# 7. The Craft 2: Data Wrangling

The examples presented in this book use data sets that have previously been curated by others. These include data sets built into R, data sets in R packages, and data sets available on the web from sites such as `www.kaggle.com`. These ready-to-go data sets are used because our focus is on the algorithms, not the data wrangling. It is important to emphasize that data is not so easily obtained in the real world. The data gathering and cleaning phases in real-world machine learning projects can last months or even years. We can divide the data cleaning itself into two phases:

- data organization - compiling data into a form that can be read into an R data frame or other structure with functions such as read.csv() and read.table()
- data standardization - making sure that the values in the data frame make sense and are internally consistent

## 7.1 Data Organization

Raw data can be gathered from many sources: data bases, unstructured data bases, xml files, or even scraped from the web. Programs to gather the data and put it in an R-compatible form can be written within R but R is probably not the best language for text processing and we will not discuss it here. Rather, other common computer programming languages and libraries can be used for data gathering and organization.

### 7.1.1 Reproducible Research

It is critically important to document each step of your data gathering and organization. This is important for your own sake in case you need to redo some steps. It is also important if and when you want to publish your results. You will need to summarize how the data was collected and make detailed notes available for reviewers and fellow researchers. The most important reason to document is to create **reproducible research**. Anyone should be able to take your original data and follow your instructions to get the same results you reported.

## 7.2 Data Standardization

Once the data is gathered into a file in some regular format it can be read with read.csv or read.table. These functions read a file in table form and create a data frame. The assumption is that observations are organized into rows and columns represent attributes. You can use the help features of R to learn more about the read functions. You can use read.csv() like this:

```
df <- read.csv("data/myfile.csv")
```

There are numerous other arguments you can add. Here are a few of the most common ones:

- header=FALSE - use this if your file does not have a header row
- na.strings="NA" - use this to encode empty cells with NA
- stringsAsFactors = FALSE - use this if you want to apply it to all columns, otherwise let R do its thing and fix it later
- encoding="UTF-16"

R automatically creates dummy variables for factors with more than 2 levels. You can check the coding with contrasts() or levels(). If R didn't interpret a column the way you intended, you can change this with the as.factor(), as.integer(), etc. functions as needed. Don't assume anything! Check your data with R functions such as str(), summary() and head().

### 7.2.1 Dealing with NAs

Another problem is missing data, the NAs. First, you can check if NAs are going to be a problem by summing the number of NAs by column:

```
sapply(df, function(x) sum(is.na(x)==TRUE)
```

One option to get rid of NAs is to use the complete.cases() function. This will remove any row in which there is any NA in any column:

```
df <- complete.cases(df)
```

This may remove too much data. You should get rid of columns you don't care about before moving complete.cases() to limit how much data is lost. A less drastic option is to replace NAs with either the mean or median of the column. Here is a function to do this:

```
fix_NA <- function(x, mean_mode){
# sample call: df$x <- fix_NA(df$x, 1)
    if (mean_mode == 1) {  # use mean
        ifelse(!is.na(x), x, mean(x, na.rm=TRUE))
    } else {
        ifelse(!is.na(x), x, median(x, na.rm=TRUE))
    }
}
```

### 7.2.2 Outliers

The boxplot() function can be used to plot the variable and detect outliers. In a boxplot, the circles at either the top or bottom of the plot may be outliers. Care must be taken to not remove data just because it is inconvenient. Outliers could be data that is simply errors introduced at some point into the data. Determining whether data is truly an outlier or not takes some expertise beyond the scope of this book. The most important thing to keep in mind for the sake of reproducible research is to document every decision you make. If it is determined that a few values are truly outliers, they may be removed by replacing them with the mean or median values. Or, it may be best to remove that observation entirely by deleting the entire row. Again: document every decision made along with justifications that reflect your thinking at the time.

## 7.3 Time Data

In base R, dates are stored internally as the number of days since January 1, 1970. Dates prior to this will be negative numbers. We can use the as.Date() function to convert a character date into an internally stored date. Interestingly, base R doesn't have a native time class. If you need one, look at package `hms`.

Here are a few examples of processing dates and times in base R. Note the built-in as.Date() and Sys.Date() functions. We can do basic arithmetic on days as shown below, and use the difftime() function with units of secs, mins, hours, days, or weeks.

**Code 7.3.1 — Base R.** Times and Dates.

```
# processing dates
hire_date <- as.Date("2016-09-01")
days_employed <- Sys.Date() - hire_date
print(days_employed)  # Time difference of 623 days

# processing time
birth_date <- as.Date("1989-04-18")
difftime(Sys.Date(), birth_date, units="secs")
# Time difference of 917654400 secs
```

### 7.3.1 Lubridate

The lubridate package contains functions to simplify time and date processing. Lubridate is part of the tidyverse, discussed in the Appendices. An instant of time may be a date, or a time, or a date-time.

**Code 7.3.2 — Lubridate.** Basics.

```
library(lubridate)
today()   # example: "2018-05-17"
now()     # example: "2018-05-17 15:51:37 CDT"
```

Next we show a couple of things you can do with lubridate and a data set. The airquality data set has a month and a day column. The year is 1973 as stated in the description of the data set. We pasted these together and used the ymd() function to create a date column which we added to the data frame. The output of the last two instructions is shown as a comment after the instruction.

**Code 7.3.3 — Lubridate.** Airquality data.

```
df <- airquality[]
df$date <- ymd(paste("1973",airquality$Month,airquality$Day))
print(range(df$date))
# "1973-05-01" "1973-09-30"
df$date[nrow(df)] - df$date[1]
# Time difference of 152 days
```

Check out the `Vignette` for the package for more examples.

## 7.4 Text Data

Before we talk about text data, a caveat: good prediction results on text data are hard to get. Your first attempts are likely to be disappointing. For example, the online notebook for this section used Naive Bayes to classify Amazon reviews as positive and negative and achieved only 56% accuracy. Why is this so hard? Primarily because language is complex and ambiguous. Words have multiple meanings and it is difficult to train a machine to know the intended meaning, much less things like sarcasm. A fuller understanding of the field of Natural Language Processing (NLP) is required to do quality work in machine language with text data. That is beyond the scope of this book. However, for the curious we will discuss how text data can be prepared for machine learning algorithms. There are various ways to handle text data, and different packages. In this section, we will look at two packages: tm and RTextTools that can be used for text processing.

### 7.4.1 Text Mining Packages tm

The text mining package, tm, uses vocabulary from the field of natural language processing, so we need to discuss that as we go. In NLP, a *corpus* is a body of text. It can be a set of documents, a set of text messages, any set of text examples. The online notebook gives an example of creating a corpus from an Amazon revew data set using the Corpus() function. After the corpus is created, some data cleaning is done: putting all text in lowercase, removing numbers and punctuation, stripping white space, and removing stopwords which are common function words that don't carry content like *the, an, this, it, in,*, etc. Then a document-term matrix is created. This is a matrix in which each row represents a document in the corpus and each column represents a unique token (word) in the corpus. The cells hold word counts. Cell x, y holds the count of word y in document x. In the online notebook we see tht the corpus contains about 4K documents and over 21K unique words. The document term matrix will be sparse: most cells are 0 indicating that the majority of words do not occur in the document. Inspecting a portion of the document term matrix reveals this sparseness:

| Docs | bought | send | someone | spent | term | thing | unless | want | worst | written |
|---|---|---|---|---|---|---|---|---|---|---|
| 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 51 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 52 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 53 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 54 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 55 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

An alternative representation is to create a one-hot matrix. This involves replacing counts with 1 so that we have a binary present/not-present encoding of words in matrices.

As you can see in the online notebook, the reviews were divided into train and test sets and converted to one-hot matrix representations. The train matrix was used to train a naive Bayes algorithm and the test set was used to generate predictions. The naive Bayes algorithm had to learn probabilities for each word in the corpus. Below we show a couple of words. Most of the words had this kind of distribution, in that they provide as much evidence for a low rating as they do for a high rating. This approach was not sophisticated enough to learn anything from the words in the reviews. The tm package has some unique features that are interesting to experiment with. For example you can create a word cloud with selected words where the size of the word is based on the frequency of the word in the corpus.

```
> nb1$tables$worst
                      worst
factor(train_labels)            0            1
                   1 0.9976470588 0.0023529412
                   2 0.9992877493 0.0007122507

> nb1$tables$best
                      best
factor(train_labels)           0           1
                   1 0.998235294 0.001764706
                   2 0.997863248 0.002136752
```

### 7.4.2 RTextTools

Another online notebook provides examples of working with package RTextTools. This package was designed for non-technical people to do text analysis. Therefore it is simple to use but gives you less freedom to fine-tune the process. In the online notebook, the same Amazon reviews data is read in and a document-term matrix is created as in the previous section. The RTextTools package uses the tm package for this. Then a container is created. This container holds the train and test split as well as the labels and will be fed into the machine learning algorithms. The algorithms included in the package are svm, glmnet, maxent, slda, boosting, bagging, rf, nnet, and tree. In the online notebook we used svm, maxent and glmnet. After classifying on the test data, the summary analytics can be printed as shown below:

```
ENSEMBLE SUMMARY

       n-ENSEMBLE COVERAGE n-ENSEMBLE RECALL
n >= 1                1.00              0.83
n >= 2                1.00              0.83
n >= 3                0.78              0.89


ALGORITHM PERFORMANCE

       SVM_PRECISION           SVM_RECALL           SVM_FSCORE
               0.820                0.815                0.815
    GLMNET_PRECISION        GLMNET_RECALL        GLMNET_FSCORE
               0.815                0.810                0.810
MAXENTROPY_PRECISION    MAXENTROPY_RECALL    MAXENTROPY_FSCORE
               0.805                0.805                0.800
```

To explore this package further, check out `this article.`