Introduction to **tm.plugin.webmining**

Mario Annau
mario.annau@gmail.com

November 26, 2011

Abstract

This vignette gives an introduction to **tm.plugin.webmining**, an add-on package to **tm** 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. Furthermore, the direct retrieval through HTML documents is implemented. Numerous data sources are supported, including Google–,Yahoo!– and Bing News, Reuters, New York Times, Twitter, etc. In addition to simple feed content retrieval, also the complete source articles can be downloaded and extracted through **RCurl** and **boilerpipeR**.

Contents

1 Introduction

In recent years the research discipline known as *text mining*[1] has seen a tremendous growth not only in the academic area but also in fields such as business intelligence, customer relationship management (CRM), marketing and financial research. Text mining applications have been fuelled by the enormous growth of the largest text corpus of all—the Internet—with over one trillion unique URLs (Alpert and Hajaj, 2008). Developments in web technology, typically referred to as *Web 2.0*, including online review sites, personal blogs or even short, text messages called *tweets* further accelerated the growth of user generated online content. In most cases web content is easily accessible through standard web browsers. However, the systematic download of web content, including relevant meta data, and the provision to statistical analysis–/text–mining software like R is non–trivial. Generally speaking, there are two approaches to retrieve and update a collection of text documents from the web:

1. Retrieval through web *crawlers*, also known as *spiders* or *robots*
2. Retrieval of content through provided news *feeds*

The implementation of a web crawler is the most general retrieval approach that can be taken. Given a set of seed web pages, a web crawler recursively retrieves the content of documents, analyzes their link structure and follows the parsed links. The complexity and size of crawlers ranges from *topical crawlers* (e.g. extraction of customer reviews from vendor sites) to *universal crawlers* which index web documents of the entire (visible) Internet, as implemented by search engines like Google, Yahoo! and Bing. Although the implementation of (topical) crawlers can be facilitated by specialized frameworks, like the Python open–source project **Scrapy** (**?**), programming a web crawler includes complex tasks like the maintenance of a page repository, efficient page retrieval through concurrent programming and the provision and maintenance of required hardware/cloud infrastructure.
Fortunately, by the introduction of *Web 2.0* technologies, most information providers offer content updates in form of structured news feeds. Most popular feed formats are either based on the *Extensible Markup Language* (XML, see W3C, 2008) or JavaScript (Mozilla, 2011). The XML category consists of the *Really Simple Syndication* (RSS, see Board, 2002) and the *Atom Syndication Format* (ATOM, see IETF, 2005) which are very popular feed formats among content providers. The *JavaScript Object Notation* (JSON, see Crockford, 2002) is a JavaScript based content description language which has already gained a lot of traction among feed providers.
tm.plugin.webcorpus supports the retrieval of web documents based on news feeds in all of the above mentioned standardized data. Additionally, the direct retrieval from HTML pages is possible. Built on the **tm** text mining infrastructure package (Feinerer et˜al., 2008) it integrates numerous Source and Reader classes for web content retrieval. Major supported feed providers include search engines like Google, Microsoft Bing and Yahoo! or newswires like the New York Times or Thomson Reuters. The feeds available can be highly customized through user specified parameters and deliver important meta information. Provided meta information supplied by the feed includes, e.g., the author of the news text, links to the original text or the publishing date–/timestamp. Since hardly any feed delivers complete articles, full document content can be downloaded and extracted by **tm.plugin.webmining** if the link to the main content is provided. Please note that all presented R–code examples in this vignette require that the **tm.plugin.webmining** has been loaded.

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

2 Package Overview

To enable a wide range of web data retrieval use cases **tm.plugin.webmining** uses functions from numerous packges. All package dependencies (**tm**, **boilerpipeR**) and imports (**RCurl**, **XML**, **RJSONIO**) are available through CRAN.
In the context of **tm.plugin.webmining** these packages are used as follows:

- **tm**: Text mining infrastructure package (Feinerer et˜al., 2008), providing data structures and functions for text corpus storage, preprocessing (e.g. stop word removal) and document–term–matrix generation.
- **boilerpipeR**: Provision of functions to extract the main content from HTML files through the boilerpipe Java library (Kohlschütter et˜al., 2010).
- **RCurl**: For web data retrieval the **RCurl** package is imported. Especially the function `getURLAsynchronous()` is used (through `getURL()`) as it implements a fast, multithreaded way to download web pages from the Internet.
- **XML**: To parse RSS Feeds, ATOM Feeds, XML– or HTML Trees from the web, the package **XML** (Lang) provides all necessary functions and interfaces. It builds upon the popular **libxml** C library.
- **RJSONIO**: JSON feeds are supported and can be parsed using the **RJSONIO** package.

[1]Text mining can be defined as the statistical analysis of large collections of text to gather valuable information.

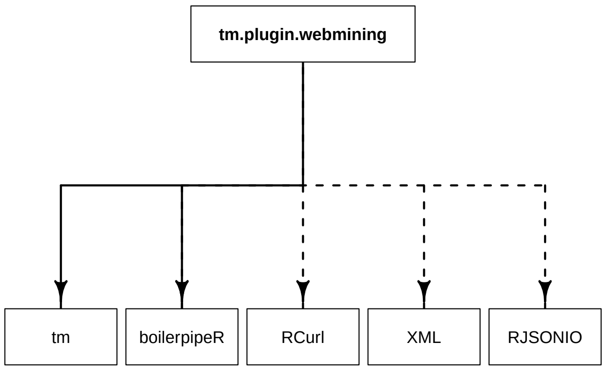

Figure 1: Illustration of the **tm.plugin.webmining** package dependencies. Solid lines indicate dependencies, dashed lines imports (as indicated by the package DESCRIPTION file). Figure created using **diagram**.

3 Content Retrieval

The complexity of feed–based web content retrieval depends on the required text document length. Apart from the delivered meta data most news feeds only deliver a fraction of the original news article. An example of a feed item from the Google News data feed for the query words `"Barack+Obama"` is shown below:

```
...
<item>
  <title>Obama ups pressure on Congress over veterans job aid ...</title>
  <link>http://news.google.com/news/url?sa=t&amp;fd=R&amp;... </link>
  <guid isPermaLink="false">tag:news.google.com,2005:cluster=...</guid>
  <pubDate>Mon, 07 Nov 2011 18:48:28 GMT</pubDate>
  <description>&lt;table border="0" ...</description>
</item>
...
```

Each feed item includes the following meta information:

- `<title>` Title of the news item
- `<link>` External link where actual content resides
- `<guid>` Internal Google News id
- `<pubDate>` Date when news item has been published
- `<description>` Short description of news item

If full–text articles need to be downloaded the retrieval procedure includes the download and extraction of the original news article source, as indicated by the *link* meta data field. Therefore, the download of text documents from news feeds, including the full article text requires the following retrieval steps:

1. Download data feed of interest.
2. Download source articles, specified in link tag of feed items.
3. Extract complete meta data from news feeds.
4. Extract main content from source articles.

These retrieval tasks are implemented by **tm.plugin.webmining** to retrieve and extract articles from news feeds. Since **tm.plugin.webmining** extends the **tm** package theses steps have been incorporated into **tm**'s defined Source–Reader concept. In this context, the Source defines the access to the text items of interest. The Reader includes the extraction/mapping logic from the raw data to the Corpus. We have therefore decided to map the entire download–part to the Source (retrieval steps 1–2) and leave the data extraction to the Reader (steps 3–4).

3.1 WebSource()

For the purposes of feed retrieval, **tm.plugin.webmining** introduces the `WebSource()` function. It represents a generic way of downloading feeds and source articles and returns a WebSource object, derived from **tm**'s Source. The definition of `WebSource()` is shown below.

```
> args(WebSource)

function (feedurls, class = "WebXMLSource", parser, encoding = "UTF-8",
    vectorized = FALSE, curlOpts = curlOptions(followlocation = TRUE,
        maxconnects = 20, maxredirs = 10, timeout = 30, connecttimeout = 30))
NULL
```

The main ingredients for the definition of a WebSource are the arguments `feedurls`, `parser` and `linkreader`.

- `feedurls`: Character vector of of feed urls to be downloaded.
- `parser`: Function which retrieves feed content and list of feed items. It therefore acts as a feed item chunker.
- `linkreader`: Function which takes a single item (as obtained from list of parsed items) and extracts the link of the source document. `WebSource()` downloads source articles in a second step.

All defined `<Provider>Source()` functions like `GoogleNewsSource()` or `TwitterSource()` are simply calls to `WebSource()` with specific parameter settings for `feedurls`, `parser` and `linkreader`. Each feed provider has a specific logic how the actual feed URL is organized. Fortunately, most feed URLs follow the following, standardized form:

```
<feedURL> ?  <param1> = <value1> & ...  & <paramN> = <valueN>
```

The integrated `feedquery()` function generates complete feed URLs and is very helpful if multiple URLs with slightly different parameter settings need to be created. The most frequent use case is the generation of various result pages for a common search term (in case of *paginated* result pages).
For our Google News example with the search terms `"Barack+Obama"` and various parameter settings, the feed URL can be created as follows:

```
> params = list(        hl= 'en',
+                                q = 'Barack+Obama',
+                                ie='utf-8',
+                                num = 100,
+                                output='rss')
> feed <- "http://news.google.com/news"
> fq <- feedquery(feed, params)
> fq

[1] "http://news.google.com/news?hl=en&q=Barack%2BObama&ie=utf-8&num=100&output=rss"
```

Additionally, we need a definition of a parser function which chunks different news article items into a list. For example, the Google News feed content needs to be chunked by the `<item>` tag. Therefore, the definition of the Google News parser function is as follows:

```
> parser <- function(cr){
+                 tree <- xmlInternalTreeParse(cr, asText = TRUE)
+                 nodes <- xpathSApply(tree, path = "//item")
+                 xmlns1 <- lapply(nodes, newXMLNamespace, "http://purl.org/dc/elements/1.1/", "dc")
+                 nodes
+         }
```

After the generation of the XML tree through `xmlInternalTreeParse()` we extract the feed items using `xpathSApply()`. Please note that the ex–post addition of XML namespaces is currently needed to surpress warning messages in the Reader.
Finally, the `linkparser` function needs to be defined to retrieve the link of the source document from the news item.

```
> linkreader <- function(tree) getNodeSet(tree, ".//link", fun = xmlValue)
```

Now we have all important `WebSource()` parameters at hand to complete the definition of our `GoogleNewsSource()` function, as implemented in **tm.plugin.webmining**:

```
> GoogleNewsSource <- function(query, params =
+                                list(       hl= 'en',
+                                                 q = query,
+                                                 ie='utf-8',
+                                                 num = 100,
+                                                 output='rss'), ...){
+         feed <- "http://news.google.com/news"
+
+         fq <- feedquery(feed, params)
+         parser <- function(cr){
+                 tree <- xmlInternalTreeParse(cr, asText = TRUE)
+                 nodes <- xpathSApply(tree, path = "//item")
+                 xmlns1 <- lapply(nodes, newXMLNamespace, "http://purl.org/dc/elements/1.1/", "dc")
+                 nodes
+         }
+         linkreader <- function(tree) getNodeSet(tree, ".//link", fun = xmlValue)
+
+         ws <- WebSource(feedurls = fq, parser = parser, linkreader = linkreader, ...)
+         ws$DefaultReader = readGoogle
+         class(ws) <- c("WebXMLSource", "Source")
+         ws
+ }
```

This function retrieves all required feed– and news article contents through `WebSource()`. To test `GoogleNewsSource()`, we again will consider the `"Barack+Obama"` search query.

`<TODO: insert example here>`

From the summary we can see, that the retrieved Source object contains the fields `$Content` and `$LinkContent`, where feed meta data items and source article contents are stored. These data fields are extracted by the specified (default) reader `readGoogle()`, which is described in the next section. The default reader has already been specified in `GoogleNewsSource()` and is stored in `$DefaultReader`. Futhermore, the class of the retrieved Source object has been set to WebXMLSource. The specification of an according Web<type>Source–class is necessary for the retrieval of single elements in the `Corpus()`–constructor. The class needs to be derived from **tm**'s Source and can be one of the following:

- WebXMLSource: XML based feed source for, e.g., XML, RSS and ATOM feeds.
- WebHTMLSource: HTML based feed source
- WebJSONSource: Feed source based on the JSON format.

3.2 readWeb()

Once the `WebSource()` function has been defined, we need a function to extract retrieved content from the `$Content` and `$LinkContent` fields and make it available to **tm**'s `Corpus()` constructor. Similar to the implementation of **tm**'s `readXML()`, **tm.plugin.webmining** implements a generic FunctionGenerator `readWeb()`, which handles the extraction of Web–Sources.

```
> args(readWeb)

function (spec, doc, parser, contentparser, freeFUN = NULL, ...)
NULL
```

Typically, `readWeb()` is only called through the following customization functions, which specify the parameters `parser`, `contentparser` and `freeFUN` depending on the data format:

- `readWebXML()`: Read contents from WebXMLSource
- `readWebHTML()`: Read contents from WebHTMLSource
- `readWebJSON()`: Read contents from WebJSONSource

The remaining parameters `spec`, `doc` and `extractFUN` need to be specified by the according `read<Provider>()` function and can be described as follows:

- `spec`: List–of–lists, specification of mapping and extraction rules, similar to `readXML()` in **tm**.
- `doc`: Document type, see **tm** for available document types.
- `extractFUN`: Extraction function to be used to content of interest from document. Takes source document as character and returns extracted content as character. For example, **boilerpipeR** integrates various functions to retrieve the main content from HTML documents.

The `readGoogle()` reader function can therefore be written as follows (as specified in **tm.plugin.webmining**):

```
> readGoogle <- readWebXML(spec = list(
+                 Heading = list("node", "//title"),
+                 DateTimeStamp = list("function", function(node){
+                                         loc <- Sys.getlocale("LC_TIME")
+                                         Sys.setlocale("LC_TIME", "C")
+                                         val <- sapply(getNodeSet(node, "//pubDate"), xmlValue)
+                                         time <- strptime(val,format = "%a, %d %b %Y %H:%M:%S",tz = "GMT")
+                                         Sys.setlocale("LC_TIME", loc)
+                                         time
+                                 }),
+                 Origin = list("node", "//link"),
+                 Description = list("function", function(node){
+                                         val <- sapply(getNodeSet(node, "//item/description"), xmlValue)
+                                         extractHTMLStrip(val, asText = TRUE)
+                                 }),
+                 ID = list("node",  "//guid")),
+         extractFUN = ArticleExtractor,
+         doc = PlainTextDocument()
+ )
```

The definition for our Google News WebXMLSource class and the according `readGoogle()` function is now finished, and we can retrieve a **tm** Corpus for the search terms `"Barack+Obama"` by simply typing:

```
> Corpus(GoogleNewsSource("Barack+Obama"), readerControl = list(reader = readGoogle, language = "en"))
```

Since the `$DefaultReader` has also been set, we could also type simply:

```
> Corpus(GoogleNewsSource("Barack+Obama"))
```

4 Examples

4.1 Twitter

To retrieve, e.g., all twitter messages including the hashtag `#BarackObama`, assuming the package **tm.plugin.webmining** is loaded, we can simply type:

```
> barackTwitter <- Corpus(TwitterSource("#BarackObama"))
```

This command retrieves (at most) the latest 1,500 tweets for the search term `#BarackObama`. The content of the first retrieved message is shown below:

```
> barackTwitter[[1]]
```

```
Switch accounts. Re-follow the Brobama. #barackobama #potus2012
```

Additionaly, the meta data of the first message can be inspected by using `meta()`

```
> meta(barackTwitter[[1]])
```

```
Available meta data pairs are:
  Author       : TheDeepLight (The Deep Light)
  DateTimeStamp: 2011-11-09 20:57:01
  Description  :
  Heading      :
  ID           : tag:search.twitter.com,2005:134373950854672384
  Language     : en
  Origin       :
User-defined local meta data pairs are:
$AuthorURI
[1] "http://twitter.com/TheDeepLight"

$Updated
[1] "2011-11-09 20:57:01 GMT"

$Source
[1] "<a href=\"http://www.tweetdeck.com\" rel=\"nofollow\">TweetDeck</a>"

$Geo
[1] ""
```

We can see, that all meta data tags have correctly been retrieved.

4.2 Reuters

For the retrieval o

4.3 New York Times

5 Conclusion and Outlook

The presented package **tm.plugin.webmining** integrates various functions to conveniently retrieve and extract text documents from numerous web data sources. Some news feeds already deliver a large number of feed items, like the Twitter Search API with up to 1,500 messages. However, histograms of the retrieved feed–item Date–Timestamps reveal that most feeds only cover a quite short period of time. Therefore, an update function for the retrieved text–Corpus is needed to grow the retrieved text collection further over time.

References

Jesse Alpert and Nissan Hajaj. We knew the web was big... - the official google blog., 7 2008. URL `http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html`.

RSS˜Advisory Board. Rss 2.0 specification, 2002. URL `http://www.rssboard.org/rss-specification`.

Douglas Crockford. Introducing json, 2002. URL `http://www.json.org`.

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`.

IETF. The atom syndication format, 2005. URL `http://tools.ietf.org/html/rfc4287`.

Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 441–450, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-889-6. doi: 10.1145/1718487.1718542. URL `http://code.google.com/p/boilerpipe/`.

Duncan˜Temple Lang. Xml: Tools for parsing and generating xml within r and s-plus. URL `http://www.omegahat.org/RSXML`.

Mozilla. Javascript, 2011. URL `https://developer.mozilla.org/en/JavaScript#Documentation`.

W3C. Extensible markup language (xml) 1.0 (fifth edition), 11 2008. URL `http://www.w3.org/TR/REC-xml/`.