Using C with R

William Wisler
wislerw@gmail.com
2013-May-31

Initial Problem

- Identifying product descriptions that are similar

  Is:

  LN PROJECTS 10OZ 24PK

  the same product as:

  LN PROJECTS 10 OZ

N-Gram Plagiarism Detection

- Comparing a text to a known corpus

$$nd_1(A,B) = \frac{\sum_{g \in P(A)} \left(\frac{2(f_A(g) - f_B(g))}{f_A(g) + f_B(g)} \right)^2}{4|P(A)|}$$

- Intrinsic Plagiarism Detection Using Character n-gram Profiles

  Efstathios Stamatatos

  University of the Aegean

  83200 - Karlovassi, Samos, Greece

  http://www.icsd.aegean.gr/lecturers/stamatatos/papers/PAN2009.pdf

R Code for N-Gram Count

```r
# calculate the different NGrams and the score each one has
# functun requires a string to parse and a length of the gram

NGram <- function(str.to.parse, n = 3){
  #setup the output
  rslt <- data.frame(gram = character(0), gram.ct = integer(0)
                     , stringsAsFactors = FALSE)
  # get rid of uppercase; whitespace; and numbers
  str.to.parse <- tolower(gsub("\\W|\\d", "", str.to.parse))

  # the string must still have a postitive length after being sanitized
  if(length(str.to.parse) > 0){
    # iterate through each charater in the string
    for(i in n:nchar(str.to.parse)){
      # grab a substring of gram length
      sub.str <- substr(str.to.parse, i - n + 1, i)

      #check if we have encountered the string before
      if(sub.str %in% rslt$gram){
        # if so add to it's counter
        rslt[rslt$gram == sub.str,"gram.ct"] <-
          as.integer(rslt[rslt$gram == sub.str,"gram.ct"]) + 1
      }else{
        # add the new string to the dictionary
        rslt[dim(rslt)[1]+1,] <- cbind(sub.str, 1)
      }
    }
  }
  # after all have been countered compute the frequency for the column
  rslt$gram.ct <- as.integer(rslt$gram.ct)
  rslt$freq <- rslt$gram.ct / sum(rslt$gram.ct)
  return(rslt)
}
```

R Code for Distance Measure

```r
# calculate the difference between two sets of n-grams
# function takes two string and how long the count should be
NGramDist <- function(str.a, str.b, n = 3){
  # calculate the gram probabilities for each
  set1 <- NGram(str.a, n)
  set2 <- NGram(str.b, n)
  d <- 0

  # for each unique gram in phrase 1
  for(i in 1:dim(set1)[1]){
    current.gram <- set1[i,"gram"]
    # find the frequence of the gram in each word set
    freq.a.g <- set1[i,"freq"]
    freq.b.g <- set2[set2$gram == current.gram, "freq"]
    # if it is missing from word b then use 0
    if(length(freq.b.g) == 0 ){
      freq.b.g <- 0
    }
    # calculate a measure for this gram based on the frequencies
    # add the gram measure to a running total for the word
    d <- d + (
      (2*(freq.a.g - freq.b.g) / (freq.a.g + freq.b.g)) ^ 2
    )
  }

  # calc the final score based on how many (non-unique+unique) grams existed
  return(d / (4 * sum(set1$gram.ct)))
}
```

- Takes ~53sec for 100 Descriptions
- Scales exponentially
  - O(n^2)

C Code for R interface - Entry

```c
#include  <R.h>
#include  <Rinternals.h>
#include  <Rmath.h>
#include  <Rdefines.h>
```

```c
// calculate the difference between two sets of n-grams
// function takes two string and how long the count should be
SEXP NGramLetters(SEXP strA, SEXP strB, SEXP N){
  // convert our parameters to a strings and an integer
  // c, unlike r, must have aset datatype for all parameters
  const char *myStrA = CHAR(STRING_ELT(strA,0));
  const char *myStrB = CHAR(STRING_ELT(strB,0));
  const int n = INTEGER_VALUE(N);

  // setup our datastructers for the results and itermediate steps
  Tnode *NGramA = NULL;
  Tnode *NGramB = NULL;

  int gram_countA, gram_countB;
  SEXP result;  // variable will contain the final score
```

C Code for R interface - Return

```
  PROTECT(result = NEW_NUMERIC(1)); // make sure R isn't accessing our return variable
  REAL(result)[0] = score;          // declare the result to be a real number(numeric)
  UNPROTECT(1);                     // allow r to access the result

  return (result);
}
```

Compile the Code

- Must be compiled from R
- R CMD SHLIB <filename.c>

```
commandermonkey@m:~/depot/packsize$ R CMD SHLIB NGram.c
gcc -std=gnu99 -I/usr/share/R/include -DNDEBUG      -fpic  -O2 -pipe -g  -c NGra
m.c -o NGram.o
gcc -std=gnu99 -shared -o NGram.so NGram.o -L/usr/lib/R/lib -lR
```

Load the Library & Call Function

- dyn.load(“<filename.so>”)
- .Call(“<function name>”, ...)

```
dyn.load("NGram.so")

str.a <- tolower("LN PROJECTS 10OZ 24PK")
str.b <- tolower("LN PROJECTS 10 OZ")

.Call("NGramLetters", str.a, str.b)
```

Performance

- 171 times faster
- Able to run 100x C the comparisons in 59% of the time of native R

```
> source("nGram.R")
[1] "Run time for 100 items (4900 calls) native R:"
   user  system elapsed
 48.057   0.005  48.075
[1] "Run time for 100 items (4900 calls) C code:"
   user  system elapsed
  0.280   0.000   0.281
[1] "Run time for 1000 items (499000 calls) C code:"
   user  system elapsed
 28.384   0.001  28.391
```

Further Speedup

- Pass entire vector rather than individual pairs
  - Increase memory footprint
  - Dramatically shorten run time since you only have to compute each gram list once for a word

Further Reading

- https://github.com/hadley/devtools/wiki/c-interface
- http://www.biostat.jhsph.edu/~rpeng/docs/interface.pdf