Technical Report 2012

R 语言环境下的文本挖掘

Text Mining in R

Version 0.02 20120321

刘思喆

主页更新 http://bjt.name/todo

联系方式 sunbjt@gmail.com

新浪微博 @ 刘思喆

Copyright ©2012 R and all the Contributors to R tm. All rights reserved.
R 以及 R tm 的作者拥有版权 ©2012。保留所有权利。
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with the Invariant Sections being Contributors, no Front-Cover Texts, and no Back-Cover Texts.
你可以拷贝、发布或者修改这份文档，但必须遵守自由软件组织颁布的 GNU 自由文档许可证 1.2 或者以后版本的条款。Invariant Sections 包括 Contributors，没有 Front-Cover Texts 和 Back-Cover Texts。

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！

目录

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！

1 文本挖掘介绍

文本挖掘被描述为“自动化或半自动化处理文本的过程”，包含了文档聚类、文档分类、自然语言处理、文体变化分析及网络挖掘等领域内容。

对于文本处理过程首先要拥有分析的语料（text corpus），比如报告、信函、出版物等。而后根据这些语料建立半结构化的文本库（text database）。而后生成包含词频的结构化的词条-文档矩阵（term-document matrix）。

图 1: 文本挖掘的处理流程

这个一般性数据结构会被用于后续的分析，比如：

- 文本分类，比如根据现有的文本分类情况，对未知文本进行归类；
- 语法分析；
- 信息提取和修复；
- 文档信息汇总，比如提取相关有代表性的关键词、句子等。

2 自然语言处理技术综述

2.1 相关的 R 包

Phonetics and Speech Processing:

emu is a collection of tools for the creation, manipulation, and analysis of speech databases. At the core of EMU is a database search engine which allows the researcher to find various speech segments based on the sequential and hierarchical structure of the utterances in which they occur. EMU includes an interactive labeller which can display spectrograms and other speech waveforms, and which allows the creation of hierarchical, as well as sequential, labels for a speech utterance.

Lexical Databases:

wordnet provides an R interface to WordNet , a large lexical database of English.

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！

Keyword Extraction and General String Manipulation:

- R's base package already provides a rich set of character manipulation routines.
- RKEA provides an R interface to KEA (Version 5.0). KEA (for Keyphrase Extraction Algorithm) allows for extracting keyphrases from text documents. It can be either used for free indexing or for indexing with a controlled vocabulary.
- gsubfn can be used for certain parsing tasks such as extracting words from strings by content rather than by delimiters. demo("gsubfn-gries") shows an example of this in a natural language processing context.
- tau contains basic string manipulation and analysis routines needed in text processing such as dealing with character encoding, language, pattern counting, and tokenization.

Natural Language Processing:

- openNLP provides an R interface to OpenNLP , a collection of natural language processing tools including a sentence detector, tokenizer, pos-tagger, shallow and full syntactic parser, and named-entity detector, using the Maxent Java package for training and using maximum entropy models.
- openNLPmodels.en ships trained models for English and openNLPmodels.es for Spanish to be used with openNLP.
- RWeka is a interface to Weka which is a collection of machine learning algorithms for data mining tasks written in Java. Especially useful in the context of natural language processing is its functionality for tokenization and stemming.
- Snowball provides the Snowball stemmers which contain the Porter stemmer and several other stemmers for different languages. See the Snowball webpage for details.
- Rstem is an alternative interface to a C version of Porter's word stemming algorithm.
- KoNLP provides a collection of conversion routines (e.g. Hangul to Jamos), stemming, and part of speech tagging through interfacing with the Lucene's HanNanum analyzer.

Text Mining:

- tm provides a comprehensive text mining framework for R. The Journal of Statistical Software article Text Mining Infrastructure in R gives a detailed overview and presents techniques for count-based analysis methods, text clustering, text classification and string kernels.
- lsa provides routines for performing a latent semantic analysis with R. The basic idea of latent semantic analysis (LSA) is, that text do have a higher order (=latent semantic) structure which, however, is obscured by word usage (e.g. through the use of synonyms

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！

or polysemy). By using conceptual indices that are derived statistically via a truncated singular value decomposition (a two-mode factor analysis) over a given document-term matrix, this variability problem can be overcome. The article Investigating Unstructured Texts with Latent Semantic Analysis gives a detailed overview and demonstrates the use of the package with examples from the are of technology-enhanced learning.

- topicmodels provides an interface to the C code for Latent Dirichlet Allocation (LDA) models and Correlated Topics Models (CTM) by David M. Blei and co-authors and the C ++ code for fitting LDA models using Gibbs sampling by Xuan-Hieu Phan and co-authors.
- RTextTools is a machine learning package for automatic text classification. It implements the nine different algorithms (svm, slda, boosting, bagging, rf, glmnet, tree, nnet, and maxent) and routines supporting the evaluation of accuracy.
- textcat provides support for n-gram based text categorization.
- corpora offers utility functions for the statistical analysis of corpus frequency data.
- zipfR offers some statistical models for word frequency distributions. The utilities include functions for loading, manipulating and visualizing word frequency data and vocabulary growth curves. The package also implements several statistical models for the distribution of word frequencies in a population. (The name of this library derives from the most famous word frequency distribution, Zipf's law.)
- maxent is an implementation of maxinum entropy minimising memory consumption of very large data-sets.
- TextRegression predict valued outputs based on an input matrix and assess predictive power ('the bag-of-words oracle').
- wordcloud provides a visualisation similar to the famous wordle ones: it horizontally and vertically distributes features in a pleasing visualisation with the font size scaled by frequency.

2.2 stemming 和 Tokenization

在英文世界，有些单词是需要进行变化的，比如我们要识别 cat 这个字符，但还可能有 catlike、catty、cats 等单词，这时候就需要进行词干化（stemming）。例如：

```
library(Snowball)
> SnowballStemmer(c('functions', 'stemming', 'liked', 'doing'))
[1] "function" "stem"     "like"     "do"
```

而单个词有时候并不能完全表达出作者的意思，所以记号化（Tokenization）是将一段文本分割成叫做 token（象征）过程，token 可能是单词、短语、符号或其他有意义的元素。比如：

```
> NGramTokenizer(' 中华人民共和国 成立 于 1949年')
```

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！


```
[1] "中华人民共和国 成立 于" "成立 于 1949年"          "中华人民共和国 成立"
[4] "成立 于"                "于 1949年"               "中华人民共和国"
[7] "成立"                   "于"                      "1949年"
```

2.3 中文分词

在英文结构中，每个表意的单词都是是独立，但中文表意的字符之间没有空格连接。比如对于“花儿为什么这样红”，包含了 8 个字符，虽然对于任意一个中国人都可以分清这句话应该怎样断句，但对于电脑却有些困难。好在有一些现成的工具可以使用比如 **Rwordseg** 这个包 [Li, 2012]，即调用了 Java 分词程序对语句进行分词。下面是一个例子：

```
library(Rwordseg)
> segmentCN(' 花儿为什么这样红')
[1] "花儿"   "为什么" "这样"   "红"
```

这段话被分为了“花儿”，“为什么”，“这样”，“红”。这个空格分隔的结果集，后期就可以按照英文的方式进行后续处理。

由于 **Rwordseg** 这个包已经发布在 R-forge 仓库上，正常安装：

```
install.packages("Rwordseg", repos = "http://R-Forge.R-project.org")
```

当然，推荐使用 Rwordseg 的中文分词，它不但分词的精度较高，且支持词性标注等功能。**rmmseg4j** 也是可供选择的中文分词引擎 [Ronggui, 2011]。

3 tm 包

本部分主要涉及 R 语言环境中 tm 包的使用。

3.1 简介

使用 R 里面 tm 包进行文本挖掘，对于中文环境还有一些包来处理中文字符。关于 tm 包最早发表于 *Journal of Statistical Software* [Feinerer et al., 2008]，而关于 text mining in R 最早表在 *R News* [Feinerer, 2008]。

3.2 数据读入

在 tm 中主要的管理文件的结构被称为语料库（Corpus），代表了一系列的文档集合。语料库是一个概要性的概念，在这里分为动态语料库（Volatile Corpus，作为 R 对象保存在内存中）和静态语料库（Permanent Corpus，R 外部保存）。

在语料库构成中，x 必须有一个说明资料来源（input location）的源对象（Source Object）。我们可以看一下 tm 中 Corpus（或 VCorpus）函数的用法，对于动态语料库：

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！


```
Corpus(x,
        readerControl = list(reader = x$DefaultReader, language = "en"),
        ...)
```

在 tm 中静态语料库也是可以处理的，但需要使用 filehash 包来支持：

```
PCorpus(x,
        readerControl = list(reader = x$DefaultReader, language = "en"),
        dbControl = list(dbName = "", dbType = "DB1"),
        ...)
```

对于这些资料来源（即 x），tm 包提供了一些相关的函数，比如

- DirSource：处理目录
- VectorSource：由文档构成的向量
- DataframeSource：数据框，就像 CSV 文件

第二个参数是 readerControl，这里必须声明 reader 和 language 两个内容。第一个 reader 是指从资料源创立的文本文件。tm 包提供了一系列的函数支持（比如，readPlain(), readGmane(), readRCV1(), readReut21578XMLasPlain(), readPDF(), readDOC() 等）。可以使用 getReaders() 获得这些函数的列表。对于每一类源，都会有自己默认的 reader。比如对 DirSource 来说，默认读入输入文件并把内容解释为文本。第二个 language 就比较简单了，即字符集，比如可能是 UTF-8 字符集。

在使用静态语料库条件下，会涉及第三个参数 dbControl，它用来声明 R 内存对象外的资料来源（比如数据库）。dbType 控制了包 filehash 支持的数据库类型。数据库支持可以有效的减少对内存的要求，但数据的访问会受到硬盘的读写能力限制。

比如，txt 目录下的包含拉丁字符的纯文本，内容是罗马诗人奥维德 Ovid 的诗集，可以这样读进来：

```
> txt <- system.file("texts", "txt", package = "tm")
> (ovid <- Corpus(DirSource(txt),
+                 readerControl = list(language = "lat")))
A corpus with 5 text documents
```

当然同样也可以从字符向量创建语料库：

```
> docs <- c("This is a text.", "This another one.")
>  Corpus(VectorSource(docs))
A corpus with 2 text documents
```

最后我们根据路透社文档创建一个语料库，用于后续示例：

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！


```
> reut21578 <- system.file("texts", "crude", package = "tm")
>  reuters <- Corpus(DirSource(reut21578),
+                    readerControl = list(reader = readReut21578XML))
> reuters
A corpus with 20 text documents
```

3.3 数据输出

加入你在 R 中创建了一个语料库，但硬盘上并没有，那么使用 writeCorpus() 函数保存。

```
> writeCorpus(ovid)
```

这时，在工作目录下会生成与语料库对应的多个纯文本文件。

3.4 语料库的提取

对于 print() 和 summary() 函数依然有效，但源信息被隐藏（可以想象一下每个语料库包含了大量的文本，就像数据库一样）。summary() 函数会提供更多的元数据（meta data）信息，完整信息的提取需要使用 inspect()，比如：

```
> inspect(ovid[1:2])
A corpus with 2 text documents

The metadata consists of 2 tag-value pairs and a data frame
Available tags are:
  create_date creator
Available variables in the data frame are:
  MetaID

$ovid_1.txt
    Si quis in hoc artem populo non novit amandi,
         hoc legat et lecto carmine doctus amet.
    arte citae veloque rates remoque moventur,
         arte leves currus: arte regendus amor.

    curribus Automedon lentisque erat aptus habenis,
         Tiphys in Haemonia puppe magister erat:
    me Venus artificem tenero praefecit Amori;
         Tiphys et Automedon dicar Amoris ego.
    ille quidem ferus est et qui mihi saepe repugnet:
```

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！


```
     sed puer est, aetas mollis et apta regi.
Phillyrides puerum cithara perfecit Achillem,
     atque animos placida contudit arte feros.
qui totiens socios, totiens exterruit hostes,
     creditur annosum pertimuisse senem.

$ovid_2.txt
quas Hector sensurus erat, poscente magistro
     verberibus iussas praebuit ille manus.
Aeacidae Chiron, ego sum praeceptor Amoris:
     saevus uterque puer, natus uterque dea.
sed tamen et tauri cervix oneratur aratro,

     frenaque magnanimi dente teruntur equi;
et mihi cedet Amor, quamvis mea vulneret arcu
     pectora, iactatas excutiatque faces.
quo me fixit Amor, quo me violentius ussit,
     hoc melior facti vulneris ultor ero:

non ego, Phoebe, datas a te mihi mentiar artes,
     nec nos a?riae voce monemur avis,
nec mihi sunt visae Clio Cliusque sorores
     servanti pecudes vallibus, Ascra, tuis:
usus opus movet hoc: vati parete perito;
```

对于单个文档的提取需要使用 [[，当然既可以通过位置也可以通过名称：

```
> identical(ovid[[2]], ovid[["ovid_2.txt"]])
[1] TRUE
```

3.5 信息转化

一旦创建了语料库，后续文档修改则不可避免，比如填充、停止词去除。在 tm 包里，这些函数都归到信息转化里面，其主要函数就是 tm_map()，这个函数可以通过 maps 方式将转化函数实施到每一个语料上。

3.6 转化为纯文本

在 reuters 这个语料库中保存的是 XML 格式的文本，XML 格式文本没有分析的意义，我们只需要使用其中的文本内容。这个操作可以使用 as.PlainTextDocument() 函数来实现：

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！


```
reuters <- tm_map(reuters, as.PlainTextDocument)
```

注意，另外一种方式就是使用 readReut21578XMLasPlain 读取方式，那么在第一步即为纯文本格式。

3.6.1 去除多余的空白

通过：

```
reuters <- tm_map(reuters, stripWhitespace)
```

3.6.2 小写变化

通过：

```
> reuters <- tm_map(reuters, tolower)
```

更广泛的字符操作请参考 gsub。

3.6.3 停止词去除

通过：[1]

```
> reuters <- tm_map(reuters, removeWords, stopwords("english"))
```

3.6.4 填充

通过以下方式，需要 Snowball 包（并行计算）支持：

```
> tm_map(reuters, stemDocument)
A corpus with 20 text documents
```

3.7 过滤

有时候，我们需要选取给定条件下的文档。tm_filter 函数即为这个目所设计。sFilter 适应于一般情况下的用户自定义过滤情况：它整合了适用于元数据的的最小查询语句。假如需要找出 ID 等于 237，表头（heading）包含 “INDONESIA SEEN AT CROSSROADS OVER ECONOMIC CHANGE” 字符的文本本舰。

```
> query <- "id == '237' &
+      heading == 'INDONESIA SEEN AT CROSSROADS OVER ECONOMIC CHANGE'"
> tm_filter(reuters, FUN = sFilter, query)
A corpus with 1 text document
```

[1]比如 the, is, at, which 和 on 等不表意的词汇，tm 包提供了近五百个英文停止词，除英文外还支持法语、荷兰语、西班牙语、意大利语等十几种语言。关于停止词的定义参见 http://en.wikipedia.org/wiki/Stopwords

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！

也可以进行全文过滤

```
> tm_filter(reuters, pattern = "company")
A corpus with 5 text documents
> tm_filter(reuters, FUN = searchFullText, "company")
A corpus with 5 text documents
```

3.8 元数据管理

元数据是为了标记语料库的附加信息，最简单的使用方式就是调用 meta() 函数。文档会被预先被定义一些属性，比如作者信息，但也可能是任意自定义的元数据标签。这些附加的元数据标签都是独立的附加在单个文档上。从语料库的视角上看，这些元数据标签被独立的存储在每个文档上。除了 meta() 函数外，DublinCore() 函数提供了一套介于 Simple Dublin Core 元数据和 tm 元数据之间的映射机制，用于获得或设置文档的元数据信息。比如：

```
>     DublinCore(crude[[1]], tag = "creator") <- "Ano Nymous"
>     DublinCore(crude[[1]])
Simple Dublin Core meta data pairs are:
  Title       : DIAMOND SHAMROCK (DIA) CUTS CRUDE PRICES
  Creator     : Ano Nymous
  Subject     :
  Description:
  Publisher   :
  Contributor:
  Date        : 1987-02-26 17:00:56
  Type        :
  Format      :
  Identifier  : 127
  Source      :
  Language    : en
  Relation    :
  Coverage    :
  Rights      :
>     meta(crude[[1]])
Available meta data pairs are:
  Author        : Ano Nymous
  DateTimeStamp: 1987-02-26 17:00:56
  Description   :
  Heading       : DIAMOND SHAMROCK (DIA) CUTS CRUDE PRICES
  ID            : 127
```

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！


```
  Language     : en
  Origin       : Reuters-21578 XML
User-defined local meta data pairs are:
$TOPICS
[1] "YES"

$LEWISSPLIT
[1] "TRAIN"

$CGISPLIT
[1] "TRAINING-SET"

$OLDID
[1] "5670"

$Topics
[1] "crude"

$Places
[1] "usa"

$People
character(0)

$Orgs
character(0)

$Exchanges
character(0)
```

上面讲到的一些示例只是针对于文档级别的元数据管理。实际上在 tm 包元数据管理体系中，元数据标签对应了两个 level：

- 对于语料库（corpus）级别：文档的集合
- 单个文档的元数据

后一种元数据的使用主要是由于一些性能原因，或者是由于一些分析上的需要，比如要对文档进行分类（注意是 classification），分类的结果直接和每个文档的标记有关系。

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！


```
>  meta(crude, tag = "test", type = "corpus") <- "test meta"
> meta(crude, type = "corpus")
$create_date
[1] "2010-06-17 07:32:26 GMT"

$creator
   LOGNAME
"feinerer"

$test
[1] "test meta"

>  meta(crude, "foo") <- letters[1:20]
> meta(crude)
   MetaID foo
1       0   a
2       0   b
3       0   c
4       0   d
5       0   e
6       0   f
7       0   g
8       0   h
9       0   i
10      0   j
11      0   k
12      0   l
13      0   m
14      0   n
15      0   o
16      0   p
17      0   q
18      0   r
19      0   s
20      0   t
```

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！

3.9 标准操作和函数

对于语料库来说，其标准操作和函数和 R 的一般函数非常类似，比如

`[, [<-, [[, [[<-, c(), lapply()`

对于像`c()`这这个函数即是连接多个语料库的意思。连接之后，元数据信息也会被更新。

3.10 创建词条 -文档关系矩阵

文本挖掘的要创建词条 -文档关系矩阵，它是后续构建模型的基础。假设我们有两个文档分别是 text mining is funny 和 a text is a sequence of words，那么对应的矩阵为：

	a	funny	is	mining	of	sequence	text	words
1	0	1	1	1	0	0	1	0
2	2	0	1	0	1	1	1	1

在 tm 包里，根据词条、文档分别作为行、列或反之，对应有 TermDocumentMatrix 和 DocumentTermMatrix 两类稀疏矩阵，下面我们看一个例子：

```
> dtm <- DocumentTermMatrix(reuters)
> inspect(dtm[1:5,100:105])
A document-term matrix (5 documents, 6 terms)

Non-/sparse entries: 1/29
Sparsity           : 97%
Maximal term length: 10
Weighting          : term frequency (tf)

     Terms
Docs  abdul-aziz ability able abroad, abu accept
  127          0       0    0       0   0      0
  144          0       2    0       0   0      0
  191          0       0    0       0   0      0
  194          0       0    0       0   0      0
  211          0       0    0       0   0      0
```

3.11 对词条 -文档关系矩阵操作

实际上对于矩阵的操作 R 有大量的函数（比如聚类、分类算法等）支持，但这个包还是提供了一些常用的函数支持。假如需要找出发生 5 次以上的条目，可以使用 findFreqTerms() 函数：

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！


```
> findFreqTerms(dtm, 5)
 [1] "15.8"          "accord"        "agency"        "ali"
 [5] "analysts"      "arab"          "arabia"        "barrel."
 [9] "barrels"       "bpd"           "commitment"    "crude"
[13] "daily"         "dlrs"          "economic"      "emergency"
[17] "energy"        "exchange"      "exports"       "feb"
[21] "futures"       "government"    "gulf"          "help"
[25] "hold"          "international" "january"       "kuwait"
[29] "march"         "market"        "meeting"       "minister"
[33] "mln"           "month"         "nazer"         "nymex"
[37] "official"      "oil"           "opec"          "output"
[41] "pct"           "petroleum"     "plans"         "posted"
[45] "price"         "prices"        "prices."       "production"
[49] "quoted"        "recent"        "report"        "reserve"
[53] "reserves"      "reuter"        "saudi"         "sheikh"
[57] "sources"       "study"         "traders"       "united"
[61] "west"          "world"
```

或者找到相关性，比如对于 opec，找到相关系数在 0.8 以上的条目，使用 findAssocs():

```
> findAssocs(dtm, "opec", 0.8)
   opec prices.    15.8
   1.00    0.81    0.80
```

这个函数还可以接受一般的矩阵。对于一般矩阵，会将矩阵直接转化为相关阵，这种方式可以实现不同的相关计算方式。

词条 -文档关系矩阵一般都是非常庞大的数据集，因此这里提供了一种删减稀疏条目的方法，比如有些条目尽在很少的文档中出现。一般来说，这样做不会对矩阵的信息继承带来显著的影响。

```
> inspect(removeSparseTerms(dtm, 0.4))
A document-term matrix (20 documents, 3 terms)

Non-/sparse entries: 55/5
Sparsity           : 8%
Maximal term length: 6
Weighting          : term frequency (tf)

     Terms
Docs  march oil reuter
```

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！


```
  127     0   3       1
  144     0   4       1
  191     0   2       1
  194     0   1       1
  211     0   2       1
  236     2   6       1
  237     1   2       1
  242     1   3       1
  246     1   2       1
  248     1   8       1
  273     1   5       1
  349     1   4       1
  352     1   4       1
  353     1   4       1
  368     1   3       1
  489     1   5       1
  502     1   5       1
  543     1   3       1
  704     1   1       1
  708     1   2       1
```

这个函数去除了低于 40% 的稀疏条目项。

3.12 字典

字典是一个字符集合。经常用于在文本挖掘中展现相关的词条时。使用 Dictionary() 函数实现，可以看示例：

```
> (d <- Dictionary(c("prices", "crude", "oil")))
[1] "prices" "crude"  "oil"
attr(,"class")
[1] "Dictionary" "character"
```

当将字典传递到 DocumentTermMatrix() 以后，生成的矩阵会根据字典提取计算，而不是漫无目的地全部提取。

```
> inspect(DocumentTermMatrix(reuters, list(dictionary = d)))
A document-term matrix (20 documents, 3 terms)

Non-/sparse entries: 41/19
Sparsity           : 32%
```

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！


```
Maximal term length: 6
Weighting          : term frequency (tf)

     Terms
Docs  crude oil prices
  127     2   3      3
  144     0   4      2
  191     3   2      0
  194     4   1      0
  211     0   2      0
  236     1   6      2
  237     0   2      0
  242     0   3      1
  246     0   2      0
  248     0   8      5
  273     6   5      4
  349     2   4      0
  352     0   4      2
  353     2   4      1
  368     0   3      0
  489     0   5      2
  502     0   5      2
  543     3   3      3
  704     0   1      2
  708     1   2      0
```

4 网页解析的利器 –XML 包

本部分涉及 R 环境中 XML 包在处理网页数据的相关手段。

4.1 网页解析

在 R 中对网页解析（XML、HTML 文件，或包含 XML、HTML 的字符串）有多种方法，比较成熟的方法是使用 **XML** 包。这个包能够将 XML、HTML 网页树（tree）解析成 R 结构数据。当解析的内容已知为（潜在畸形的）HTML 时应当使用 `htmlTreeParse` 函数，这个函数拥有大量的参数来适应解析需要。它可以在 R 中或者使用内置 C-level 节点来创建树，两种方式在不同的环境下都非常有用。

`xmlParse` 、 `htmlParse` 分别等价于 `xmlTreeParse` 和 `htmlTreeParse` ，只是使用了一个默认的值代替了为 TRUE 的 useInternalNodes 参数。

欢迎加入QQ群（Use R！群）：538911602，交流学习R语言知识，群里免费提供300+本R语言书籍！

对于 xmlTreeParse 函数，我们先看一下它的用法：

```
xmlTreeParse(file, ignoreBlanks=TRUE, handlers=NULL, replaceEntities=FALSE,
             asText=FALSE, trim=TRUE, validate=FALSE, getDTD=TRUE,
             isURL=FALSE, asTree = FALSE, addAttributeNamespaces = FALSE,
             useInternalNodes = FALSE, isSchema = FALSE,
             fullNamespaceInfo = FALSE, encoding = character(),
             useDotNames = length(grep("^\\.", names(handlers))) > 0,
             xinclude = TRUE, addFinalizer = TRUE, error = xmlErrorCumulator())
```

这其中包含了非常多的参数，下面将一些比较重要的逐项地介绍：

file 包含 XML 内容的文件名。可以使用声明用户主工作目录，以可以是一个 URL（参考 isURL）。甚至文件还可以是一个压缩文件（gzip），文件可以被直接读取而不需要用户解压缩（gunzip）。

ignoreBlanks 逻辑型，在结果“树”中是否包含空白行。

handlers 选项函数，用于将不同的 XML 节点映射到 R 对象中。一般是函数的列表（list）。它提供了一种通过增加、减少节点来过滤树的方式。一般使用 C 代码来做处理。

replaceEntities 逻辑值，是否将单位的参考条目进行对于那个文本的替换。默认为 False。

asText 逻辑值，和第一个参数 file 相关。file 应被作为 XML 文本处理，而不是文件名称。这个选项可以允许还原不同的数据源（比如 HTTP、XML-RPC 等），并且依旧使用这个解析器。

trim 是否删除文本字符串中的空白。

validate 逻辑值，是否使用一个确认的解析器，或者根据 DTD 规范检查内容。TRUE 的话，DTD 的错误告警信息将被展示，但解析仍将继续，除非是致命的错误。

getDTD 逻辑值，DTD 是否被返回（包括内部和外部）。将改变返回的类型。

isURL 声明 file 参数是否涉及 URL（通过 ftp 和 http）或者系统上的常规文件。如果 asText 为 TRUE，则这个参数不应该被指定。这个函数试图从数据源中通过 grep 找到是否有 URL。libxml 解析器处理服务器连接，这并不是 R 的特性（比如 scan ）。

asTree 仅仅在 handlers 参数使用时有效

useInternalNodes 是否使用 XMLInternalNode 而不是用 XMLNode 类。内部节点（internal nodes）不转化到 R 时这个参数能使过程变得更快。

encoding 文档的编码方式。文档应该包含编码信息，但如果没有的话，这个参数可以为解析器指定文档的编码方式。

下面是关于使用 **XML** 包进行网页解析的一些例子。首先是网页源代码，可以通过 readLines ，我们的示例文件是这样的：

```
<?xml version="1.0"?>

<doc xmlns:xi="http://www.w3.org/2001/XInclude">

<caveat><para>This is a caveat that we repeat often.</para></caveat>

<section>
<title>A</title>
 <xi:include xpointer="xpointer(//caveat[1])"/>
</section>

<section>
<title>B</title>
<xi:include  xpointer="xpointer(/doc/caveat)"/>
</section>

</doc>
```

这个例子在 **XML** 包中 exampleData 目录下：

```
1  fileName <- system.file("exampleData", "include.xml", package="XML")
2  root <- xmlParse(fileName)
3  test <- getNodeSet(root, '//para')
```

那 test 的返回的结果是一个 list 结构：

```
> test
[[1]]
<para>This is a caveat that we repeat often.</para>

[[2]]
<para>This is a caveat that we repeat often.</para>

[[3]]
<para>This is a caveat that we repeat often.</para>
```

```
[[4]]
<para>This is a caveat that we repeat often.</para>

[[5]]
<para>This is a caveat that we repeat often.</para>

attr(,"class")
[1] "XMLNodeSet"
```

因为返回的是一个 list，所以可以使用 `lapply` 进行批量提取

`sapply(test, xmlValue)`

我们将 test 中 para 的内容提取出来，使用 `xmlValue` 函数：

```
1    vdata <- sapply(test, xmlValue)
```

使用 sapply 函数提取的的内容为：

```
> vdata
[1] "This is a caveat that we repeat often." "This is a caveat that we repeat often."
[3] "This is a caveat that we repeat often." "This is a caveat that we repeat often."
[5] "This is a caveat that we repeat often."
```

在上述的例子里有一个非常重要的函数 `getNodeSet` ，这个函数提供了使用 XPath 语法抽取 XML 节点信息的功能，当然 XPath 语法还需要一些知识，但资料也比较容易获得。比如对于这个文件的解析：

```
<doc>
<!-- A comment -->
 <a xmlns:omegahat="http://www.omegahat.org"
    xmlns:r="http://www.r-project.org">
  <b>
  <c>
    <b/>
  </c>
  </b>
  <b omegahat:status="foo">
   <r:d>
    <a status="xyz"/>
```

```
    <a/>
    <a status="1"/>
   </r:d>
  </b>
 </a>
</doc>
```

文件可以在 exampleData 目录下找到：

```
doc <- xmlParse(system.file("exampleData", "tagnames.xml", package = "XML"))
els <- getNodeSet(doc, "/doc//a[@status]")
sapply(els, function(el) xmlGetAttr(el, "status"))
```

首先解析一个 xml 文件，而后在 doc 的 XPath 目录下，找到含有 status 字段的 a 节点，可以看一下 els 是什么：

```
> els
[[1]]
<a status="xyz"/>

[[2]]
<a status="1"/>

attr(,"class")
[1] "XMLNodeSet"
```

最后通过 `xmlGetAttr` 函数获得 a 节点的属性值（Attribute Value）

```
[1] "xyz" "1"
```

4.2 字符集转化

如果处理的是中文字符，可能还会遇到字符编码转换的问题，可以使用 `iconv` 处理：

```
iconv(x, from = "", to = "", sub = NA, mark = TRUE, toRaw = FALSE)
```

5 XML 同 tm 包的配合使用（to do）

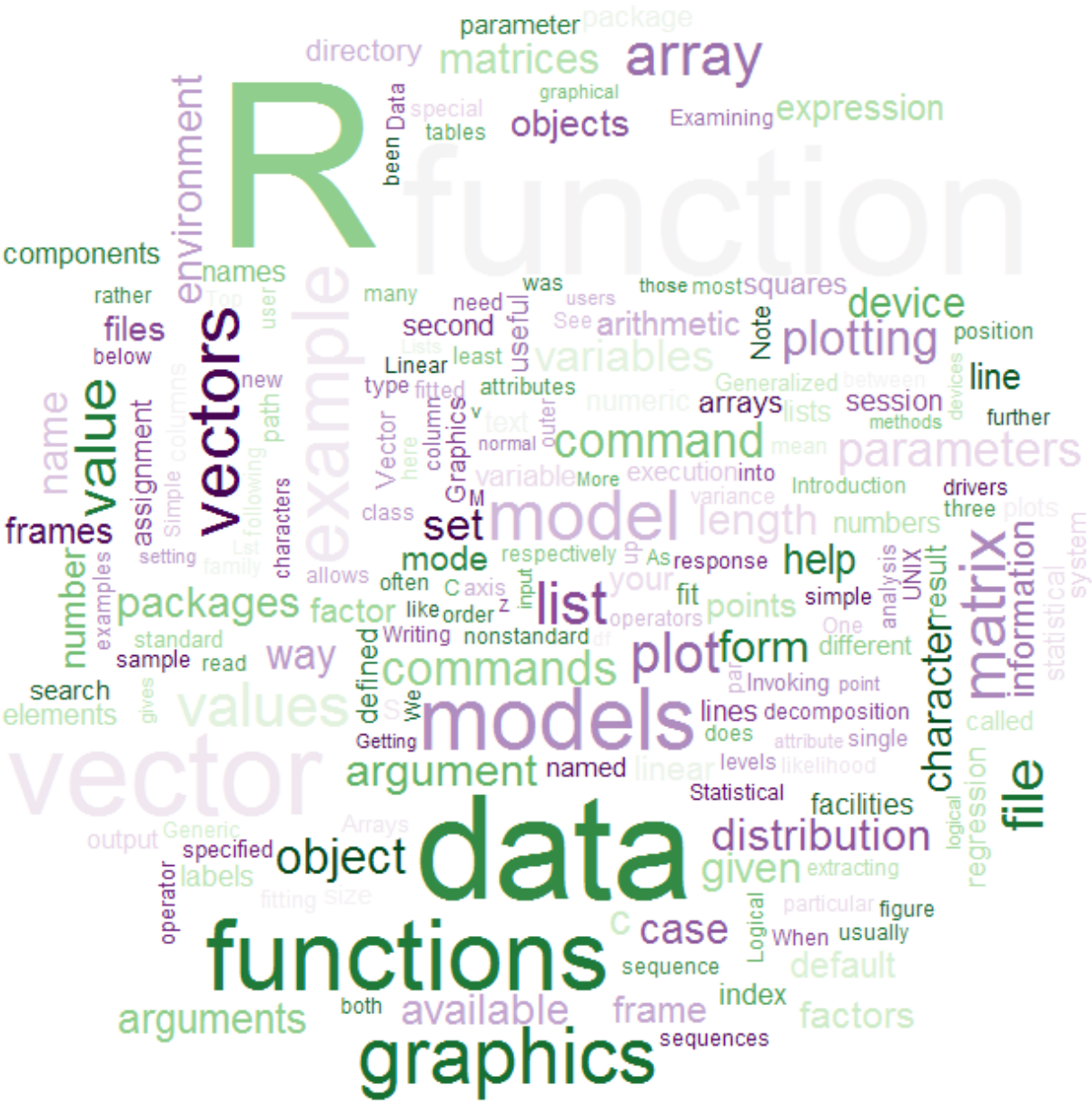
6 一些文本挖掘方面的应用

在得到 TermDocument 矩阵以后，基本上所有的数据挖掘算法都可以使用，如 Cluster、Classification、Regression 等，甚至 Apriori、SNA 等技术。

6.1 基础分析技术

在 **tm** 包中提供了诸如频数提取（ `findFreqTerms` ）、相关性提取（ `findAssocs` ）等函数，以及用于计算 cosine 距离的 `dissimilarity` 函数，这个函数在下面会被讨论到。

R 中的 **wordcloud** 包提供了绘制文本云的技术，甚至还可以对两个文本云进行比较。



6.1.1 文本聚类

聚类一般会考虑使用层次（Hierarchical）聚类法或分割（Partitioning）聚类，对于层次聚类的核心实际在距离阵的计算，一般会使用欧氏距离、闵氏距离等，但在大型数据条件下会优先选择 cosine 距离，及 `dissmilarity` 函数，直接上代码：

```
library(tm)
data(acq)
data(crude)
m <- c(acq, crude)
dtm <- DocumentTermMatrix(m)
dtm <- removeSparseTerms(dtm, 0.8)
inspect(dtm[1:5, 1:5])
dist_dtm <- dissimilarity(dtm, method = 'cosine')
hc <- hclust(dist_dtm, method = 'ave')
plot(hc, xlab = '')
```

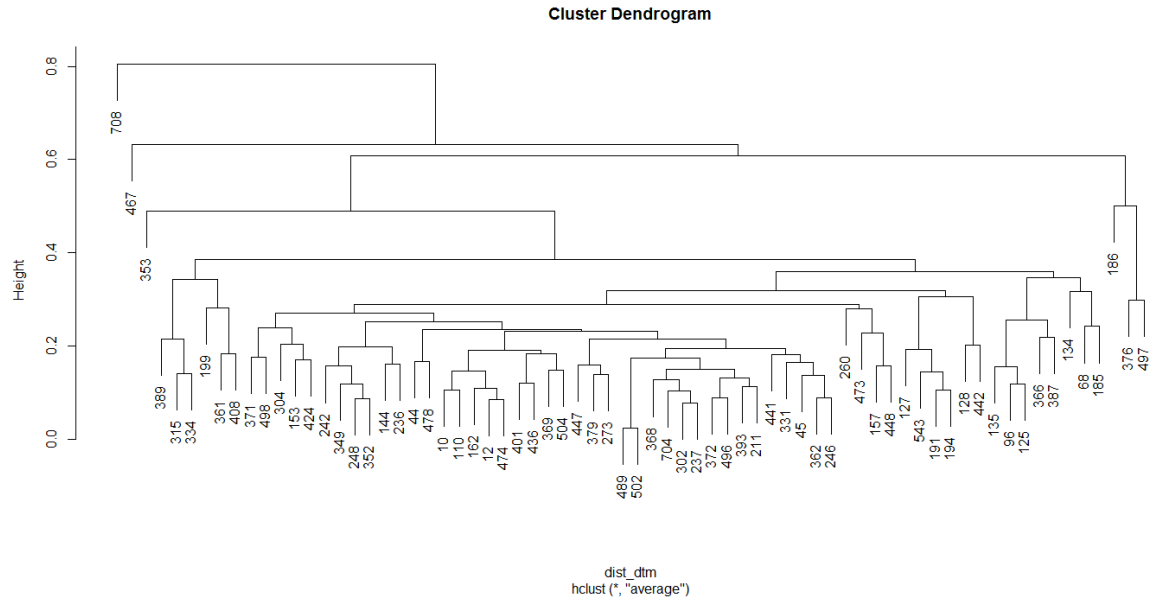


图 2: 使用 cosine 距离阵绘制的层次聚类图

有了距离阵后续可以做的事情非常多，如相关性可视化展示，或者 Multi dimensional Scaling 空间展示等。

或者使用 `kmeans` 函数构建 K 均值聚类模型。

6.1.2 文本分类

假如我们有表 1 这样的矩阵，那能做分类么？

答案是肯定的，这个矩阵即是用 document-term 矩阵加“文档标记”合并而成。一般的 Classification 类的方法都可以使用在这个矩阵上，比如从最简单的 knn，稍稍高级点的 regression、decision tree，甚至 SVMs 等方法都可以使用。这部分就不做过多叙述。

下面介绍一些不太为人熟知的方法：

表 1: 矩阵示例

Y	have	inc	its	last	mln	new	...
1	0.00	1.00	1.00	0.00	0.00	0.00	
1	0.00	0.00	0.00	0.00	1.00	1.00	
0	1.00	0.00	9.00	1.00	2.00	2.00	
0	0.00	0.00	2.00	0.00	1.00	1.00	
1	0.00	0.00	1.00	0.00	0.00	0.00	
0	0.00	1.00	1.00	0.00	1.00	0.00	
...							...

6.2 潜变量语义分析 (not done)

潜变量语义分析（Latent Semantic Analysis）是一种通过语料的统计计算，对语境的提取和抽象。LSA 和神经网络模型非常接近，但它基于奇异值分解（singular value decomposition）。

LSA 是一个提取语义的纯粹的数学（统计）方法，它不是传统的自然语言处理或人工智能，也没有使用任何人类主导的字典、语义网络、语法、词法，而仅仅是把文本解析为有意义的单次而已。它的基本步骤如下：

- 第一步是构建词条 -文档（文章）矩阵，即每一行代表了词条，每一类代表了文档（文章），每个单元代表了发生的频数；
- 接下来对这个矩阵实施奇异值分解。在奇异值分解中，原始矩阵被分解为三个矩阵，其中的一个矩阵描述了原始的行信息，作为正交基构成的向量；另外一个以相同的方式描述了列的信息；第三个是对角矩阵，包含了标度值（scaling values）。当这三个矩阵相乘时，可以得到原始矩阵。
- and

  to do

6.3 主题模型（Topic model）

在机器学习和自然与然处理中，主题模型是专门抽象一组文档所表达 “主题” 的统计技术。最早的模型是 probabilistic latent semantic indexing (PLSI)，后来 Latent Dirichlet allocation (LDA，潜在狄利克雷分配模型) 模型成为了最常见的主题模型，它可以认为是 PLSI 的泛化形式。LDA 主题模型涉及到贝叶斯理论、Dirichlet 分布、多项分布、图模型、变分推断、EM 算法、Gibbs 抽样等知识。

LDA 是一个三层的贝叶斯概率模型，且有一个假设：bag of word。意思就是认为文档就是一个词的集合，忽略任何语法或者出现顺序关系。不论 PLSI 还是 LDA，都遵循一个通式：

$$P(w|d) = \sum p(w|z) * p(z|d) \tag{1}$$

简单的说，即为两种分布：一种是 topic-word 分布，即 $p(w|z)$，一种是 doc-topic 分布，即 $p(z|d)$，而这两者构成了 word-topic-doc 的关系。

doc topic 这一级，PLSA 把这一级的所有变量都看作模型的参数，即有多少文档那么就有多少模型的参数；而 LDA 引入了一个超参数，对 doc topic 这一个层级进行 model。这样无论文档有多少，那么最外层模型显露出来的［对于 doc topic］就只有一个超参数。

最基本的 PLSA 和 LDA 在刻画 doc topic 和 topic word 都利用了一个模型，就是 multinomial model。为了计算的方便及先验的有意义，共轭先验是首选。multinomial distribution 的共轭分布是 Dirichlet distribution，很 nice 的一个分布。这也是 Latent Dirichlet Allocation 中 Dirichlet 的由来。

在 R 中可以使用 **topicmodels** 包来构建主题模型，下面我用一个例子简单的介绍一下关于这个包的使用。在各大互联网门户网站上收集了 720 篇文章，他们涉及了很多内容，而这些文章是分别属于不同主题的，比如有些可能是军事题材的、有些是社会题材的等等。

假如我按照 6 类进行划分，那么分成了这些主题（见表 2。请注意：主题的关键词是通过 LDA 模型计算，无人工干预）：

表 2: 预测种类及主题关键词

Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6
美食	旅游	拍摄	购物	汽车	招聘

每个主题下分别对应了 102, 140, 118, 110, 129, 121 篇文档。

实际上收集的这些文章是有类型标记的，他们分别属于” 购物类”, ” 旅游类”, ” 美食类”, ” 汽车类”, ” 数码类”, ” 招聘类”，共计六类。[2] 那我们看看实际人工的标记同建模后的标记的差异情况（见第 25 页的表 3）：

表 3: 预测种类同人工判别种类的混淆矩阵

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6
购物类	9	1	3	91	15	1
旅游类	0	113	0	4	0	3
美食类	90	24	0	5	1	0
汽车类	3	0	0	7	110	0
数码类	0	2	115	0	3	0
招聘类	0	0	0	3	0	117

可以看到实际的分类结果是十分接近人工分类结果的，而且非常 “智慧” 的提取出了几个主题的关键词，amazing!

[2]在使用 topic model 过程中设置了 6 类的参数是为了和实际情况进行比对

A 附录

A.1 关于 XML 文件

XML（eXtensive Markup Language）可扩展的标记语言，是万维网（World Wide Web Consortium W3C）定义的一种标准。用户可按照 XML 规则自定义标记（tags 标签）。

可作为跨平台的纯文本通讯数据，且无关语言。其设计目标核心是数据的内容，同显示分离。但 XML 不能直接用来写网页，即使包含了 XML 数据，依然要转化为 HTML 格式才能在浏览器上显示。简单的说：XML 被设计用来传输和存储数据。而 HTML 被设计用来显示数据。

- XML 指可扩展标记语言（EXtensible Markup Language）
- XML 是一种标记语言，很类似 HTML
- XML 的设计宗旨是传输数据，而非显示数据
- XML 标签没有被预定义。您需要自行定义标签。
- XML 被设计为具有自我描述性。
- XML 是 W3C 的推荐标准

在了解 XML 之前，需要对 HTML 文件有了解。XML 文件由声明、元素、属性、实体、注释构成。

一个简单是实例：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

第一行是 XML 声明，下一行描述文档的根元素（note，像在说：“本文档是一个便签”），接下来 4 行描述根的 4 个子元素（to, from, heading 以及 body），最后一行定义根元素的结尾（note）。

XML 和 HTML 文件一样，可以有属性值：

```
<note date="08/08/2008">
<to>George</to>
<from>John</from>
</note>
```

note 元素中的 date 属性是一个时间。
详细的 XML 文件学习可以参考这个专业网站 http://www.w3school.com.cn/x.asp

A.2 关于正则表达式

正则表达式就是用于描述这些规则的工具。换句话说，正则表达式就是记录文本规则的代码。正则并不是本文档的讨论重点，这里有一个 < 正则表达式 30 分钟入门教程 >，但需要提醒一下，30 分钟学会是幌子，正则表达式的学习是陡峭而漫长的。

参考文献

I. Feinerer. An introduction to text mining in R. *R News*, 8(2):19–22, Oct. 2008. URL http://CRAN.R-project.org/doc/Rnews/.

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

J. Li. *Rwordseg: Chinese word segmentation*, 2012. URL http://R-Forge.R-project.org/projects/rweibo/. R package version 0.0-4/r37.

H. Ronggui. *rmmseg4j: Chinese word segmentation based on mmseg4j*, 2011. URL http://R-Forge.R-project.org/projects/rqda/. R package version 0.1-0/r389.