Watch Your Spelling!

by Kurt Hornik and Duncan Murdoch

Abstract We discuss the facilities in base R for spell checking via Aspell, Hunspell or Ispell, which are useful in particular for conveniently checking the spelling of natural language texts in package Rd files and vignettes. Spell checking performance is illustrated using the Rd files in package **stats**. This example clearly indicates the need for a domain-specific statistical dictionary. We analyze the results of spell checking all Rd files in all CRAN packages and show how these can be employed for building such a dictionary.

R and its add-on packages contain large amounts of natural language text (in particular, in the documentation in package Rd files and vignettes). This text is useful for reading by humans and as a testbed for a variety of natural language processing (NLP) tasks, but it is not always spelled correctly. This is not entirely unsurprising, given that available spell checkers are not aware of the special Rd file and vignette formats, and thus rather inconvenient to use. (In particular, we are unaware of ways to take advantage of the ESS (Rossini et al., 2004) facilities to teach Emacs to check the spelling of vignettes by only analyzing the LaTeX chunks in suitable TeX modes.)

In addition to providing facilities to make spell checking for Rd files and vignettes more convenient, it is also desirable to have programmatic (R-level) access to the possibly mis-spelled words (and suggested corrections). This will allow their inclusion into automatically generated reports (e.g., by R CMD check), or aggregation for subsequent analyses. Spell checkers typically know how to extract words from text employing language-dependent algorithms for handling morphology, and compare the extracted words against a known list of correctly spelled ones, the so-called dictionary. However, NLP resources for statistics and related knowledge domains are rather scarce, and we are unaware of suitable dictionaries for these. The result is that domain-specific terms in texts from these domains are typically reported as possibly mis-spelled. It thus seems attractive to create additional dictionaries based on determining the most frequent possibly mis-spelled terms in corpora such as the Rd files and vignettes in the packages in the R repositories.

In this paper, we discuss the spell check functionality provided by `aspell()` and related utilities made available in the R standard package **utils**. After a quick introduction to spell checking, we indicate how `aspell()` can be used for checking individual files and packages, and how this can be integrated into typical work flows. We then compare the agreement of the spell check results obtained by two different programs (Aspell and Hunspell) and their respective dictionaries. Finally, we investigate the possibility of using the CRAN package Rd files for creating a statistics dictionary.

Spell checking

The first spell checkers were widely available on mainframe computers in the late 1970s (e.g., `http://en.wikipedia.org/wiki/Spell_checker`). They actually were "verifiers" instead of "correctors", reporting words not found in the dictionary but not making useful suggestions for replacements (e.g., as close matches in the Levenshtein distance to terms from the dictionary). In the 1980s, checkers became available on personal computers, and were integrated into popular word-processing packages like WordStar. Recently, applications such as web browsers and email clients have added spell check support for user-written content. Extending coverage from English to beyond western European languages has required software to deal with character encoding issues and increased sophistication in the morphology routines, particularly with regard to heavily-agglutinative languages like Hungarian and Finnish, and resulted in several generations of open-source checkers. Limitations of the basic unigram (single-word) approach have led to the development of context-sensitive spell checkers, currently available in commercial software such as e.g. Microsoft Office 2007. Another approach to spell checking is using adaptive domain-specific models for mis-spellings (e.g., based on the frequency of word n-grams), as employed for example in web search engines ("Did you mean …?").

The first spell checker on Unix-alikes was `spell` (originally written in 1971 in PDP-10 Assembly language and later ported to C), which read an input file and wrote possibly mis-spelled words to output (one word per line). A variety of enhancements led to (International) Ispell (`http://lasr.cs.ucla.edu/geoff/ispell.html`), which added interactivity (hence "i"-spell), suggestion of replacements, and support for a large number of European languages (hence "international"). It pioneered the idea of a programming interface, originally intended for use by Emacs, and awareness of special input file formats (originally, TeX or nroff/troff).

GNU Aspell, usually called just Aspell (`http://aspell.net`), was mainly developed by Kevin Atkinson and designed to eventually replace Ispell. Aspell can either be used as a library (in fact, the Omegahat package **Aspell** (Temple Lang, 2005) provides a fine-grained R interface to this) or as a standalone program. Compared to Ispell, Aspell can also easily check documents in UTF-8 without having to use a special dictionary, and supports using multiple dictionaries. It also tries to do a better job suggesting corrections (Ispell only suggests words with a Levenshtein distance of 1), and provides powerful and customizable TEX filtering. Aspell is the standard spell checker for the GNU software system, with packages available for all common Linux distributions. E.g., for Debian/Ubuntu flavors, **aspell** contains the programs, **aspell-en** the English language dictionaries (with American, British and Canadian spellings), and **libaspell-dev** provides the files needed to build applications that link against the Aspell libraries.

See `http://aspell.net` for information on obtaining Aspell, and available dictionaries. A native Windows build of an old release of Aspell is available on `http://aspell.net/win32/`. A current build is included in the Cygwin system at `http://cygwin.com`[1], and a native build is available at `http://www.ndl.kiev.ua/content/patch-aspell-0605-win32-compilation`.

Hunspell (`http://hunspell.sourceforge.net/`) is a spell checker and morphological analyzer designed for languages with rich morphology and complex word compounding or character encoding, originally designed for the Hungarian language. It is in turn based on Myspell, which was started by Kevin Hendricks to integrate various open source spelling checkers into the OpenOffice.org build, and facilitated by Kevin Atkinson. Unlike Myspell, Hunspell can use Unicode UTF-8-encoded dictionaries. Hunspell is the spell checker of OpenOffice.org, Mozilla Firefox 3 & Thunderbird and Google Chrome, and it is also used by proprietary software like MacOS X. Its TEX support is similar to Ispell, but nowhere near that of Aspell. Again, Hunspell is conveniently packaged for all common Linux distributions (with Debian/Ubuntu packages **hunspell** for the standalone program, dictionaries including **hunspell-en-us** and **hunspell-en-ca**, and **libhunspell-dev** for the application library interface). For Windows, the most recent available build appears to be version 1.2.8 on `http://sourceforge.net/projects/hunspell/files/`.

Aspell and Hunspell both support the so-called Ispell pipe interface, which reads a given input file and then, for each input line, writes a single line to the standard output for each word checked for spelling on the line, with different line formats for words found in the dictionary, and words not found, either with or without suggestions. Through this interface, applications can easily gain spell-checking capabilities (with Emacs the longest-serving "client").

Spell checking with R

The basic function for spell checking provided by the **utils** package is `aspell()`, with synopsis

```
aspell(files, filter, control = list(),
       encoding = "unknown", program = NULL)
```

Argument `files` is a character vector with the names of the files to be checked (in fact, in R 2.12.0 or later alternatively a list of R objects representing connections or having suitable srcrefs), `control` is a list or character vector of control options (command line arguments) to be passed to the spell check program, and `program` optionally specifies the name of the program to be employed. By default, the system path is searched for `aspell`, `hunspell` and `ispell` (in that order), and the first one found is used. Encodings which can not be inferred from the files can be specified via `encoding`. Finally, one can use argument `filter` to specify a filter for processing the files before spell checking, either as a user-defined function, or a character string specifying a built-in filter, or a list with the name of a built-in filter and additional arguments to be passed to it. The built-in filters currently available are `"Rd"` and `"Sweave"`, corresponding to functions `RdTextFilter` and `SweaveTeXFilter` in package **tools**, with self-explanatory names: the former blanks out all non-text in an Rd file, dropping elements such as `\email` and `\url` or as specified by the user via argument `drop`; the latter blanks out code chunks and Noweb markup in an Sweave input file.

`aspell()` returns a data frame inheriting from `"aspell"` with the information about possibly misspelled words, which has useful `print()` and `summary()` methods. For example, consider 'lm.Rd' in package **stats** which provides the documentation for fitting linear models. Assuming that `src` is set to the URL for the source of that file, i.e. `"http://svn.R-project.org/R/trunk/src/library/stats/man/lm.Rd"`, and `f` is set to `"lm.Rd"`, we can obtain a copy and check it as follows:

```
> download.file(src, f)
> a <- aspell(f, "Rd")
> a

accessor
  lm.Rd:128:25

ANOVA
  lm.Rd:177:7

datasets
  lm.Rd:199:37

differenced
  lm.Rd:168:57

formulae
  lm.Rd:103:27

Ihaka
  lm.Rd:214:68

logicals
  lm.Rd:55:26

priori
  lm.Rd:66:56
```

[1]The Cygwin build requires Unix-style text file inputs, and Cygwin-style file paths, so is not fully compatible with `aspell()` in R.

```
regressor
  lm.Rd:169:3
```

(results will be quite different if one forgets to use the Rd filter). The output is self-explanatory: for each possibly mis-spelled word, we get the word and the occurrences in the form *file*:*linenum*:*colnum*. Clearly, terms such as 'ANOVA' and 'regressor' are missing from the dictionary; if one was not sure about 'datasets', one could try visiting http://en.wiktionary.org/wiki/datasets.

If one is only interested in the possibly mis-spelled words themselves, one can use

```
> summary(a)
Possibly mis-spelled words:
[1] "accessor"    "ANOVA"       "datasets"
[4] "differenced" "formulae"    "Ihaka"
[7] "logicals"    "priori"      "regressor"
```

(or directly access the `Original` variable); to see the suggested corrections, one can print with `verbose = TRUE`:

```
> print(subset(a,
+              Original %in%
+              c("ANOVA", "regressor")),
+       verbose = TRUE)

Word: ANOVA (lm.Rd:177:7)
Suggestions: AN OVA AN-OVA NOVA ANICA ANIA
     NOVAE ANA AVA INVAR NOV OVA UNIV ANITA
     ANNORA AVIVA ABOVE ENVOY ANNA NEVA ALVA
     ANYA AZOV ANON ENVY JANEVA ANODE ARNO
     IVA ANVIL NAIVE OLVA ANAL AVON AV IONA
     NV NAVY OONA AN AVOW INFO NAVE ANNOY
     ANN ANY ARV AVE EVA INA NEV ONO UNA
     ANCHOVY ANA'S

Word: regressor (lm.Rd:169:3)
Suggestions: regress or regress-or regress
     regressive regressed regresses
     aggressor regressing egress regrets
     Negress redress repress recross regrows
     egress's Negress's
```

Note that the first suggestion is 'regress or'.

The print method also has an `indent` argument controlling the indentation of the locations of possibly mis-spelled words. The default is 2; Emacs users may find it useful to use an indentation of 0 and visit output (directly when running R from inside ESS, or redirecting output to a file using `sink()`) in `grep-mode`.

The above spell check results were obtained using Aspell with an American English dictionary. One can also use several dictionaries:

```
> a2 <- aspell(f, "Rd",
+              control =
+              c("--master=en_US",
+                "--add-extra-dicts=en_GB"))
> summary(a2)
Possibly mis-spelled words:
[1] "accessor"    "ANOVA"       "datasets"
[4] "differenced" "Ihaka"       "logicals"
[7] "priori"      "regressor"

> setdiff(a$Original, a2$Original)

[1] "formulae"
```

This shows that Aspell no longer considers 'formulae' mis-spelled when the "British" dictionary is added, and also exemplifies how to use the `control` argument: using Hunspell, the corresponding choice of dictionaries could be achieved using `control = "-d en_US,en_GB"`. (Note that the dictionaries differ, as we shall see below). This works for "system" dictionaries; one can also specify "personal" dictionaries (using the common command line option '`-p`').

We already pointed out that Aspell excels in its TEX filtering capabilities, which is obviously rather useful for spell-checking R package vignettes in Sweave format. Similar to Ispell and Hunspell, it provides a TEX mode (activated by the common command line option '`-t`') which knows to blank out TEX commands. In addition to the others, it allows to selectively blank out options and parameters of these commands. This works via specifications of the form '*name signature*' where the first gives the name of the command and the second is a string containing 'p' or 'o' indicating to blank out parameters or options, or 'P' or 'O' indicating to keep them for checking. E.g., '`foo Pop`' specifies to check the first parameter and then skip the next option and the second parameter (even if the option is not present), i.efollowing the pattern `\foo{checked}[unchecked]{unchecked}`, and is passed to Aspell as

```
  --add-tex-command="foo Pop"
```

Aspell's TEX filter is already aware of a number of common (LATEX) commands (but not, e.g., of `\citep` used for parenthetical citations using **natbib**). However, this filter is based on C++ code internal to the Aspell data structures, and hence not reusable for other spell checkers: we are currently exploring the possibility to provide similar functionality using R.

Package **utils** also provides three additional utilities (exported in R 2.12.0 or later):
`aspell_package_Rd_files()`,
`aspell_package_vignettes()`, and
`aspell_write_personal_dictionary_file()`.
The first two, obviously for spell checking the Rd files and (Sweave) vignettes in a package, are not only useful for automatically computing all relevant files and choosing the appropriate filter (and, in the case of vignettes, adding a few commands like `\Sexpr` and `\citep` to the blank out list), but also because they support a package default mechanism described below. To see a simple example, we use Aspell to check the spelling of all Rd files in package **stats**:

```
> require("tools")
> drop <- c("\\author", "\\source", "\\references")
> ca <- c("--master=en_US",
+         "--add-extra-dicts=en_GB")
> asa <- aspell_package_Rd_files("stats",
+          drop = drop, program = "aspell",
+          control = ca)
```

This currently (2011-05-11) finds 1114 possibly mis-spelled words which should hopefully all be false positives (as we regularly use `aspell()` to check the spelling in R's Rd files). The most frequent occurrences are

```
> head(sort(table(asa$Original), decreasing = TRUE))

  quantile dendrogram        AIC univariate
        57         44         42         42
 quantiles      ARIMA
        30         23
```

which should clearly be in every statistician's dictionary!

Package defaults for spell checking (as well as a personal dictionary file) can be specified in a '`defaults.R`' file in the '`.aspell`' subdirectory of the top-level package source directory, and are provided via assignments to suitably named lists, as e.g.

```
  vignettes <-
      list(control = "--add-tex-command=mycmd op")
```

for vignettes and assigning to `Rd_files` for Rd files defaults. For the latter, one can also give a `drop` default specifying additional Rd sections to be blanked out, e.g.,

```
  Rd_files <- list(drop = c("\\author",
                            "\\source",
                            "\\references"))
```

The default lists can also contain a `personal` element for specifying a package-specific personal dictionary. A possible work flow for package maintainers is the following: one runs the spell checkers and corrects all true positives, leaving the false positives. Obviously, these should be recorded so that they are not reported the next time when texts are modified and checking is performed again. To do this one re-runs the check utility (e.g., `aspell_package_vignettes()` and saves its results into a personal dictionary file using `aspell_write_personal_dictionary_file()`. This file is then moved to the '`.aspell`' subdirectory (named, e.g., '`vignettes.pws`') and then activated via the defaults file using, e.g.,

```
   vignettes <-
       list(control = "--add-tex-command=mycmd op",
            personal = "vignettes.pws")
```

Following a new round of checking and correcting, the procedure is repeated (with the personal dictionary file temporarily removed before re-running the utility and re-creating the dictionary file; currently, there is no user-level facility for reading/merging personal dictionaries).

A different work flow is to keep the previous spell check results and compare the current one to the most recent previous one. In fact, one of us (KH) uses this approach to automatically generate "check diffs" for the R Rd files, vignettes and Texinfo manuals and correct the true positives found.

A spell check utility R itself does not (yet) provide is one for checking a list of given words, which is easily accomplished as follows:

```
> aspell_words <- function(x, control = list()) {
+     con <- textConnection(x)
+     on.exit(close(con))
+     aspell(list(con), control = control)
+ }
```

Of course, this reports useless locations, but is useful for programmatic dictionary lookup:

```
> print(aspell_words("flavour"), verbose = TRUE)

Word: flavour (<unknown>:1:1)
Suggestions: flavor flavors favor flour
     flair flours floury Flor flavor's Fleur
     floor flyover flour's
```

(again using Aspell with an American English dictionary).

Spell checking performance

Should one use Aspell or Hunspell (assuming convenient access to both)? Due to its more powerful morphological algorithms, Hunspell is slower. For vignettes, Aspell performance is clearly superior, given its superior TeX capabilities. For Rd files, we compare the results we obtain for package **stats**. Similar to the above, we use Hunspell on these Rd files:

```
> ch <- "-d en_US,en_GB"
> ash <- aspell(files, filter,
+               program = "hunspell", control = ch)
```

(In fact, this currently triggers a segfault on at least Debian GNU/Linux squeeze on file '`lowess.Rd`' once the en_GB dictionary is in place: so we really check this file using en_US only, and remove two GB spellings.) We then extract the words and terms (unique words) reported as potentially mis-spelled:

```
> asaw <- asa$Original; asat <- unique(asaw)
> ashw <- ash$Original; asht <- unique(ashw)
> allt <- unique(c(asat, asht))
```

This gives 1114 and 355 words and terms for Aspell, and 789 and 296 for Hunspell, respectively. Cross-tabulation yields

```
> tab <- table(Aspell = allt %in% asat,
+              Hunspell = allt %in% asht)
> tab
```

```
      Hunspell
Aspell  FALSE TRUE
  FALSE     0   27
  TRUE     86  269
```

and the amount of agreement can be measured via the Jaccard dissimilarity coefficient, which we compute "by hand" via

```
> sum(tab[row(tab) != col(tab)]) / sum(tab)

[1] 0.2958115
```

which is actually quite large. To gain more insight, we inspect the most frequent terms found only by Aspell

```
> head(sort(table(asaw[! asaw %in% asht]),
+           decreasing = TRUE), 4L)

  quantile univariate  quantiles         th
        57         42         30         18
```

and similarly for Hunspell:

```
> head(sort(table(ashw[! ashw %in% asat]),
+           decreasing = TRUE), 4L)

      's Mallows'  hessian      BIC
      21        6        5        4
```

indicating that Aspell currently does not know quantiles, and some differences in the morphological handling (the apostrophe s terms reported by Hunspell are all instances of possessive forms of words typeset using markup blanked out by the filtering). It might appear that Hunspell has a larger and hence "better" dictionary: but in general, increasing dictionary size will also increase the true negative rate. To inspect commonalities, we use

```
> head(sort(table(asaw[asaw %in%
+                      intersect(asat, asht)]),
+           decreasing = TRUE),
+      12L)

 dendrogram          AIC        ARIMA         ARMA
         44           42           23           22
   Wilcoxon      Tukey's          GLM          Nls
         16           15           12           10
periodogram          MLE         GLMs         IWLS
         10            9            8            8
```

re-iterating the fact that a statistics dictionary is needed.

We can also use the above to assess the "raw" performance of the spell checkers, by counting the number of terms and words in the stats Rd file corpus. Using the text mining infrastructure provided by package **tm** (Feinerer et al., 2008), this can be achieved as follows:

```
> require("tm")
> texts <-
+     lapply(files, RdTextFilter, drop = drop)
> dtm <- DocumentTermMatrix(
+     Corpus(VectorSource(texts)),
+     control = list(removePunctuation = TRUE,
+                    removeNumbers = TRUE,
+                    minWordLength = 2L))
```

(the `control` argument is chosen to match the behavior of the spell checkers). This gives

```
> dtm

A document-term matrix (304 documents, 3533 terms)

Non-/sparse entries: 31398/1042634
Sparsity           : 97%
Maximal term length: 24

> sum(dtm)

[1] 69910
```

with 3533 terms and 69910 words, so that Aspell would give "raw" false positive rates of 10.05 and 1.59 percent for terms and words, respectively (assuming that the regular spell checking of the Rd files in the R sources has truly eliminated all mis-spellings).

Towards a dictionary for statistics

Finally, we show how the spell check results can be employed for building a domain-specific dictionary (supplementing the default ones). Similar to the above, we run Aspell on all Rd files in CRAN and base R packages (Rd files are typically written in a rather terse and somewhat technical style, but should nevertheless be a good proxy for current terminology in computational statistics). With results in `ac`, we build a corpus obtained by splitting the words according to the Rd file in which they were found, and compute its document-term matrix:

```
> terms <- ac$Original
> files <- ac$File
> dtm_f <- DocumentTermMatrix(
+     Corpus(VectorSource(split(terms, files))),
+     control =
+     list(tolower = FALSE,
+          minWordLength = 2L))
```

(Again, `minWordLength = 2L` corresponds to the Aspell default to ignore single-character words only.) This finds 50658 possibly mis-spelled terms in 43287 Rd files, for a total number of 332551 words. As is quite typical in corpus linguistics, the term frequency distribution is heavily left-tailed

```
> require("slam")
> tf <- col_sums(dtm_f)
> summary(tf)

  Min.  1st Qu.  Median    Mean  3rd Qu.      Max.
 1.000    1.000   2.000   6.565    4.000  3666.000
```

and can conveniently be visualized by plotting the cumulative frequencies for the terms sorted in decreasing order (see Figure 1):

```
> tf <- sort(tf, decreasing = TRUE)
> par(las=1)
> plot(seq_along(tf),
+      cumsum(tf) / sum(tf),
+      type = "l",
+      xlab = "Number of most frequent terms",
+      ylab = "Proportion of words",
+      panel.first = abline(v=1000, col="gray"))
```

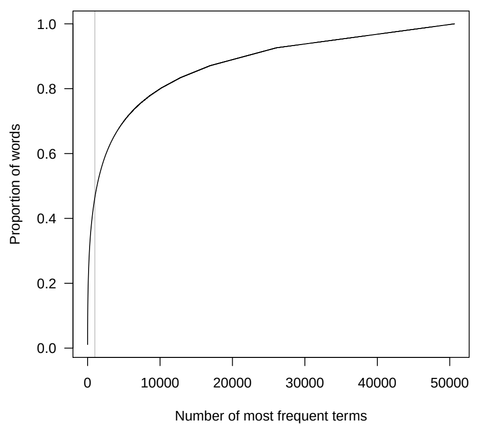

Figure 1: The number of possibly mis-spelled words (cumulative term frequency) in the Rd files in the CRAN and R base packages versus the number of terms (unique words), with terms sorted in decreasing frequency. The vertical line indicates 1000 terms.

We see that the 1000 most frequent terms already cover about half of the words. The corpus also reasonably satisfies Heap's Law (e.g., http://en.wikipedia.org/wiki/Heaps%27_law or Manning et al. (2008)), an empirical law indicating that the vocabulary size V (the number of different terms employed) grows polynomially with text size T (the number of words), as shown in Figure 2:

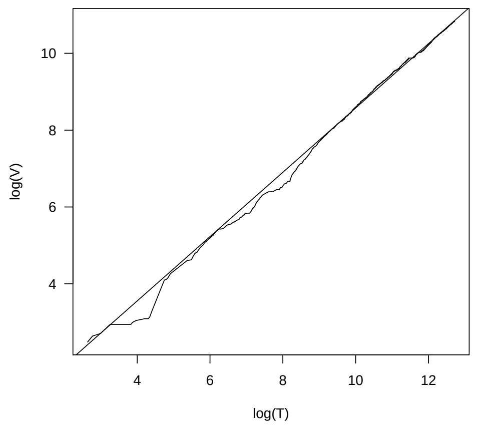

Figure 2: An illustration of Heap's Law for the corpus of possibly mis-spelled words in the Rd files in the CRAN and R base packages, taking the individual files as documents.

```
> Heaps_plot(dtm_f)

(Intercept)           x
  0.2057821   0.8371087
```

(the regression coefficients are for the model $\log(V) = \alpha + \beta \log(T)$).

To develop a dictionary, the term frequencies may need further normalization to weight their "importance". In fact, to adjust for possible authoring and package size effects, it seems preferable to aggregate the frequencies according to package. We then apply a simple, so-called binary weighting scheme which counts occurrences only once, so that the corresponding aggregate term frequencies become the numbers of packages the terms occurred in. Other weighting schemes are possible (e.g., normalizing by the total number of words checked in the respective package). This results in term frequencies `tf` with the following characteristics:

```
> summary(tf)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   1.000   1.000   2.038   1.000 682.000

> quantile(tf, c(seq(0.9, 1.0, by = 0.01)))

 90%  91%  92%  93%  94%  95%  96%  97%  98%  99%
   3    3    3    4    4    5    6    7   10   17
100%
 682
```

Again, the frequency distribution has a very heavy left tail. We apply a simple selection logic, and drop terms occurring in less than 0.5% of the packages (currently, about 12.5), leaving 764 terms for possible inclusion into the dictionary. (This also eliminates the terms from the few CRAN packages with non-English Rd files.) For further processing, it is highly advisable to take advantage of available lexical resources. One could programmatically employ the APIs of web search engines: but note that the spell correction facilities these provide are not domain specific. Using WordNet (Fellbaum, 1998), the most prominent lexical database for the English language, does not help too much: it only has entries for about 10% of our terms.

We use the already mentioned Wiktionary (http://www.wiktionary.org/), a multilingual, web-based project to create a free content dictionary which is run by the Wikimedia Foundation (like its sister project Wikipedia) and which has successfully been used for semantic web tasks (e.g., Zesch et al., 2008). Wiktionary provides a simple API at http://en.wiktionary.org/w/api.php for English language queries. One can look up given terms by queries using parameters `action=query`, `format=xml`, `prop=revision`, `rvprop=content` and `titles` as a list of the given terms collapsed by a vertical bar (actually, a maximum of 50 terms can be looked up in one query). When doing so via R, one can conveniently use the **XML** package (Temple Lang, 2010) to parse the result. For our terms, the lookup returns information for 416 terms. However, these can not be accepted unconditionally into the dictionary: in addition to some terms being flagged as mis-spelled or archaic, some terms have possible meanings that were almost certainly not intended (e.g., "wether" as a castrated buck goat or ram). In addition, 2-character terms need special attention (e.g., ISO language codes most likely not intended). Therefore, we extract suitable terms by serially working through suitable subsets of the Wiktionary results (e.g., terms categorized as statistical or mathematical (unfortunately, only a very few), acronyms or initialisms, and plural forms) and inspecting these for possible inclusion. After these structured eliminations, the remaining terms with query results as well as the ones Wiktionary does not know about (the majority of which actually are mis-spellings) are handled. Quite a few of our terms are surnames, and for now we only include the most frequent ones.

```
> p <- readLines("en-stats.pws")
```

We finally obtain a dictionary with 443 terms, which we save into 'en-stats.pws' using `aspell_write_personal_dictionary_file()`. We intend to make this easily available from within R at some point in the future.

A few false positives remain systematically: Aspell's word extraction turns '1st' and '2nd' into 'st' and 'nd' (and so forth), which clearly are not terms to be included in the dictionary. Similarly, 'a-priori' is split with 'priori' reported as mis-spelled (actually, Aspell has some support for accepting "run-together words"). These and other cases could be handled by enhancing the filtering routines before calling the spell checkers.

We re-run the checks on **stats** using this dictionary:

```
> asap <-
+     aspell(files, filter, program = "aspell",
+            control = c(ca, "-p ./en-stats.pws"))
```

(the strange '`./`' is needed to ensure that Aspell does not look for the personal dictionary in its system paths). This reduces the number of possibly misspelled words found to 411, and the "estimated" false positive rates for terms and words to 5.97 and 0.59 percent, respectively.

Many of the remaining false positives could also be eliminated by using the appropriate Rd markup (such as `\acronym` or `\code`). With R 2.12.0 or later, one can also use markup in titles. In fact, R 2.12.0 also adds `\newcommand` tags to the Rd format to allow user-defined macros: using, e.g., `\newcommand{\I}{#1}` one could wrap content to be ignored for spell checking into `\I{}`, as currently all user-defined macros are blanked out by the Rd filtering. Similar considerations apply to vignettes when using Aspell's superior filtering capabilities.

Using a custom dictionary has significantly reduced the false positive rate, demonstrating the usefulness of the approach. Clearly, more work will be needed: modern statistics needs better lexical resources, and a dictionary based on the most frequent spell check false alarms can only be a start. We hope that this article will foster community interest in contributing to the development of such resources, and that refined domain specific dictionaries can be made available and used for improved text analysis with R in the near future.

Bibliography

Ingo Feinerer, Kurt Hornik, and David Meyer. Text mining infrastructure in R. *Journal of Statistical Software*, 25(5):1–54, 2 2008. ISSN 1548-7660. URL `http://www.jstatsoft.org/v25/i05`.

Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998. ISBN 978-0-262-06197-1.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. URL `http://nlp.stanford.edu/IR-book/information-retrieval-book.html`.

Anthony J. Rossini, Richard M. Heiberger, Rodney Sparapani, Martin Mächler, and Kurt Hornik. Emacs speaks statistics: A multi-platform, multi-package development environment for statistical analysis. *Journal of Computational and Graphical Statistics*, 13(1):247–261, 2004.

Duncan Temple Lang. *Interface to aspell library*, 2005. URL `http://www.omegahat.org/Aspell`. R package version 0.2-0.

Duncan Temple Lang. *XML: Tools for parsing and generating XML within R and S-Plus.*, 2010. URL `http://CRAN.R-project.org/package=XML`. R package version 3.1-1.

Torsten Zesch, Christof Müller, and Iryna Gurevych. Using wiktionary for computing semantic relatedness. In *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence*, pages 861–866. AAAI Press, 2008. ISBN 978-1-57735-368-3.

Kurt Hornik
Department of Finance, Accounting and Statistics
Wirtschaftsuniversität Wien
Augasse 2–6, 1090 Wien
Austria
`Kurt.Hornik@R-project.org`

Duncan Murdoch
Department of Statistical and Actuarial Sciences
University of Western Ontario
London, Ontario, Canada
`murdoch@stats.uwo.ca`