
Version
1.0



Guide to the ngram Package

An n-gram Babbling

Drew Schmidt

GUIDE TO THE ngram PACKAGE

AN N-GRAM BABBLER

JUNE 21, 2014

DREW SCHMIDT
WRATHEMATICS@GMAIL.COM



VERSION 1.0

© 2014 Drew Schmidt.

Permission is granted to make and distribute verbatim copies of this vignette and its source provided the copyright notice and this permission notice are preserved on all copies.

This manual may be incorrect or out-of-date. The authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

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

This publication was typeset using L^AT_EX.

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 an R package for constructing n-grams and generating new text as described above. It also contains a few 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 this (or any other) package from source on Windows, you will need to first install the [Rtools](#) package.

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

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

from an R session should do it.

3 Using the Package

3.1 Background

The input to the 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:

```
1 > letters
2 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
   "p" "q" "r" "s"
3 [20] "t" "u" "v" "w" "x" "y" "z"
4 > library(ngram)
5 > concat(letters)
6 [1] "abcdefghijklmnopqrstuvwxyz"
```

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

```
1 x <- readLines("file1")
2 y <- readLines("file2")
3
4 str <- concat(x, y)
```

Splitting of n-grams with the processor `ngram()` always occurs 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 modify this behavior to some degree.

3.2 Package Use and Example

The general process goes

1. Prepare the input string; you may find `concat()` and `preprocess()` useful.
2. Process with `ngram()`.
3. Generate nonsense 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`:

```
1 library(ngram)
2 x <- "A B A C A B B"
```

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

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

We can then inspect the sequence:

```
1 > ng
2 [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`:

```
1 > print(ng, full=TRUE)
2 C A
3 B {1} |
4
5 B A
6 C {1} |
7
8 B B
9 NULL {1} |
10
11 A C
12 A {1} |
13
14 A B
15 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()`:

```
1 > babble(ng, 10)
2 [1] "A C A B B B A C A B "
3 > babble(ng, 10)
4 [1] "B B C A B A C A B A "
5 > babble(ng, 10)
6 [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:

```
1 > babble(ng, 10, seed=10)
2 [1] "A C A B A C A B B B "
3 > babble(ng, 10, seed=10)
4 [1] "A C A B A C A B B B "
5 > babble(ng, 10, seed=10)
6 [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:

```
1 > str(ng)
2 Formal class 'ngram' [package "ngram"] with 6 slots
3 ..@ str_ptr:<externalptr>
4 ..@ strlen : int 13
5 ..@ n      : int 2
6 ..@ ng_ptr :<externalptr>
7 ..@ ngsz  : int 5
8 ..@ 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.