Introduction to the **wordnet** Package

Ingo Feinerer

June 8, 2024

Abstract

The **wordnet** package provides a R interface to the WordNet lexical database of English.

Introduction

The **wordnet** package provides a R via Java interface to the WordNet[1] lexical database of English which is commonly used in linguistics and text mining. Internally **wordnet** uses Jawbone[2], a Java API to WordNet, to access the database. Thus, this package needs both a working Java installation, activated Java under R support, and a working WordNet installation.

Since we simulate the behavior of Jawbone, its homepage is a valuable source of information for background information and details besides this vignette.

Loading the Package

The package is loaded via

```
> library("wordnet")
```

Dictionary

A so-called *dictionary* is the main structure for accessing the WordNet database. Before accessing the database the dictionary must be initialized with the path to the directory where the WordNet database has been installed (e.g., `/usr/local/WordNet-3.0/dict`). On start up the package searches environment variables (`WNHOME`) and default installation locations such that the WordNet installation is found automatically in most cases. On success the package stores internally a pointer to the WordNet dictionary which is used package wide by various functions. You can manually provide the path to the WordNet installation via `setDict()`.

[1]`https://wordnet.princeton.edu/`
[2]`https://sites.google.com/site/mfwallace/jawbone`

Filters

The package provides a set of filters in order to search for terms according to certain criteria. Available filter types can be listed via

```
> getFilterTypes()
```

```
[1] "ContainsFilter"   "EndsWithFilter"   "ExactMatchFilter"
[4] "RegexFilter"      "SoundFilter"      "StartsWithFilter"
[7] "WildcardFilter"
```

A detailed description of available filters gives the Jawbone homepage. E.g., we want to search for words in the database which start with `car`. So we create the desired filter with `getTermFilter()`, and access the first five terms which are nouns via `getIndexTerms()`. So-called *index terms* hold information on the word itself and related meanings (i.e., so-called *synsets*). The function `getLemma()` extracts the word (so-called *lemma* in Jawbone terminology).

```
> filter <- getTermFilter("StartsWithFilter", "car", TRUE)
> terms <- getIndexTerms("NOUN", 5, filter)
> sapply(terms, getLemma)
```

```
[1] "car"      "car-ferry"     "car-mechanic"     "car battery"
[5] "car bomb"
```

Synonyms

A very common usage is to find synonyms for a given term. Therefore, we provide the low-level function `getSynonyms()`. In this example we ask the database for the synonyms of the term `company`.

```
> filter <- getTermFilter("ExactMatchFilter", "company", TRUE)
> terms <- getIndexTerms("NOUN", 1, filter)
> getSynonyms(terms[[1]])
```

```
[1] "caller"     "companionship"   "company"   "fellowship"
[5] "party"      "ship's company"   "society"   "troupe"
```

In addition there is the high-level function `synonyms()` omitting special parameter settings.

```
> synonyms("company", "NOUN")
```

```
[1] "caller"     "companionship"   "company"   "fellowship"
[5] "party"      "ship's company"   "society"   "troupe"
```

Related Synsets

Besides synonyms, synsets can provide information to related terms and synsets. Following code example finds the antonyms (i.e., opposite meaning) for the adjective `hot` in the database.

```
> filter <- getTermFilter("ExactMatchFilter", "hot", TRUE)
> terms <- getIndexTerms("ADJECTIVE", 1, filter)
> synsets <- getSynsets(terms[[1]])
> related <- getRelatedSynsets(synsets[[1]], "!")
> sapply(related, getWord)

[1] "cold"
```