Short Introduction to **tm.plugin.webmining**

Mario Annau
mario.annau@gmail.com

December 19, 2012

Abstract

This vignette gives a short introduction to **tm.plugin.webmining** which facilitates the retrieval of textual data from the web. The main focus of **tm.plugin.webmining** is the retrieval of web content from structured news feeds in the XML (RSS, ATOM) and JSON format. Additionally, retrieval and extraction of HTML documents is implemented. Numerous data sources are currently supported through public feeds/APIs, including Google– and Yahoo! News, Reuters, New York Times, and Twitter.

1 Getting Started

After package installation we make the functionality of **tm.plugin.webmining** available through

```
> library(tm.plugin.webmining)
```

tm.plugin.webmining depends on numerous packages, most importantly **tm** by Feinerer et˜al. (2008) for text mining capabilities and data structures. **RCurl** functions are used for web data retrieval and **XML** for the extraction of XML/HTML based feeds. As a first experiment, we can retrieve a (Web-)Corpus using data from Yahoo! News and the search query "Microsoft":

```
> yahoonews <- WebCorpus(YahooNewsSource("Microsoft"))
```

Users already familiar with **tm** will notice the different function call `WebCorpus()` for corpus construction. Like **tm**'s `Corpus()` constructor it takes a (Web-)Source object as input and constructs a (Web-)Corpus object. A Review of the object's `class()`

```
> class(yahoonews)
[1] "WebCorpus" "VCorpus"   "Corpus"    "list"
```

reveals, that WebCorpus is directly derived from Corpus and adds further functionality to it. It can therefore be used like a "normal" Corpus using **tm**'s text mining capabilities.

```
> yahoonews
A corpus with 20 text documents
```

Under the hood, a call of `YahooNewsSource()` retrieves a data feed from Yahoo! News and pre–parses its contents. Subsequently, `WebCorpus()` extracts (meta–)data from the WebSource object and also downloads and extracts the actual main content of the news item (most commonly an HTML–Webpage). In effect, it implements a two–step procedure to

1. Download meta data from the feed (through WebSource)
2. Download and extract main content for the feed item (through WebCorpus)

These procedures ensure that the resulting WebCorpus not only includes a rich set of meta data but also the full main text content for text mining purposes. An examination of the meta data for the first element in the corpus is shown below.

```
> meta(yahoonews[[1]])
Available meta data pairs are:
  Author       :
  DateTimeStamp: 2012-11-19 13:59:10
  Description  : Microsoft Corp. ) signed a multiyear agreement with Telefonica Brasil,...
  Heading      : Microsoft Inks Deal with Telefonica
  ID           : http://finance.yahoo.com/news/microsoft-inks-deal-telefonica-215910330...
  Language     :
  Origin       : http://finance.yahoo.com/news/microsoft-inks-deal-telefonica-215910330...
```

Source Name	Items	URL	Auth	Format
GoogleBlogSearchSource	100	`http://www.google.com/blogsearch`	-	RSS
GoogleFinanceSource	20	`http://www.google.com/finance`	-	RSS
GoogleNewsSource	100	`http://news.google.com`	-	RSS
GoogleReaderSource	1000+	`http://www.google.com/reader/atom/feed`	x	ATOM
NYTimesSource	100	`http://api.nytimes.com`	x	JSON
ReutersNewsSource	20	`http://www.reuters.com/tools/rss`	-	ATOM
TwitterSource	1500	`http://search.twitter.com/api`	-	ATOM
YahooFinanceSource	20	`http://finance.yahoo.com`	-	RSS
YahooInplaySource	100+	`http://finance.yahoo.com/marketupdate/inplay`	-	HTML
YahooNewsSource	20	`http://news.search.yahoo.com/rss`	-	RSS

Table 1: Overview of implemented WebSources listing the maximum number of items per feed, a descriptive URL, if authentification is necessary (x for yes) and the feed format.

For a Yahoo! News TextDocument we get useful meta–data like `DateTimeStamp`, `Description`, `Heading`, `ID` and `Origin`. The main content, as specified in the `Origin` of a TextDocument can be examined as follows (shortened for output):

```
> yahoonews[[1]]
```

```
Follow @bobmcmillan
Microsoft will use fuel-cell power plants -- similar to the three pictured here ...
```

It has been extracted from an unstructured HTML page and freed from ads and sidebar content by **boilerpipeR**'s `DefaultExtractor()`. To view the entire corpus main content also consider `inspect()` (output omitted):

```
> inspect(yahoonews)
```

2 Implemented Sources

All currently implemented (web–)sources are listed on Table˜1. The following commands show, how to use the implemented Sources. If available, the search query/stock ticker `Microsoft` has been used. Since Reuters News only offers a predefined number of channels we selected `businessNews`.

```
> googleblogsearch <- WebCorpus(GoogleBlogSearchSource("Microsoft"))
> googlefinance <- WebCorpus(GoogleFinanceSource("NASDAQ:MSFT"))
> googlenews <- WebCorpus(GoogleNewsSource("Microsoft"))
> nytimes <- WebCorpus(NYTimesSource("Microsoft", appid = nytimes_appid))
> reutersnews <- WebCorpus(ReutersNewsSource("businessNews"))
> twitter <- WebCorpus(TwitterSource("Microsoft"))
> yahoofinance <- WebCorpus(YahooFinanceSource("MSFT"))
> yahooinplay <- WebCorpus(YahooInplaySource())
> yahoonews <- WebCorpus(YahooNewsSource("Microsoft"))
```

3 Extending/Updating Corpora

Most data feeds only contain 20–100 feed items. A text corpus of such a small size may not be sufficient for text mining purposes. For that reason, the `corpus.update()` method has been implemented. In a nutshell, it first downloads a feed's meta data, checks which items are new (as determined by the meta–data ID field) and finally downloads the main content of new web documents. Since most time of WebCorpus construction is spend downloading the main content of corpus items, this procedures ensures a more efficient and faster WebCorpus–update.
The Yahoo! News corpus can now simply be updated:

```
> yahoonews <- corpus.update(yahoonews)
```

To continously update a WebCorpus a scheduled task/cron job could be set up which runs `corpus.update()` in a script.

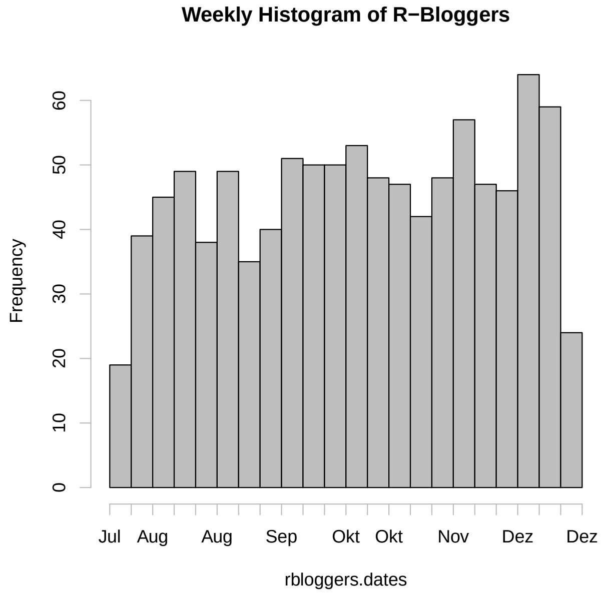

Figure 1: Weekly histogram of DateTimeStamps from R-Bloggers corpus metadata.

4 Google Reader for more data

A main limitation of most data feeds is the little number of items per feed. 20–100 items are barely enough for more serious text mining research. Continously updating corpora (e.g. by using `corpus.update()`) can be a quite time consuming task if a lot of history needs to be generated immediately.
At this point the Google Reader API comes into play. Google stores the histories of numerous news feeds for its Google Reader application. Accessing any feed through the provided Google Reader API can result in a much larger number of content items. To get access to the Google Reader API one first needs to retrieve an authentification token from Google (using a standard Google account):

```
> token <- auth.google.reader(email="<username>@gmail.com", password="<password>")
```

With the retrieved token string it is possible to get the content of any web-feed through the Google–Reader API. We can, for example, retrieve the content of the R-Bloggers feed with the command:

```
> rbloggers <- WebCorpus(GoogleReaderSource("http://feeds.feedburner.com/RBloggers", token,
+                                   params = list(n = 1000)))
```

Depending on the internet connection and number of items, this command can take quite a while. For larger requests and richer user feedback it is better to set the `verbose` option `TRUE`. Further we recommend to build an empty corpus first, without downloading any main content items. That way we can first check if the feed throught the Google Reader API is available and makes sense for our research purposes. By simply setting the WebSource's `$postFUN`[1] field to `NULL`, we build a corpus only consisting of meta data:

```
> rbloggers <- WebCorpus(GoogleReaderSource("http://feeds.feedburner.com/RBloggers", token,
+                                   params = list(n = 1000)), postFUN = NULL)
```

The number of items retrieved and the histogram of meta–`DateTimeStamp`s gives us information about the size and the update frequency of the feed. Figure˜1 shows the histogram for the feed.

```
> length(rbloggers)

[1] 1000

> rbloggers.dates <- do.call(c,lapply(rbloggers, meta, "DateTimeStamp"))
> hist(rbloggers.dates, breaks = "weeks", col = "grey", freq=T,
+                 main = "Weekly Histogram of R-Bloggers")
```

We can therefore start to download the entire feed with the `verbose` option enabled:

```
> options(verbose = TRUE)
> rbloggers <- WebCorpus(GoogleReaderSource("http://feeds.feedburner.com/RBloggers", token,
+                                   params = list(n = 1000)))
```

[1] specifies a function to be called after corpus generation. Typically set to retrieve main content items.

5 Conclusion

This vignette has given a short introduction to **tm.plugin.webmining**, a package to retrieve textual data from the web. Although **tm.plugin.webmining** has been tested for the retrieval of 10000+ items per feed it is generally not recommended to start massive feed downloads due to memory– and **RCurl** restrictions. For this purpose, web scraping frameworks like Scrapy (`scrapy.org`), Heritrix (`crawler.archive.org`) or Nutch (`nutch.apache.org`) are much better suited.
Keeping these issues in mind, **tm.plugin.webmining** is well suited for the retrieval and processing of small to medium sized text corpora. By using the full meta data and textual contents, quite interesting text mining experiments can be done using the full capabilities of the **tm** package.

References

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`.