Handout 11: cleanNLP

Taylor Arnold

The **cleanNLP** package is designed to make it as painless as possible to turn raw text into feature-rich data frames. At the moment, it is best to grab the development version using devtools:

```
devtools::install_github("statsmaths/cleanNLP")
```

As described in detail below, the package has a bare-bones parser that does not require any additional system dependencies. However, to benefit from the package most users will want to load either a Python or Java backend.

Basic usage

The **cleanNLP** package is designed to make it as painless as possible to turn raw text into feature-rich data frames. Take for example the opening lines of Douglas Adam's *Life, the Universe and Everything*:

```
text <- c("The regular early morning yell of horror was the sound of",
          "Arthur Dent waking up and suddenly remembering where he",
          "was. It wasn't just that the cave was cold, it wasn't just",
          "that it was damp and smelly. It was the fact that the cave",
          "was in the middle of Islington and there wasn't a bus due",
          "for two million years.")
text <- paste(text, collapse = " ")
```

A minimal working example of using **cleanNLP** consists of loading the package, setting up the NLP backend, initalizing the backend, and running the function `run_annotators`. Because our input is a text string we set `as_strings` to `TRUE` (the default is to assume that we are giving the function paths to where the input data sits on the local machine"):

```
library(cleanNLP)
init_spaCy()
obj <- run_annotators(text, as_strings = TRUE)
```

Here, we used the spaCy backend. A discussion of the various backend that are available are given the following section. The returned annotation object is nothing more than a list of data frames (and one matrix), similar to a set of tables within a database. The names of these tables are:

```
names(obj)
```

```
## [1] "coreference" "dependency"  "document"
## [4] "entity"      "sentence"    "token"
## [7] "vector"
```

```
## [1] "coreference" "dependency"  "document"    "entity"      "sentence"
## [6] "token"       "vector"
```

The canonical way of accessing these data frames is by using functions of the form `get_TABLE`. For example, the document table gives metadata about each document in the **corpus**, which here consists of only a single document:

```
get_document(obj)
```

```
## # A tibble: 1 × 5
##      id                time version language
##   <int>              <dttm>   <chr>    <chr>
## 1     1 2017-05-23 22:06:34   1.8.2     <NA>
## # ... with 1 more variables: uri <chr>
```

```
# A tibble: 1 × 5
     id                time version language
  <int>              <dttm>   <chr>    <chr>
1     1 2017-05-20 15:24:44   1.8.2     <NA>
# ... with 1 more variables: uri <chr>
```

The tokens table has one row for each word in the input text, giving data about each word such as its lemmatized form and its part of speech. We access these table with the `get_token` function:

```
get_token(obj)
```

```
## # A tibble: 68 × 8
##       id   sid   tid    word   lemma  upos
##    <int> <int> <int>   <chr>   <chr> <chr>
## 1      1     1     1     The     the   DET
## 2      1     1     2 regular regular   ADJ
## 3      1     1     3   early   early   ADJ
## 4      1     1     4 morning morning  NOUN
## 5      1     1     5    yell    yell  NOUN
## 6      1     1     6      of      of   ADP
## 7      1     1     7  horror  horror  NOUN
## 8      1     1     8     was      be  VERB
## 9      1     1     9     the     the   DET
## 10     1     1    10   sound   sound  NOUN
## # ... with 58 more rows, and 2 more
## #   variables: pos <chr>, cid <int>
```

```
# A tibble: 68 × 8
      id   sid   tid    word   lemma  upos   pos   cid
   <int> <int> <int>   <chr>   <chr> <chr> <chr> <int>
1      1     1     1     The     the   DET    DT     0
2      1     1     2 regular regular   ADJ    JJ     4
3      1     1     3   early   early   ADJ    JJ    12
4      1     1     4 morning morning  NOUN    NN    18
5      1     1     5    yell    yell  NOUN    NN    26
6      1     1     6      of      of   ADP    IN    31
7      1     1     7  horror  horror  NOUN    NN    34
8      1     1     8     was      be  VERB   VBD    41
9      1     1     9     the     the   DET    DT    45
10     1     1    10   sound   sound  NOUN    NN    49
# ... with 58 more rows
```

The output from the `get` functions are (mostly) pre-calculated. All of the hard work is done in the `run_annotators` function.

Backends

There are three "backends" for parsing text in **cleanNLP**. These are:

- an R-based version using only the **tokenizers** package. It offers minimal output but requires no external dependencies.
- a Python-based parser using the spaCy library. Requires installing Python and the library separately; this is generally pain free and works with any version of Python >= 2.7. The library is very fast and provides the basic annotators for parsing text such as lemmatization, part of speech tagging, dependency parsing, and named entity recognition. There is also support for computing word embeddings in English.
- a Java-based parser using the CoreNLP library. Setting this up is often more involved and the data files are quite large. The pipeline also takes significantly longer than the spaCy implementation. However, the CoreNLP parser offers many more bleeding-edge annotation tasks such as coreference resolution and speaker detection.

The only benefit of the R-based version is its lack of external dependencies. We supply it mainly for testing and teaching purposes when access to the machine(s) running the code are not under our control. For most use-cases we recommend using the spaCy library as it strikes a balance between features, speed, and ease of set-up. The CoreNLP backend should be used when access to the advanced annotation tasks is required or when parsing in languages not yet available in spaCy.

spaCy backend (python)

To use the Python-based backend spaCy, we first need to install Python and spaCy on our system. The package **reticulate** must also be installed; see the spaCy website for download instructions on your platform. After this is done, we can run the code snippet given in the prior section

coreNLP backend (Java)

In order to make use of the Java-based coreNLP backend, a version of Java >= 7.0 must be installed and the **rJava** package must be set up. This should be straightforward, but this has caused problems on some machines, particularly with macOS. For help, see the GitHub issues tracker. Once these system requirements are met, we can install the ".jar" files with

```
download_core_nlp()
```

These files are large and may take several minutes to download. The Java backend is then configured with `setup_coreNLP_backend`. The easiest interface is to specify a speed code from 0 to 3, with higher numbers including more models but taking increasingly long to parse the text. Setting it equal to 2 is a good balance between time and feature-richness:

```
init_coreNLP(anno_level = 2L, lib_location = lib_loc)
```

After the pipeline is loaded, we again call run_annotators and set the backend to "coreNLP" (by default run_annotators will use whichever backend for most recently initalized, so this option is technically not needed if you just ran `init_coreNLP`):

```
obj <- run_annotators(text, as_strings = TRUE, backend = "coreNLP")
obj
```

```
##
## A CleanNLP Annotation:
##   num. documents: 1
```

The annotation object contains the same tables as the spaCy models, with slightly different fields filled in.

Saving annotations

Once an annotation object is created there are two ways of saving the output. Using `saveRDS` saves the output as a binary file which can be read back into R at any time. Alternatively, the function `write_annotation` saves the annotation as a collection of comma separated files:

```
od <- tempfile()
write_annotation(obj, od)
dir(od)
```

```
## [1] "dependency.csv" "document.csv"
## [3] "entity.csv"     "token.csv"
## [5] "vector.csv"
```

```
[1] "dependency.csv" "document.csv"   "entity.csv"     "token.csv"
[5] "vector.csv"
```

Notice that only those tables that are non-empty are saved. These may be read back into R as an annotation object using `read_annotation`:

```
anno <- read_annotation(od)
```

Alternatively, as these are just comma separated values, the data may be read directly using `read.csv` in R or whatever other programming language or software a user would like to work with.