Using Textual and Binary Formats for Storing Data in R-Programming

[1]N.V.Neeraja, [2]V.Rajya Lakshmi, [3]D.Sowjanya, [4]B.Rajesh

[1]Assistant Professor, [2]Assistant Professor, [3]Assistant Professor, [4]Assistant Professor
Computer Science & Engineering,
Mother Theresa Institute of Engineering and Technology, Palamaner, India

Abstract: **The aim of this study is to develop an answer for text mining scientific articles exploitation the R language within the "Knowledge Extraction and Machine Learning" course. Automatic text outline of papers could be a difficult downside whose approach would permit researchers to browse massive article collections and quickly read highlights and drill down for details. In proposed system, there are a variety of ways that data can be stored, including structured text files like CSV or tabdelimited, or more complex binary formats. However, there's associate degree intermediate format that's matter, but not as simple as something like CSV. The format is native to R and is somewhat legible as a result ofits matter nature. One will produce a additional descriptive illustration of associate degree R object by exploitation the dput() or dump() functions. The dump() and dput() functions are useful because the resulting textual format is editable, and in the case of corruption, potentially recoverable. By using dump() and dput(), we can easily mine the text. Unlike writing out a table or CSV file, dump() and dput() preserve the information (sacrificing some readability), so another user doesn't got to specify it everywhere once more.**

Index Terms **CSV, Knowledge Extraction, Machine Learning, Text Mining.**

INTRODUCTION:

The R Project:

- It is a background for statistical computing and graphics.
- It is a free software
- It is related with simple programming language.
- Analogous to S and S-plus.
- The R-programming software is available in www.r-project.org.
- Versions of R be present of Windows, MacOS, Linux and various other Unix platforms.
- R was primarily written by Ross Ihaka and Robert Gentleman, at the University of Auckland.
- It is an achievement of the S language, which was principally residential by John Chambers.

Compiled C vs Interpreted R:

- C requires an entire program to run.
- Program is decoded into machine code.
- Can then be executed iteratively.
- R can run interactively.
- Statements converted to machine instructions as they are come across.
- This is much more lithe, but also slower.

About R Programming Language:

- Interpreted language.
- To start, we will study.
- Syntax and common assemble.
- Function definitions.
- Commonly used functions.

R Function Libraries:

- Execute many common statistical procedures.
- Offer excellent graphics functionality.
- A suitable starting point for many data analysis projects.

Interactive R:

- R defaults to an interactive form.
- A prompt ">" is obtainable to users.
- Each input expression is evaluated…
- … and a outcome returned.

RELATED WORK:

Text categorization is a problem divided into eight steps.
1) Data Collection: In text categorization, the first step is collecting data. The sample data are texts that belong to a limited scientific field, i.e., "high production computing as support to medical image processing"
2) Text preprocessing: Text preprocessing is nothing but eliminate worthless information. It may include deletion of punctuation, stop words (any prepositions and pronouns), and numbers.
3) Data division: Next step divides the data into two component, training data and testing data. Based on training data, the classification algorithm will be skilled to produce a classification model. The testing data will be used to authenticate the performance of the resulting classification model. The text categorization experiments offered have been used 25% for guidance and 75% for testing. The classification performance is the average performance of realize classification models.
4) Feature extraction: Texts are characterized by features that:
a) Are not linked to the content of the text, such as author gender, author name, and others &
b) Reveal the text content, such as lexical items and grammatical grouping.
Considering single words, the simplest of lexical features, as a illustration feature in text categorization has proven effective for a number of function. The result of this step is a list of features and their consequent frequency in the training data set.
5) Feature selection: The result of the feature mining step is a long group of features, however, not all of these features are good for classification for many reasons: first, some classification algorithms are negatively exaggerated when using many features due to what is called "curse of dimensionality" next, the over-fitting problem may occur when the classification algorithm is trained in all features and finally some other features are familiar in most of the classes. To solve these troubles, many methods were proposed to opt for the most representative features for each class in the training data set.
6) Data representation: The outcome obtained from the previous step are represented in matrix format, and will be used by the classification algorithm. Generally, the data are in matrix format with n rows and m columns wherein the columns communicate to the selected feature and the rows correspond to the texts in the training data. Weighting methods, such as term frequency inverse document frequency (TFIDF) and term frequency (TF) are used to work out the value of each cell in this matrix, which symbolize the weight of the feature in the text.
7) Classifier training: The classification algorithm is trained using the training matrixes that enclose the selected features and their consequent weights in each text of the training data. Support Vector Machine (SVM) and Naive Bayes (NB) are the classical machine learning algorithms that have been the most utilize in text classification. The outcome is a classification model to be tested by means of the testing data.
8) Classification model evaluation: Evaluation procedure are assessing to estimate future performance by dealings such as accuracy, recall, precision, f-measure and to maximize empirical results.

EXPERIMENTS AND RESULTS:

We can protect the class of each column of a table or the levels of a factor variable. Textual formats can effort much better with version control programs like subversion or git which can only path changes meaningfully in text files. In addition, textual layout can be longer-lived; if there is sleaze somewhere in the file, it can be easier to fix the problem because one can just open the file in an editor and appear at it. Finally, textual formats stick on to the Unix philosophy, if that means whatever thing to you. There are a few downsides to using these in-between textual formats. The format is not very space capable, because all of the metadata is precise. Also, it is really only partly readable. In some case it might be preferable to have data stored in a CSV file and then have a separate code file that specifies the metadata.
Using dput() and dump()
One way to pass data around is by deparsing the R object with dput() and reading it back in (parsing it) using dget().

```
> ## Create a data frame
 > y <- data.frame(a = 1, b = "a")
 > ## Print 'dput' output to console
> dput(y) structure(list(a = 1, b = structure(1L, .Label = "a", class = "factor")), .Names\ = c("a", "b"), row.names = c(NA, -1L), class = "data.frame")
```

Observe that the dput() output is in the form of R code and that it conserve metadata like the class of the object, the row names, and the column names.
The outcome of dput() can also be saved directly to a file.

```
> ## Send 'dput' output to a file
 > dput(y, file = "y.R")
 > ## Read in 'dput' output from a file
 > new.y <- dget("y.R")
 > new.y a b
1 1 a
```

Several objects can be deparsed at once using the dump function and read back in using source.

```
> x <- "foo"
 > y <- data.frame(a = 1L, b = "a")
```

We can dump() R objects to a file by passing a character vector of their names.
> dump(c("x", "y"), file = "data.R")
> rm(x, y)
The inverse of dump() is source().
> source("data.R")
> str(y) 'data.frame' : 1 obs. of 2 variables:
$ a: int 1
$ b: Factor w/ 1 level "a": 1
> x [1] "foo"

3.1) Binary Formats

The harmonize to the textual format is the binary format, which is sometimes essential to use for efficiency purposes, or because there's just no functional way to represent data in a textual manner. Also, with numeric data, one can habitually lose precision when converting to and from a textual format, so it's enhanced to stick with a binary format. The key functions for translate R objects into a binary format are save(), save.image(), and serialize(). Individual R objects can be saved to a file using the save() function.

> a <- data.frame(x = rnorm(100), y = runif(100))
> b <- c(3, 4.4, 1 / 3)
> ## Save 'a' and 'b' to a file
> save(a, b, file = "mydata.rda")
> ## Load 'a' and 'b' into your workplace
> load("mydata.rda")

If you have a lot of objects that you want to save to a file, you can save all objects in your workplace using the save.image() function.

> ## Save everything to a file
> save.image(file = "mydata.RData")
> ## load all objects in this file
> load("mydata.RData")

Observe that we've used the .rda extension when using save() and the .RData extension when using save.image(). This is just my personal first choice; you can use whatever file extension you want. The save() and save.image() functions do not mind. However, .rda and .RData are reasonably common extensions and you may want to use them because they are recognized by other software. The serialize() function is used to change individual R objects into a binary format that can be communicated across an arbitrary link. This may get sent to a file, but it could get sent over a system. When you call serialize() on an R object, the outcome will be a raw vector coded in hexadecimal format.

> x <- list(1, 2, 3)
> serialize(x, NULL)
[1] 58 0a 00 00 00 02 00 03 02 01 00 02 03 00 00 00 00 13 00 00 00 03 00
[24] 00 00 0e 00 00 00 01 3f f0 00 00 00 00 00 00 00 00 00 0e 00 00 00 01
[47] 40 00 00 00 00 00 00 00 00 00 00 00 0e 00 00 00 01 40 08 00 00 00 00 00
[70] 00

If you want, this can be send to a file, but in that case you are better off using somewhat like save(). The advantage of the serialize() function is that it is the only way to perfectly represent an R object in an exportable format, without losing precision or any metadata. If that is what you require, then serialize() is the function for you.

CONCLUSION:

In this job we studied how to use R language to text mining, and the great importance of a good preprocessing of the data in order to obtain good quality results in the text classification step. In general, building an oversized quantity of labeled coaching information for text classification could be a effortful and long task.

That was the most drawback throughout the classifier development, and it is defined as future work. We found that the only document illustration was a minimum of pretty much as good as representations involving additional difficult syntactical and morphological analysis. Topic networks are also helpful in analyzing documents collections.

REFERENCES:

[1] Cooper, H.M.: The structure of knowledge synthesis, Knowledge in Society, vol. 1(1988)
[2] Edinger T FAU Edinger, T., Cohen AM FAU Cohen, A.M.: A large-scale analysis of the reasons given for excluding articles that are retrieved by literature search during systematic review. In: AMIA Annual Symposium Proceedings. pp. 379–387. No. 1942-597X (Electronic) (Nov 2013), http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3900186/
[3] Feinerer, I., Hornik, K., Meyer, D.: Text mining infrastructure in R. Journal of Statistical Software 25(5), 1–54 (3 2008), http://www.jstatsoft.org/v25/i05

[4] Hotho, A., N¨urnberger, A., Paab, G.: A brief survey of text mining. LDV Forum - GLDV Journal for Computational Linguistics and Language Technology (2005)
[5] Reed, C.: Latent Dirichlet Allocation: A Student Companion (2012)
[6] Weiss, S.M., Indurkhya, N., Zhang, T.: Fundamentals of Predictive Text Mining. No. e-ISBN 978-1-84996-226-1, Springer (2010)
[7] Weiss, S.M., Indurkhya, N., Zhang, T., Damerau, F.J.: Text Mining: Predictive Methods for Analyzing Unstructured Information. No. ISBN 0-387-95433-3, Springer (2005)
[8] Zhao, Y.: R and Data Mining: Examples and Case Studies. Academic Press (2013), http://www.sciencedirect.com/science/article/pii/B9780123969637000015
[9] Eddelbuettel, Dirk, Murray Stokely, and Jeroen Ooms. 2016. "RProtoBuf: Efficient Cross-Language Data Serialization in R." *Journal of Statistical Software* 71 (1): 1–24. doi:10.18637/jss.v071.i02.
[10]Carl Boettiger, Dirk Eddelbuettel with contributions by, Sebastian Gibb, Colin Gillespie, Jan Górecki, Matt Jones, Thomas Leeper, Steven Pav, and Jan Schulz. 2016. *Drat: Drat R Archive Template*. https://CRAN.R-project.org/package=drat.
[11] Chan, Chung-hong, and Thomas J. Leeper. 2016. *Rio: A Swiss-Army Knife for Data I/O*. https://CRAN.R-project.org/package=rio.