, subsamples of interest can be defined, such as categories of a categorical variable that has been collected as metadata (e.g. country of residence of participants, or sexual preference, or age group). It is also possible to define unit specification levels that are *smaller* than persons, for example by specifying the different questions in an interview scheme or topic list as a level in the unit specification.

In the ROCK, the identifiers, such as case identifiers and section identifiers, can be used, as well as metadata or even codes.

It is important to have sufficient units compared to the number of codes. This equation expresses how many units you need as a function of the number of codes:

$$\text{units} \geq \binom{\text{codes}}{2} + 1$$

Discourse - big D discourse = the norms, consistency in communications;

quantitative ethnography

discourse = ‘population’

sample has observations - inferences towards discourse

unit = anything you want to see a network for. Can be entire sample; can also be subsample.

conversation = interview; metadata may nest within conversations

conversations determine the boundaries where the conversations (used to be called strophe)

co-occurrences can only occur within conversations

interpretation of the ENA space requires n+2 codes

n choose 2

combinations

so 10 choose 2 means you need 45 codes +1 = 46

units are the combination

rigid body rotation - always the same

units are a sequence of nestings where one level can be the question, one level can be the individuals, one level can be an attribute (categorisation) of the individuals

n choose 2 + 1

Using ROCK for Cognitive Interviews

This is based on the Respondent Problem Matrix (see e.g. Conrad & Blair, 1996; http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.451.3389&rep=rep1&type=pdf).

Using ROCK with a Decentralized Construct Taxonomy