Motivation and quanteda principles

Kenneth Benoit

14 November 2016

Core objectives

1. Simplicity
2. Power
3. Best practice
   - workflow
   - transparency and reproducability
4. Performance
5. Inter-operability with other tools

Basic Principles

1. Corpus texts are text repositories.
   - Should not have their texts modified as part of preparation or analysis
   - Subsetting or redefining documents is allowable
2. A corpus should be capable of holding additional objects that will be associated with the corpus, such as dictionaries, stopword, and phrase lists, that will propagate downstream.
3. Downstream objects should record the settings used to produce them.
   - includes: tokenized texts (`tokens`), document-feature matrixes (`dfm`)
   - settings examples are: `tolower`, `stem`, `removeTwitter` etc.
   - also include any objects used in feature selection, such as dictionaries or stopword lists

Basic Principles (cont.)

4. A document-feature matrix is a sparse matrix that is always *documents* (or document groups) in rows by *features* in columns.
5. Encoding of texts should be done in the corpus, and recorded as meta-data in the corpus
    - This encoding should be `UTF-8` by default (problem for Windows machines)
    - Basic text operations will use the `stringi` package, based on the ICU libraries for Unicode compliance

Text analysis workflow: Corpora, documents, and features

1. Creating the corpus
   - reading files, now in a separate text package called **readtexts**
   - creating a corpus using `corpus()`
   - adding document variables (`docvars`) and metadata (`metadoc` and `metacorpus`).
2. Defining and delimiting documents
   - defining what are "texts", for instance using `changeunits` or grouping

Text analysis workflow: Corpora, documents, and features (cont.)

3. Defining and delimiting textual features, using:
   - `tokens`, to indentify instances of defined features ("tokens") and extract them as vectors
   - usually these will consist of terms, but may also consist of:
     - `ngrams` and `skipgrams`, sequences of adjacent or nearby tokens
     - multi-word expressions, through `phrasetotoken`
   - in this step we also apply rules that will keep or ignore elements, such as
     - punctuation
     - numbers, including or currency-prefixed digits
     - URLs
     - Twitter tags
     - inter-token separators

Text analysis workflow: Corpora, documents, and features (cont.)

4. Further feature selection
   Once defined and extracted from the texts (the tokenization step), features may be:
   - *removed* or *kept* through use of predefined lists or patterns, using `selectFeatures`
   - *collapsed* by:
     - stemming, through the `stem` option in `dfm` or `wordstem.dfm()`
     - defining feature equivalency classes, either exclusively (`dfm` option `dictionary`) or as a supplement to uncollapsed features (`dfm` option `thesaurus`)
     - `toLower` to consider different cases as equivalent, by converting to lower case

Text analysis workflow: Analysis of documents and features

1. From a corpus.
   These steps don't necessarily require the processing steps above.
   - `kwic`
   - `lexdiv`
   - `summary`
2. From a dfm – after `dfm` on the processed document and features.

`dfm()`, the Swiss Army knife

1. Most common use case: from texts or corpus to dfm
   - most above options are available at this stage
2. If separate steps are desired
   - we can still perform same steps on intermediate objects (`tokenizedTexts`)
   - we can perform many operations of feature selection, removal, equivalencies on a dfm, or during its creation

Text analysis workflow: Analyzing a dfm

1. Many analyses are possible directly from the dfm

```
dfm                    print, show
convert                removeFeatures
docfreq                similarity
docnames               sort
features               textmodel
lexdiv                 topfeatures
ndoc                   trim
ntoken                 weight
plot                   settings
```

Text analysis workflow: Analyzing a dfm (cont.)

2. Plan is to incorporate wrappers for many `textmodel` functions that work in a similar fashion, e.g.
   - text regression
   - predictive methods (Naive Bayes, SVM, kNN, etc.)
   - scaling methods (Poisson scaling aka "wordfish", correspondence analysis)
3. Hands off nicely to other packages needing a dfm
   - `convert()` converts to formats needed by `topicmodels`, `LDA`, and `STM` packages