Guide to the ngram Package

An n-gram Babbler

Drew Schmidt and Christian Heckendorf

Guide to the **ngram** Package

An n-gram Babbler

June 24, 2014

Drew Schmidt
wrathematics@gmail.com

Christian Heckendorf
heckendorfc@gmail.com

Version 1.1

Cover art is *Hydra*, uploaded to openclipart.org by Tavin.

This publication was typeset using LaTeX.

Contents

1 Introduction **1**

2 Installation **1**

2.1 Installing from Source 1

2.2 Installing from CRAN 2

3 Using the Package **2**

3.1 Background 2

3.2 Package Use and Example 2

3.3 Important Notes About the Internal Representation 4

1 Introduction

An n-gram is an ordered sequence of n "words" taken from a body of text. For example, consider the string `A B A C A B B`. This is the "blood code" for the video game Mortal Kombat for the Sega Genesis, but you can pretend it's a biological sequence or something boring if you prefer. If we examine the 2-grams (or bigrams) of this sequence, they are:

```
A B, B A, A C, C A, A B, B B
```

or without repetition:

```
A B, B A, A C, C A, B B
```

That is, we take the input string and group the "words" 2 at a time (because `n=2`). If we form all of the n-grams and record the next "words" for each n-gram (and their frequency), then we can generate new text which has the same statistical properties as the input.

The **ngram** package is an R package for constructing n-grams and generating new text as described above. It also contains a few preprocessing utilities to aid in this process. Additionally, the C code underlying this library can be compiled as a standalone shared library.

2 Installation

In this section, we will describe the various ways that one can install the **ngram** package.

2.1 Installing from Source

The sourcecode for this package is available (and actively maintained) on GitHub. To install an R package from source on Windows, you will need to first install the Rtools package. To install an R package from source on a Mac, you will need to install the latest Xcode, which you can get from the App store.

The easiest way to install **ngram** from GitHub is via the **devtools** package by Hadley Wickham. To install **ngram** using **devtools**, simply issue the command:

```
1 library(devtools)
2 install_github(repo="ngram", username="wrathematics")
```

from R. Alternatively, you could download the sourcecode from github, unzip this archive, and issue the command:

```
R CMD INSTALL ngram-master
```

from your shell.

2.2 Installing from CRAN

The usual

```
install.packages("ngram")
```

from an R session should do it.

3 Using the Package

3.1 Background

The input to the n-gram processor must be a single string (character vector of length 1). To aid in what could be a repetitive task, the package offers the `concat()` function. For example:

```
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
    "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
> library(ngram)
> concat(letters)
[1] "abcdefghijklmnopqrstuvwxyz"
> concat(concat(letters), letters, collapse=" ")
[1] "abcdefghijklmnopqrstuvwxyz a b c d e f g h i j k l m n o p
   q r s t u v w x y z"
```

So if data is coming from multiple files, the simplest way to merge them together would be to call

```
x <- readLines("file1")
y <- readLines("file2")

str <- concat(x, y)
```

The `ngram()` function (which does the processing and forms the n-grams) always splits words at a space. You can preprocess the string with R's regular expression utilities, such as `gsub()`, or use the `preprocess()` utility in the **ngram** package to be able to split at non-spaces for the purpose of n-gram generation (by inserting your own beforehand).

3.2 Package Use and Example

The general process goes

1. Prepare the input string; you may find `concat()` and `preprocess()` useful (see the previous subsection).

2. Process with `ngram()`.
3. Generate new text with `babble()`, and/or

3.5 Extract pieces of the processed ngram data with the `get.*()` functions.

Let us return to the example sequence of letters from Section 1. If we store this string in `x`:

```
x <- "A B A C A B B"
```

then the next step is to process with `ngram()`:

```
library(ngram)
ng <- ngram(x, n=2)
```

We can then inspect the sequence:

```
> ng
[1] "An ngram object with 5 2-grams"
```

If you don't have too many n-grams, you may want to print all of them by calling `print()` directly, with option `full=TRUE`:

```
> print(ng, full=TRUE)
C A
B {1} |

B A
C {1} |

B B
NULL {1} |

A C
A {1} |

A B
A {1} | B {1} |
```

Here we see each 3-gram, followed by its next possible "words" and each word's frequency of occurrence (occurrence following the given n-gram). So in the above, the first n-gram printed `C A` has `B` as a next possible word, because the sequence `C A` is only ever followed by the "word" `B` in the input string. On the other hand, `A B` is followed by `A` once and `B` once. The sequence `B B` is terminal, i.e. followed by nothing; we treat this case specially.

Next, we might want to generate some new strings. We for this, we use `babble()`:

```
> babble(ng, 10)
[1] "A C A B B B A C A B "
```

```
> babble(ng, 10)
[1] "B B C A B A C A B A "
> babble(ng, 10)
[1] "A C A B A C A B A C "
```

This generation includes a random process. For this, we developed our own implementation of MT19937, and so R's seed management does not apply. To specify your own seed, use the `seed=` argument:

```
> babble(ng, 10, seed=10)
[1] "A C A B A C A B B B "
> babble(ng, 10, seed=10)
[1] "A C A B A C A B B B "
> babble(ng, 10, seed=10)
[1] "A C A B A C A B B B "
```

3.3 Important Notes About the Internal Representation

The entirety of the interesting bits of the **ngram** package take place outside of R (completely in C). Observe:

```
> str(ng)
Formal class 'ngram' [package "ngram"] with 6 slots
  ..@ str_ptr:<externalptr>
  ..@ strlen : int 13
  ..@ n      : int 2
  ..@ ng_ptr :<externalptr>
  ..@ ngsize : int 5
  ..@ wl_ptr :<externalptr>
```

So everything is wrangled up top as an S4 class, and underneath the data is stored as 2 linked lists, outside the purview of R. This means that, for example, that you cannot save the n-gram object with a call to `save()`. If you do and you shut down and restart R, the pointers will no longer be valid.

Extracting a the data into a native R data structure is not currently possible. Full support is planned for a later release. Some pieces can be extracted. At this time, `get.ngrams()` and `get.string()` are implemented, but `get.nextwords()` is not.

```
> get.ngrams(ng)
[1] "C A" "B A" "B B" "A C" "A B"
> get.string(ng)
[1] "A B A C A B B"
> get.nextwords(ng)
Error in .local(ng, ...) : Not yet implemented
```