Earnings Calls Analysis in R - Part2

Dave Hurst

Friday, November 21, 2014

This article is a continuation of the an exercise to explore a corpus of Earnings Calls transcripts using R. In Part1 I explored some basic text exploratory analysis and built a dictionary and merged a few synonyms. I concentrated on single word terms, but the limitation was pretty obvious, so in this exercise I'll look at multi-word concepts. The corpus is made up of transcripts for the primary two hard disk manufacturers, Seagate (STX) and Western Digital (WDC).

Start by loading the R packages, read the files into a corpus and repeat the transformations we did for Part1 to keep things consistent:

```
library(tm)
library(stringr)
library(RWeka)
```

```
##  [1] "STQ 2013Q4 Earnings Q&A.txt" "STQ 2013Q4 Earnings.txt"
##  [3] "STQ 2014Q4 Earnings Q&A.txt" "STQ 2014Q4 Earnings.txt"
##  [5] "STQ 2015Q1 Earnings Q&A.txt" "STQ 2015Q1 Earnings.txt"
##  [7] "WDC 2013Q4 Earnings Q&A.txt" "WDC 2013Q4 Earnings.txt"
##  [9] "WDC 2014Q4 Earnings Q&A.txt" "WDC 2014Q4 Earnings.txt"
## [11] "WDC 2015Q1 Earnings Q&A.txt" "WDC 2015Q1 Earnings.txt"
```

In order to look for words we can combine into a term (or "concept" in SPSS-speak), I decided to explore using N-grams. N-grams are simply word combinations found in the corpus. For example, the following sentence has 6 terms or 1-grams: "I love watching soccer on Saturday". There are 5 2-grams: "I love", "love watching", "watching soccer", "soccer on", "on Saturday", although you'll often see the beginning and end of sentences added as tokens to get the additional 2-grams "<BEGIN> I" and "Saturday <END>".

N-grams are amazingly powerful, and widely used for predicting word probabilities, but I won't go into all their uses here. I'm strictly using them to figure out which words I should combine into a single concept.

We'll need a separate TermDocumentMatrix for each N-gram, but the Rweka library makes this easy work:

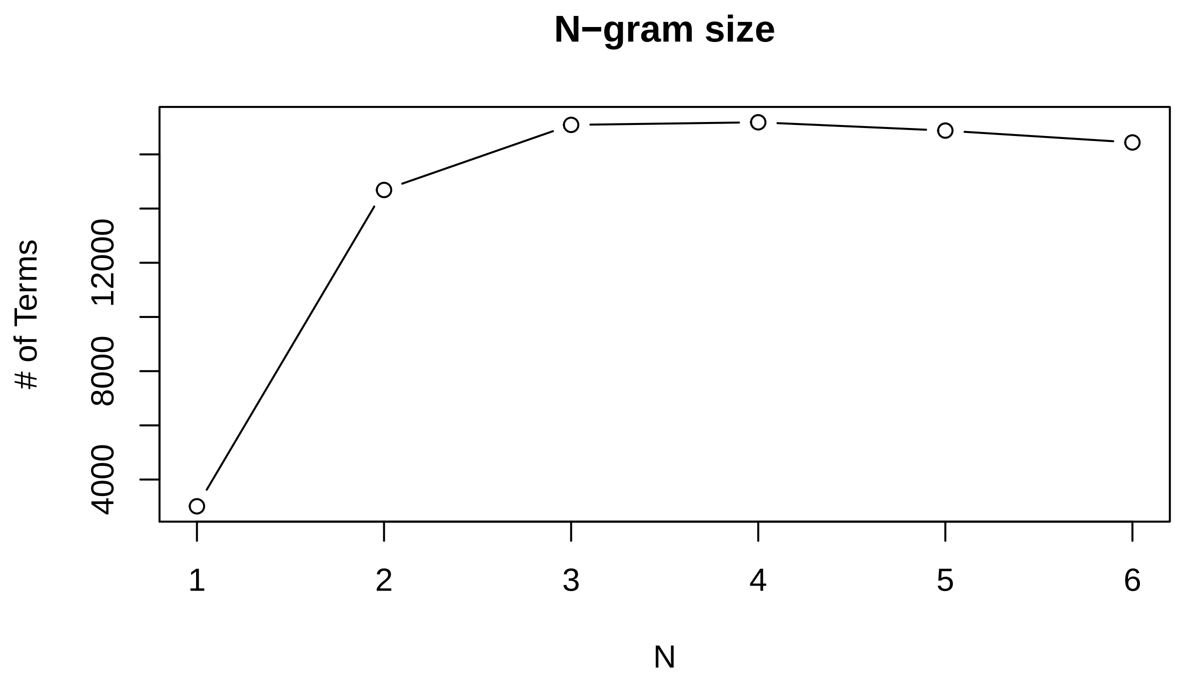
```
tdm.1 <- TermDocumentMatrix(docs)  #1-grams are just a regular TDM

makeNGram
```

```
## function( corpus, n ) {
##   nTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = n, max = n))
##   x.tdm <- TermDocumentMatrix(corpus, control = list(tokenize = nTokenizer))
##   x.tdm
## }
```

```
tdm.2 <- makeNGram( docs, 2)
tdm.3 <- makeNGram( docs, 3)
tdm.4 <- makeNGram( docs, 4)
tdm.5 <- makeNGram( docs, 5)
tdm.6 <- makeNGram( docs, 6)

tcnt <- c( tdm.1$nrow, tdm.2$nrow, tdm.3$nrow, tdm.4$nrow, tdm.5$nrow , tdm.6$nrow)
plot (tcnt, type="b", ylab = "# of Terms", xlab="N", main="N-gram size")
```

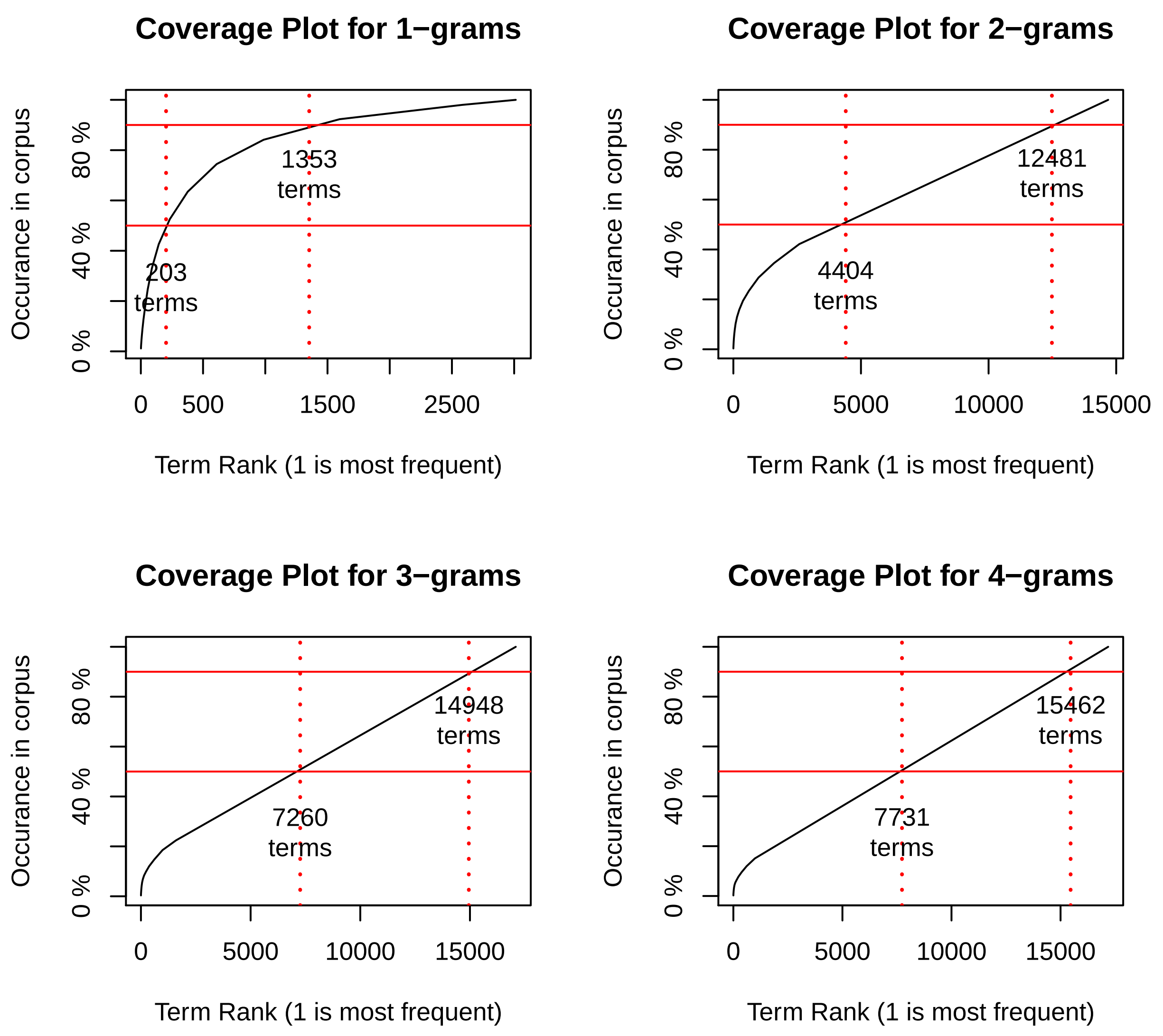


(Note that I've switched from DTM's in Part1 to TDM's here. They are just transposes of each other, so I haven't found a technical reason to favor one over the other. I prefer long narrow tables, so since I only have a dozen documents, I'll go with terms as rows.)

There are 3013 1-gram terms in the corpus and abot 41,000 words in total, so on average each 1-gram term occurs 13.6 times, althogh that's a poor estimate since the frequency distribution is heavily skewed. That skew turns out to be very useful since it means we can focus on treatments for the most frequent terms and have a large effect on the overall analysis.

By contrast the 4-grams have 17184 terms, a ratio of 2.4 words/term, since 4-grams are not as prevalent in the document. Where the higher level N-grams are repeated though, we can divine good information. We can see how much leverage the most frequent terms have by plotting them against the number of words they cover in the document.

```
par(mfrow=c(2,2))
plotCoverage (tdm.1, 1)
plotCoverage (tdm.2, 2)
plotCoverage (tdm.3, 3)
plotCoverage (tdm.4, 4)
```

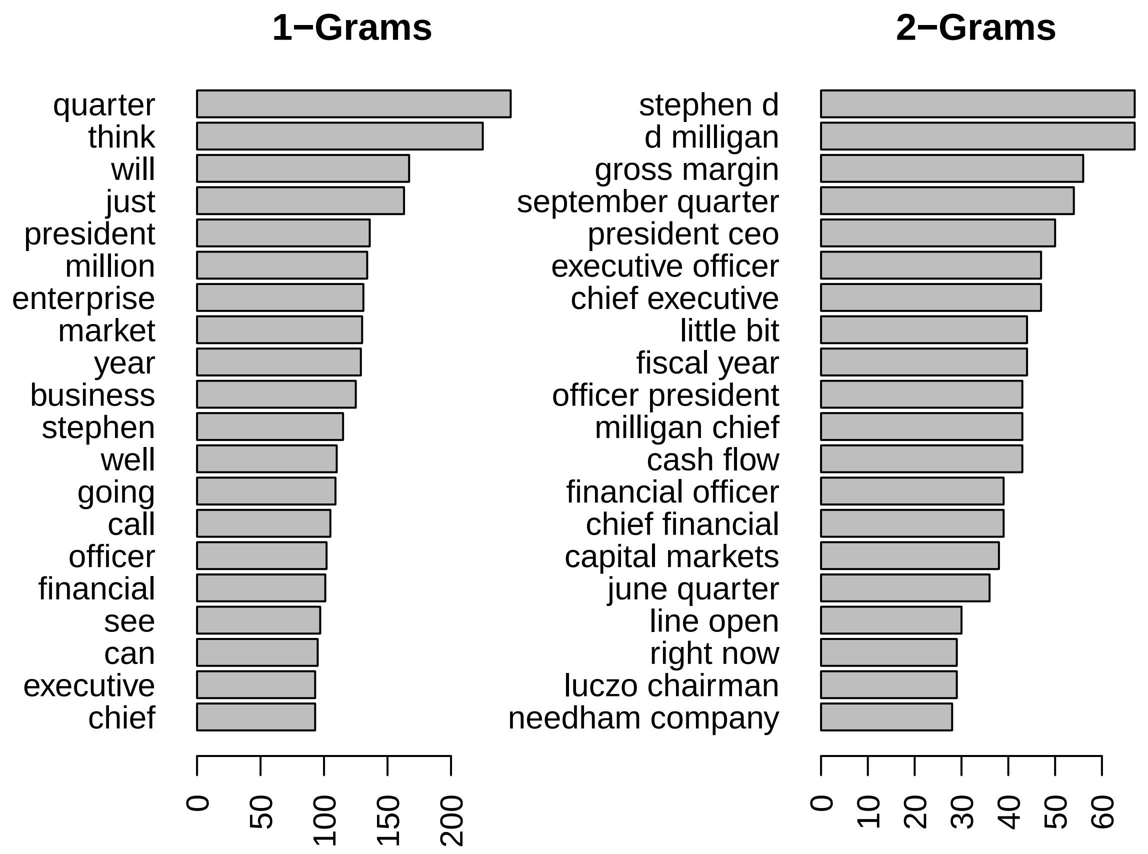


```
par(mfrow=c(1,1))
```

The higher order N-grams go linear pretty quickly, which is an indicator that they only occur once in the document, and don't hold a lot of informational value. One problem with N-grams is that they consume a lot of memory, so based on these plots we could discard most of them in order to concentrate on more frequent occurances.

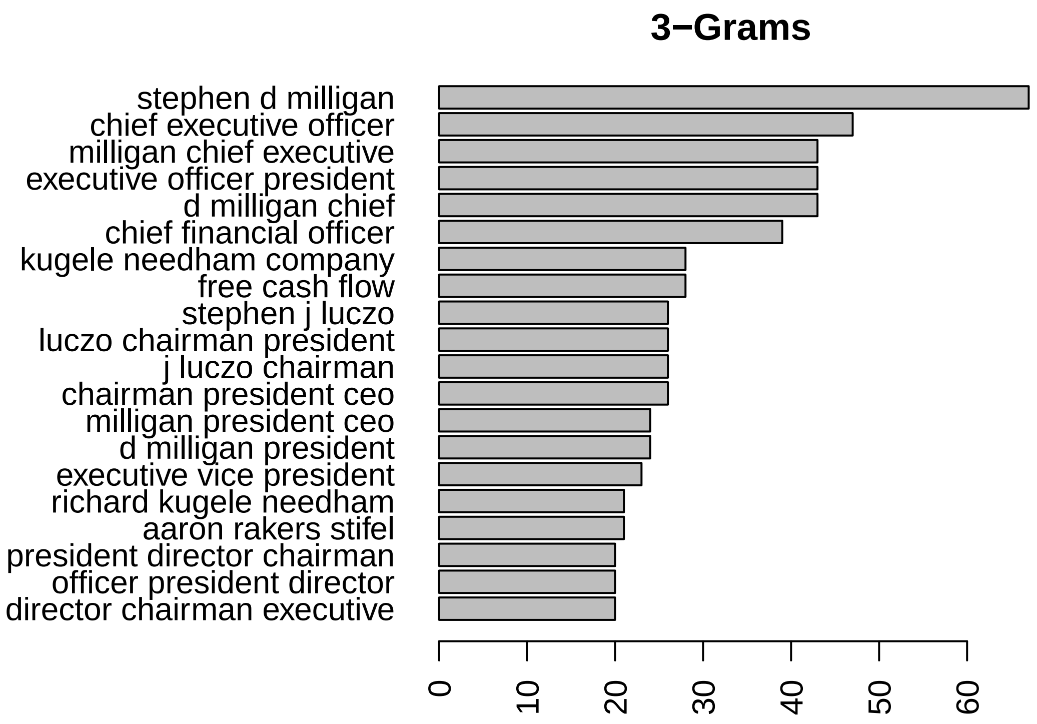
Let's look at the top 20 terms for each N-gram

```
mar.orig <- par()$mar
par(las=2, mar=c(5,6,2,2), mfrow=c(1,2))
barplot( mostFrequentTerms(tdm.1, 20), horiz=TRUE, main="1-Grams" )
barplot( mostFrequentTerms(tdm.2, 20), horiz=TRUE, main="2-Grams"    )
```

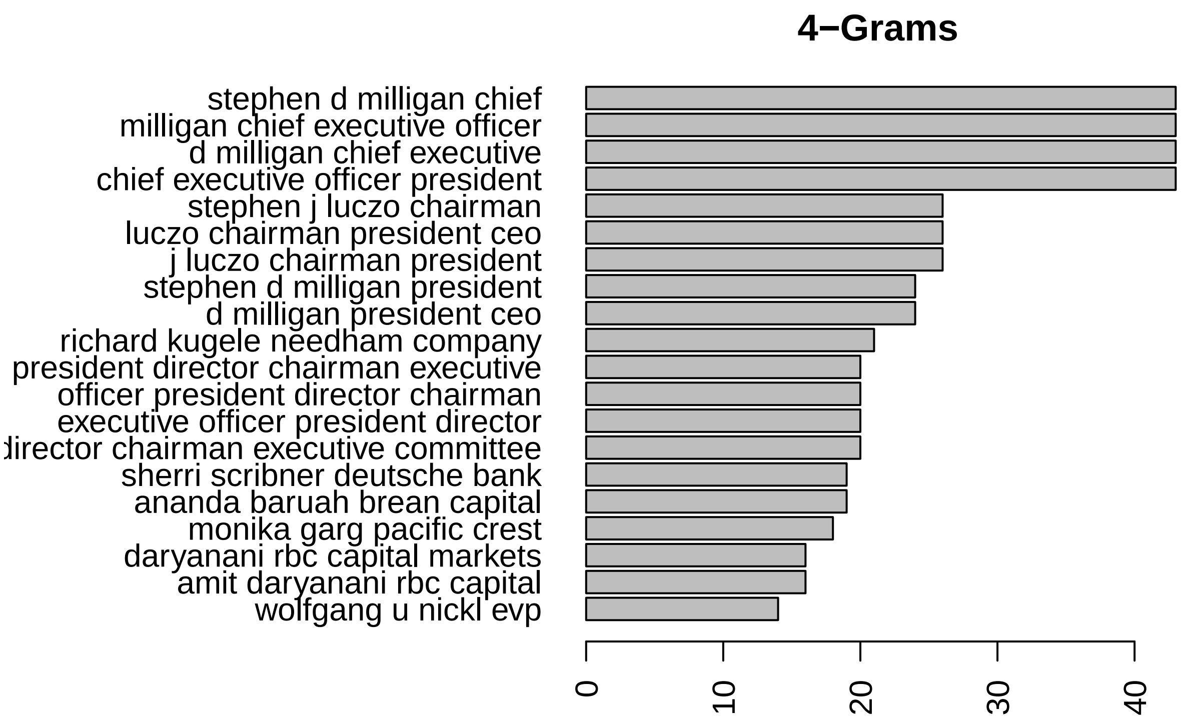


1-grams appear almost random, and without more context are not very interesting. 2-grams very quicly begin to show some structure. We see for instance that we are talking about the September and June quarters, "cash flow" and "fiscal year" reveal words we can combine into concepts, but mainly we see that the people and roles are dominant in the text.

```
par(las=2, mar=c(3,15,2,2), mfrow=c(1,1))
barplot( mostFrequentTerms(tdm.3, 20), horiz=TRUE, main="3-Grams"  )
```

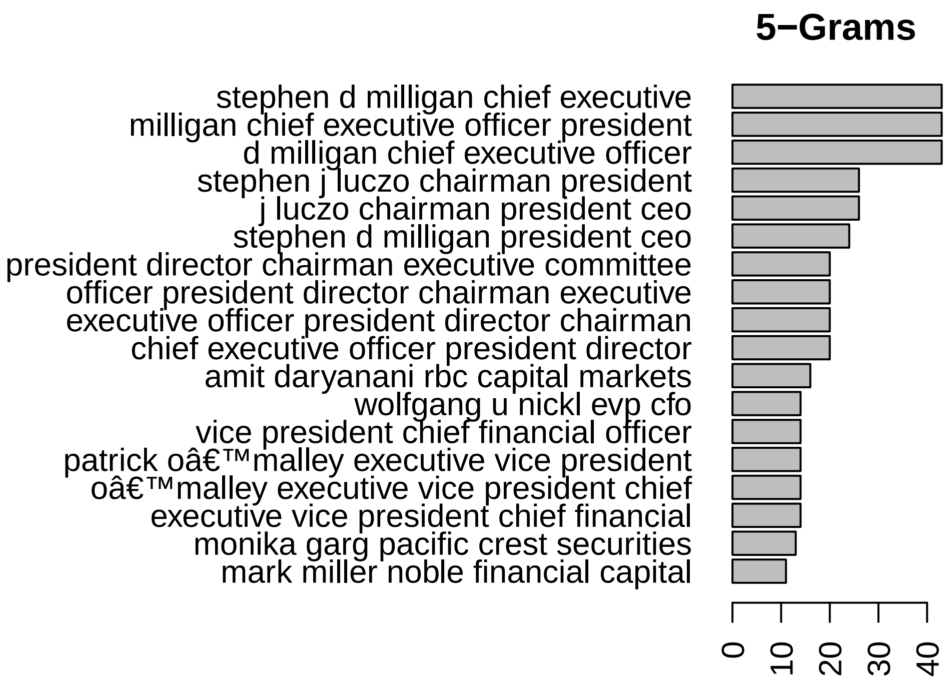


```r
barplot( mostFrequentTerms(tdm.4, 20), horiz=TRUE , main="4-Grams" )
```

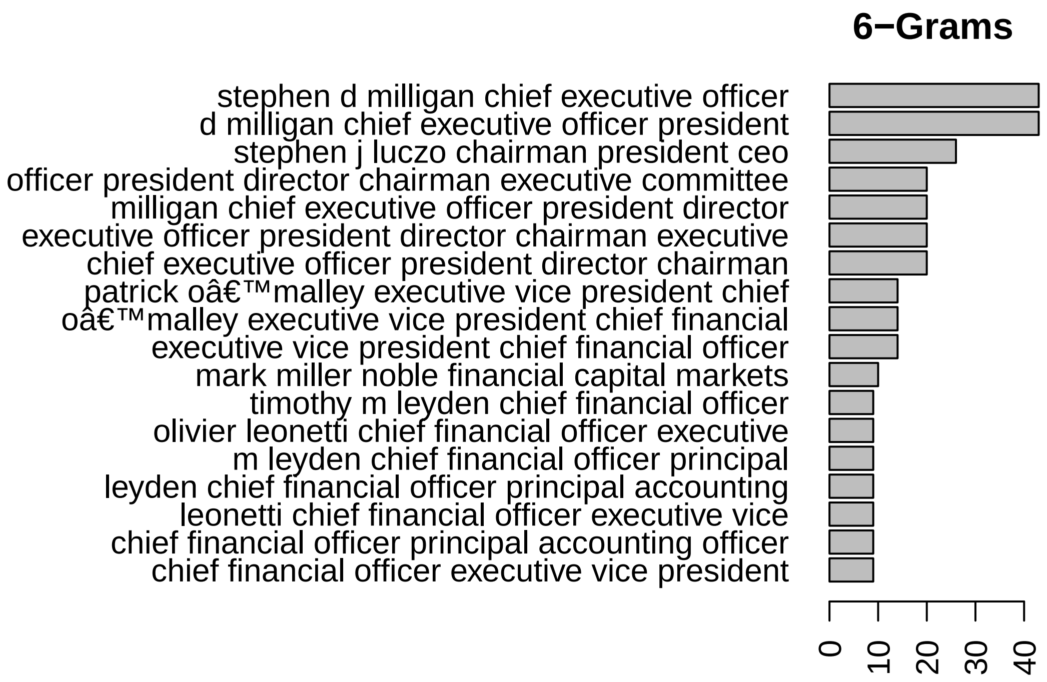


3-grams get really interesting and start revealing the call participants with more clarity. 4-grams are similar to 3-grams but we can start to see both names and roles, however 5- and 6-grams start to turn into gibberish.

```r
par(las=2, mar=c(3,25,2,2))
barplot( mostFrequentTerms(tdm.5, 18), horiz=TRUE , main="5-Grams" )
```



```r
barplot( mostFrequentTerms(tdm.6, 18), horiz=TRUE , main="6-Grams" )
```



```
par(las=0, mar=mar.orig)
```

I was excited to see the names starting to reveal themselves, since I want to get to text link analysis, and would like to have some way of identifying the most important people and concepts as a precursor to that. For that reason the concepts that seemed like the best place to start were the executives.

We can use the 6-grams to show us the context in which WD CEO Steve Milligan is mentioned:

```
head(findTerms (tdm.6, "(ste.*){0,1}milligan"))
```

```
##   break ceo milligan discussed stec acquisition
##                                               1
##  call milligan unwilling answer attibuting size
##                                               1
##   call steve milligan president chief executive
##                                               3
##              call steve milligan tim leyden will
##                                               1
##  ceo milligan discussed stec acquisition commit
##                                               1
## ceo milligan stated â€œthere plenty opportunity
##                                               1
```

Using the ‘toConcept’ function, we combine all the different references into a single concept. I’ve repeated this for CEO Steve Luczo as well as some of the other roles.

```
toConcept
```

```
## function( x, pattern, concept) {
##   tm_map(x, content_transformer(function(x, p, str) gsub(p, str, x)), pattern, concept)
## }
```

```
docs.pre_concept <- docs
docs <- toConcept (docs, "(ste.*){0,1}milligan", "<MILLIGAN>")
docs <- toConcept (docs, "(ste.*){0,1}luczo", "<LUCZO>")
docs <- toConcept (docs, "chief executive officer|ceo", "<CEO>")
docs <- toConcept (docs, "chief financial officer|cfo", "<CEO>")
docs <- toConcept (docs, "executive vice president|evp", "<EVP>")
docs <- toConcept (docs, "vice president|vp", "<VP>")  #must come after EVP
```

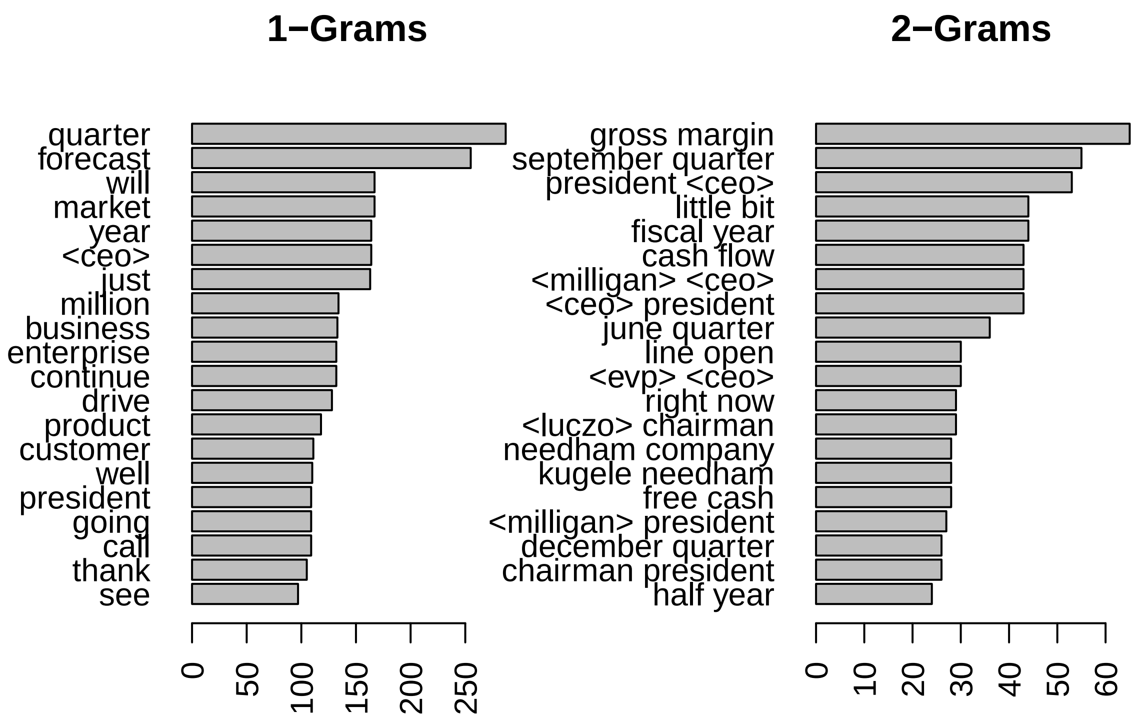
I’ll also collapse some of the synonyms as I did in Part 1 and regenerate the TDM files so that we can take a look at the new N-grams.

```
infile <- "synonym.csv"
synonyms <- read.csv(infile, stringsAsFactors=FALSE)
docs <- replaceSynonyms(docs, synonyms)

psyn <- findTerms(tdm.1, "predict|forecast|think")
psyn.words <- names(psyn)[- grep("predictable", names(psyn))]  #remove "predictable"
docs <- replaceSynonyms(docs, data.frame(matrix( psyn.words, nrow=1 )))
```

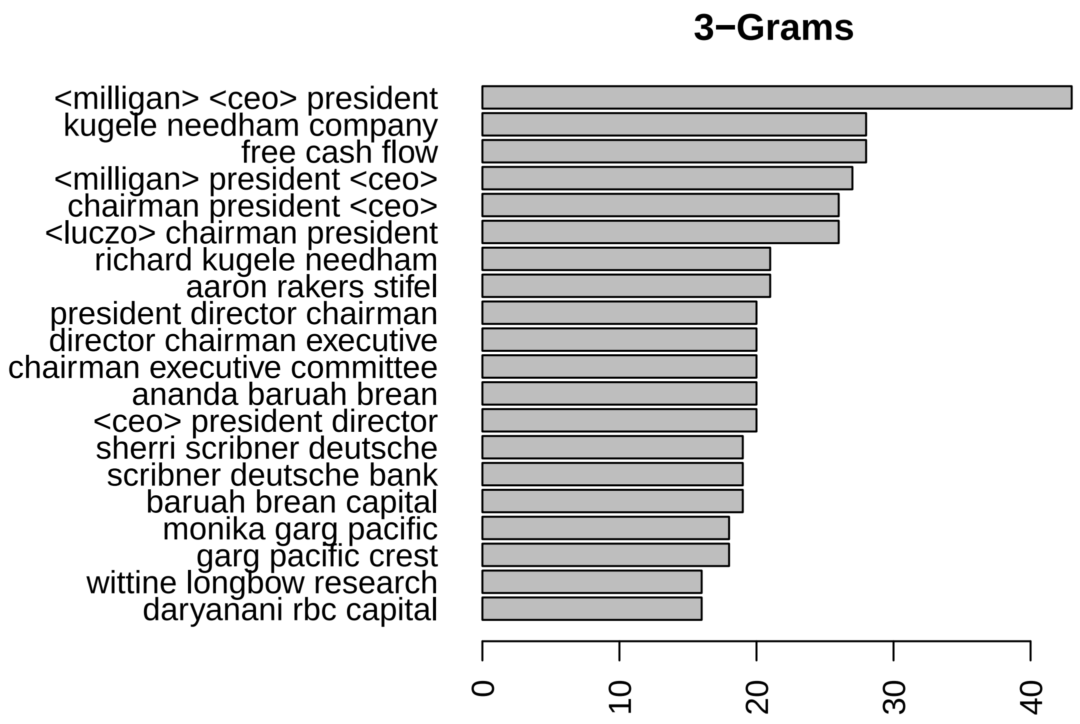
```r
#repeat analysis (selective plots)
tdm.1 <- TermDocumentMatrix(docs)
tdm.2 <- makeNGram( docs, 2)
tdm.3 <- makeNGram( docs, 3)
tdm.4 <- makeNGram( docs, 4)
tdm.5 <- makeNGram( docs, 5)
tdm.6 <- makeNGram( docs, 6)

mar.orig <- par()$mar
par(las=2, mar=c(5,6,4,2), mfrow=c(1,2))
barplot( mostFrequentTerms(tdm.1, 20), horiz=TRUE, main="1-Grams" )
barplot( mostFrequentTerms(tdm.2, 20), horiz=TRUE, main="2-Grams"   )
```

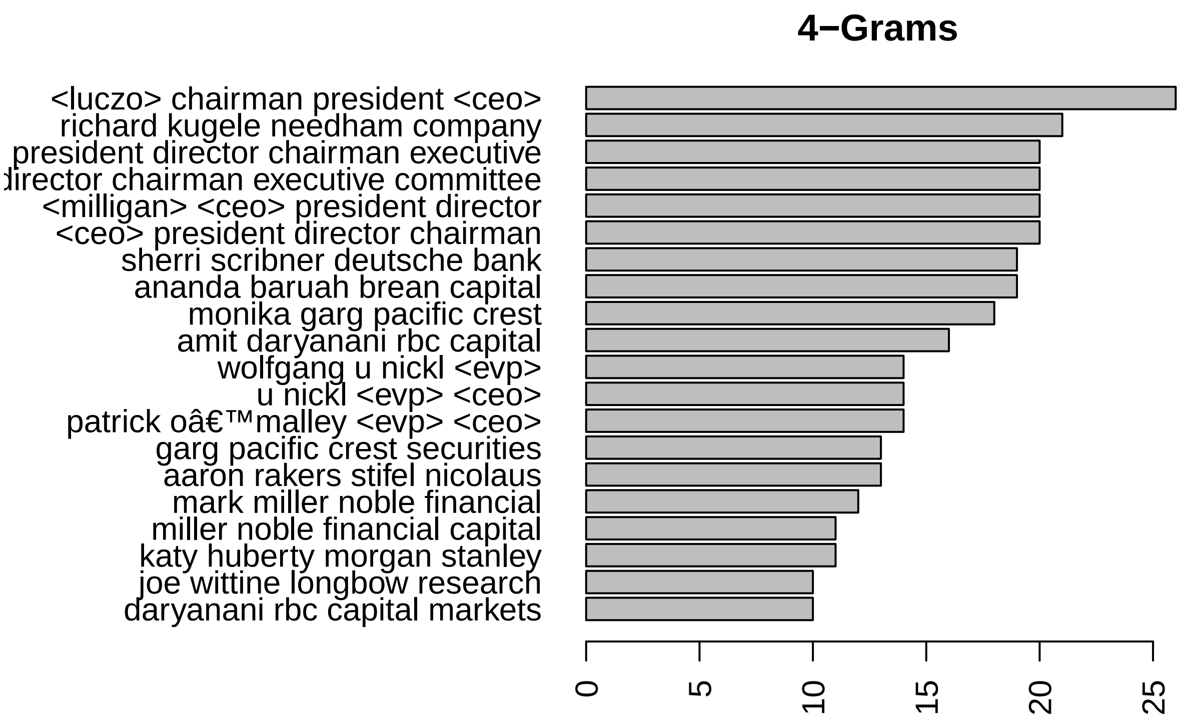


The CEO term is now part of the top 20 1-grams, and the 2-grams are starting to show person/role correlations. The more meaningful domain concepts are also starting to reveal themselves in the top 20, such as “gross margin”, “free cash”, and “half year”. With a little more cleanup, and diving deeper into the 2-grams list, I can imagine building my concepts fairly quickly.

```r
par(las=2, mar=c(3,15,2,2), mfrow=c(1,1))
barplot( mostFrequentTerms(tdm.3, 20), horiz=TRUE, main="3-Grams"   )
```



```r
barplot( mostFrequentTerms(tdm.4, 20), horiz=TRUE , main="4-Grams" )
```



3- and 4-grams are also richer now, drawing clear associations to Milligan and Luzco as CEO as well as other call participants and their organizations.

I've left off the 5- and 6- grams, since they are still a little unweildy. In the context of this corpus, names dominate the text. The next step would be to use these techniques to identify all the people and organizations, so that we could start doing text link and sentiment analysis. It should be easy to classify all the people so that we can boost the prominance of other terms. Looks like Part 3 is writing itself . . .

Appendix

References: The R code, data and markdown files used ot produce this document can be found at on Github at https://github.com/dsdaveh/R453-Text-Analytics

I found the videos on N-grams in Natural Language Processing by Dan Jurafsky, Christopher Manning - Stanford/Coursera particularly helpful.

The latest version of this document is at http://goo.gl/qc4X3b

Other functions used in this exercise:

```
mostFrequentTerms
```

```
## function (x.tdm, n=6) {
##   freq <- rowSums(as.matrix(x.tdm))
##   ord <- order(freq)
##   freq[tail(ord, n)]
## }
```

```
findTerms
```

```
## function (x.tdm, pattern) {
##   freq <- rowSums(as.matrix(x.tdm))
##   terms <- names(freq)
##   freq[grep(pattern, terms)]
## }
```

```
replaceSynonyms
```

```
## function(x, syn) {
##   #replace synonyms with primary term
##   syn
##   for (i in 1:nrow(syn)) {
##     #if(i%%100 == 0) print(sprintf("%d...", i))
##     root <- sprintf(" %s ", syn[i,1])
##     j <- 2
##     while (!is.null(syn[i,j]) && grepl("[a-z]", syn[i,j]) ) {   #first entry that has no chars (most
##       pattern <- sprintf("\\s%s\\s", syn[i,j])
##       x <- tm_map(x, content_transformer(function(y) gsub(pattern, root, y)))
##       j <- j+1
##     }
##   }
##   x
## }
```

```
findCoverageX
```

```
## function( tdm, pct ) {
##   freq <- rowSums(as.matrix(tdm))
##   x0 <- 0
##   xi <- as.integer(.2 * length(freq))  # initialize 20% steps
##   c <- sum(mostFrequentTerms(tdm, x0)) / sum(freq)
##   pos <- TRUE
##   while  (c < pct-.01 || c > pct+.01 ) {
##     if (c < pct ) { x0 <- x0+xi ; pos <- TRUE
##     } else        {
##       if (pos == TRUE) { x0 <- x0 - xi}
##       xi <- as.integer(xi/2)
##       pos <- FALSE
##     }
##     c <- sum(mostFrequentTerms(tdm, x0+xi)) / sum(freq)
##   }
##   x0+xi
## }
```

```
plotCoverage
```

```
## function ( tdm, n ) {
##   #create a Fibonacci series for xvalues
##   title.cp <- sprintf("Coverage Plot for %d-grams", n)
##
##   nterms <- tdm$nrow
##   freq <- rowSums(as.matrix(tdm))
##
##   xFib <- numeric()
##   xFib[1] <- xFib[2] <- 1
##   while (max(xFib) < nterms) { xFib <- c(xFib, sum(tail(xFib,2)))}
##   xFib <- xFib[c(-1,(-length(xFib)))]; #first 2 values are identical, last is out of bounds
##
##   ydata <- numeric()
##   for (i in seq_along(xFib))  {
##     ydata[i] <- sum(mostFrequentTerms(tdm,xFib[i]))/sum(freq)
##   }
##   plot( c(xFib, length(freq)), c(ydata,1), type="l", yaxt = "n"
##         , xlab="Term Rank (1 is most frequent)"
##         , ylab="Occurance in corpus", main=title.cp)
##   axis(2, at=pretty(seq(0,1,.2)), lab=paste0(pretty(seq(0,1,.2)) * 100, " %"), las=FALSE)
##   abline(h=.5, col="red")
##   abline(h=.9, col="red")
##
##   #trial and error (and looking at the graph)
##   cover.50 <- findCoverageX (tdm, .5)
##   cover.90 <- findCoverageX (tdm, .9)
##
##   abline(v=cover.50, col="red", lty=3, lwd=2)
##   abline(v=cover.90, col="red", lty=3, lwd=2)
##   text( x= cover.50, y=0.1, pos=3, sprintf("%d\nterms", cover.50))
##   text( x= cover.90, y=0.55, pos=3, sprintf("%d\nterms", cover.90))
## }
```