Word Stemming in R

Duncan Temple Lang
Department of Statistics,
UC Davis

August 4, 2004

Stemming is the process of removing suffixes from words to get the common origin. In statistical analysis, it greatly helps when comparing texts to be able to identify words with a common meaning and form as being identical. For example, we would like to count the words stopped and stopping as being the same and derived from stop. Stemming identifies these common forms.

This Rstem package provides an interface to C code that performs stemming on words. The basic interface is simple. You pass a collection of words as a character vector in S to the function `wordStem()` and it returns the corresponding stem for each word in a parallel character vector. The documentation for `wordStem()` provides examples.

The architecture of the stemming system created by Martin Porter allows others to specify the rules for stemming as inputs to a processor that generates C (or Java) code that implements that stemming procedure. There are already rules for languages including English, French, Danish, German, etc. One can query the available languages with built-in support within the package using the function `getStemLanguages()`.

```
> getStemLanguages()
 [1] "french"     "english"    "spanish"    "portuguese" "german"
 [6] "dutch"      "swedish"    "norwegian"  "danish"     "russian"
[11] "finnish"
```

To use the stemming algorithm for a particular language in `wordStem()`, one can specify the name of the language via the *language* argument.

```
wordStem(words, language = "norwegian")
```

The package is extensible in two ways. Firstly, it contains a script that can be used to download new C code from the Snowball Web site. As updated code for current languages or code for new languages becomes available, this package can be used to update itself and then re-installed. Indeed, this could be made a configuration option so that the up-to-the-minute versions of the Snowball-generated code would be downloaded and used, and only for the desired languages.

The second way in which this package is extensible is at run-time. The basic architecture of the Snowball-generated C code is that we first create a "context" or environment which is created and used each time we stem a collection of words. Typically, we just create a new `SN_env` object in the default manner. This allocates a new instance of the struct using the Snowball internals. There is a different way to do this for each language and so the creation/initialization routine is language-specific.

There are times when we might want to override the creation/initialization of these routines are self. For example, we might want add debugging information, or we might want to profile the stemming behaviour. Alternatively, we might want to cache stemmed words, or cache these structures so that new ones are recycled from a pool of existing ones. Or, more interestingly, we might want to "extended" the structure in some way and create it simply in a different way.

The **Rstem** package allows the caller to specify constructor and destructor routines to use to create and remove a `SN_env` object to use for this one stemming operation. We typically would specify the names of these routines. R then looks these up with all of the dynamically loaded code to which it has access and resolves these into the addresses of the corresponding C routines. In this case, we just give the names of the two routines. For example, if we wanted to use our own routines named *testDynCreate()* and *testDynClose()*, we could merely call `wordStem()` s

```
wordStem(words, language = c("testDynCreate", "testDynClose"))
```

One of the drawbacks about simply identifying the routines by name is that they may be found in the wrong dynamically loaded library. For example, we might have a routine *testDynCreate()* in a different library as well as in the one we expect to find the one of interest. If we find the wrong one simply because it is earlier in the search path, then, calamitous things will occur. At best, we will crash R. At worst, we will get wrong answers and not necessarily know.

To avoid this, it is better to explicitly resolve the routines of interest using the R function `getNativeSymbol()`. This allows us to specify the name of a routine and the name of the library (often the same as the package) in which to find it.

```
wordStem(stems,
         language = list(getNativeSymbolInfo("testDynCreate", "Rstem")$address,
                         getNativeSymbolInfo("testDynClose", "Rstem")$address,
                         getNativeSymbolInfo("testDynStem", "Rstem")$address)
        )
```

One of the reasons for adding this facility to the package is to illustrate how it might be used in other packages. We have an example of how to do this to extend the distance function in the **mva** package, and essentially how to avoid fixed sets of options in C code implemented by a switch statement.