Demonstration of textreg Package

Luke Miratrix

November 21, 2014

1 Introduction

The following document illustrates the `textreg` package on a cut down version of the "Fat/Cat" database discussed in Miratrix & Ackerman (2014). In a nutshell, the textreg package allows for regressing a vector of +1/-1 labels onto raw text. The textreg package takes care of converting the text to all of the possible related features, allowing you to think of the more organic statement of regressing onto "text" in some broad sense.

1.1 Installing the textreg package

First install the `Rcpp` and `tm` packages. The first connects C++ to R in a nicer way than the default, and the second is a text manipulation package of great utility. You will also need a C++ compiler that R can access. We don't know how to advise you to get that if it is not already installed.
Once you have your compiler, you might try, if you have the package as a file on your system:

```
install.packages("textreg_0.1.tar.gz", repos = NULL, type="source")
```

You can also install via CRAN.

1.2 Getting ready to regress

To get started, load the package and the data. Here we use a small dataset that comes with the package.

```
library( textreg )
library( tm )
data( bathtub )
bathtub

## <<VCorpus (documents: 127, metadata (corpus/indexed): 0/1)>>
```

Notice it is a tm Corpus object.
Next obtain some labeling and decide on what ban words you want to use:

```
mth.lab = meta(bathtub)$meth.chl
table( mth.lab )

## mth.lab
##  -1   1
## 110  17

banwords = c( "methylene", "chloride")
```

Ban words are words in the text you wish to disallow from any summary generated. This is in liu of classic “stop-word” lists; they are situation-dependent. Classic stop words are automatically removed by appropriate regularization in the regression.

1.3 Obtaining the Summary

You get a summary by calling the `textreg` function. It has a lot of parameters, but let’s ignore them for now.

```
rs = textreg( bathtub, mth.lab, banwords, C=4, gap=1, min.support = 1,
            verbosity=0, convergence.threshold=0.00001, maxIter=100 )
rs

## textreg Results
##     C = 4 a = 1 Lq = 2
##     min support = 1  phrase range = 1-100 with up to 1 gaps.
##     itrs:  58 / 100
##
## Banned phrases: 'methylene', 'chloride'
##
## Label count:
##  -1   1
## 110  17
##
## Final model:
##             ngram     beta     Z support totalDocs posCount negCount
##       *intercept* -0.84537 1.000     127       127       17      110
##         contained  0.21282 4.359      15        13       10        3
##               due  0.01137 2.646       7         7        5        2
##             paint  0.44999 7.280      21        12        9        3
##       respiratory  0.76850 2.828       6         5        5        0
##        stripper *  2.39155 4.899      14        10       10        0
##         stripping  1.30798 3.317       9         8        7        1
##  vapors * heavier  0.18637 1.414       2         2        2        0
##     was * and * a  0.01670 1.414       2         2        2        0
```

One diagnostic to always check is whether there was convergence. If the number of iterations equals `maxIter`, it is likely there was no convergence. Try upping `maxIter` or relaxing your convergence threshold.

Note, you can also just pass a filename instead of the corpus as the first parameter. This is good if the file is very large and you don’t want to load it into R. The file needs to be one document per line of the file (so you will need to removed newlines, etc., from your documents in order to have made such a file).

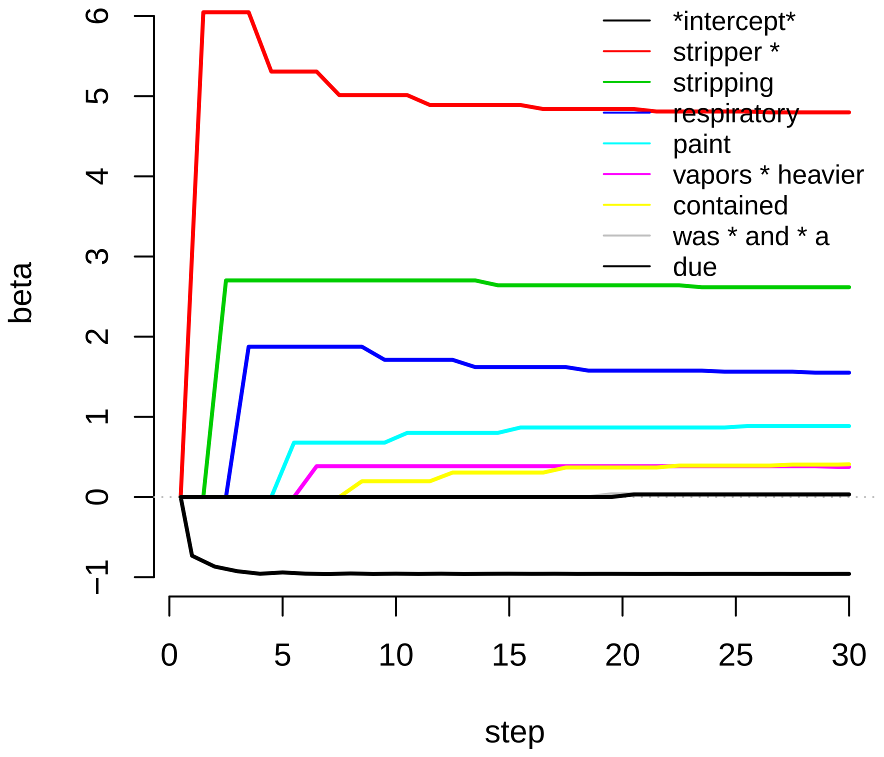
You can also print the results in a more easy-to-read form:

```
print( reformat.textreg.model( rs ), row.names=FALSE )
```

```
##             phrase num.phrase num.reports num.tag per.tag per.phrase
##        *intercept*        127         127      17      13        100
##          contained         15          13      10      77         59
##         stripper *         14          10      10     100         59
##              paint         21          12       9      75         53
##          stripping          9           8       7      88         41
##                due          7           7       5      71         29
##        respiratory          6           5       5     100         29
##   vapors * heavier          2           2       2     100         12
##      was * and * a          2           2       2     100         12
```

You can plot to see when phrases were introduced in the greedy coordinate descent.

```r
plot( rs )
```



This is simply a call to the provided `path.matrix.chart` method which uses `make.path.matrix` which is a matrix of all the coeficients for each step of the descent algorithm.

1.4 Tuning the Summary

There are several knobs that you can twiddle to change the summary you get from `textreg()`. The main ones to consider are

C The level of regularization. Bigger values give shorter summaries as it is harder for a phrase to be selected. Small values give longer summaries, and with C too small, you get phrases that are likely there due to random chance.

Lq The q for the L^q-rescaling of terms. Anything above 10 is treated as infinity. Bigger values means select more general phrases. Smaller values means select less general ones. 2 is standard.

positive.only Set this to TRUE or FALSE. Only allow positive features (other than the intercept). Useful if there are few positive documents and many negative, baseline, documents.

binary.features Set this to TRUE or FALSE. When TRUE, the feature vectors are changed to 0-1 vectors indicating whether a phrase is in or not in any given document, as compared to vectors of counts of how many times a phrases in a document. These feature vectors are regularized regardless.

min.support Phrases that do not appear this many times are not considered viable features. Increasing this number can substantially decrease the running time of the algorithm, but it will force the dropping of very rare phrases regardless of regularization choice. If you don't want to see very rare phrases, this is a good option (even if it is a bit ad hoc).

min.pattern, max.pattern Minimum and maximum lengths (in words) for phrases that are considered.

gap Number of words that can appear in a gap. A phrase can have multiple gaps of this length. So `gap=2` would allow, e.g., "the * * truck" as a phrase. For `gap=1` "the * truck * slowed" could be a phrase, because the skips are not adjacent.

Here are some different models we might fit:

```
rs5 = textreg( bathtub, mth.lab, banwords, C = 5, gap=1, min.support = 1,
            verbosity=0, convergence.threshold=0.00001, maxIter=100 )
rsLq5 = textreg( bathtub, mth.lab, banwords, C = 3, Lq=5, gap=1, min.support = 1,
               verbosity=0, convergence.threshold=0.00001, maxIter=100 )
rsMinSup10 = textreg( bathtub, mth.lab, banwords, C = 3, Lq=5, gap=1, min.support = 10,
                    verbosity=0, positive.only=TRUE, convergence.threshold=0.00001, maxIter=100 )
rsMinPat2 = textreg( bathtub, mth.lab, banwords, C = 3, Lq=5, gap=1, min.support = 1,
                   min.pattern=2, verbosity=0, convergence.threshold=0.00001, maxIter=100 )
```

We can merge lists to see overlap quite easily via the `make.list.table` command. This gives a table that we can easily render in latex:

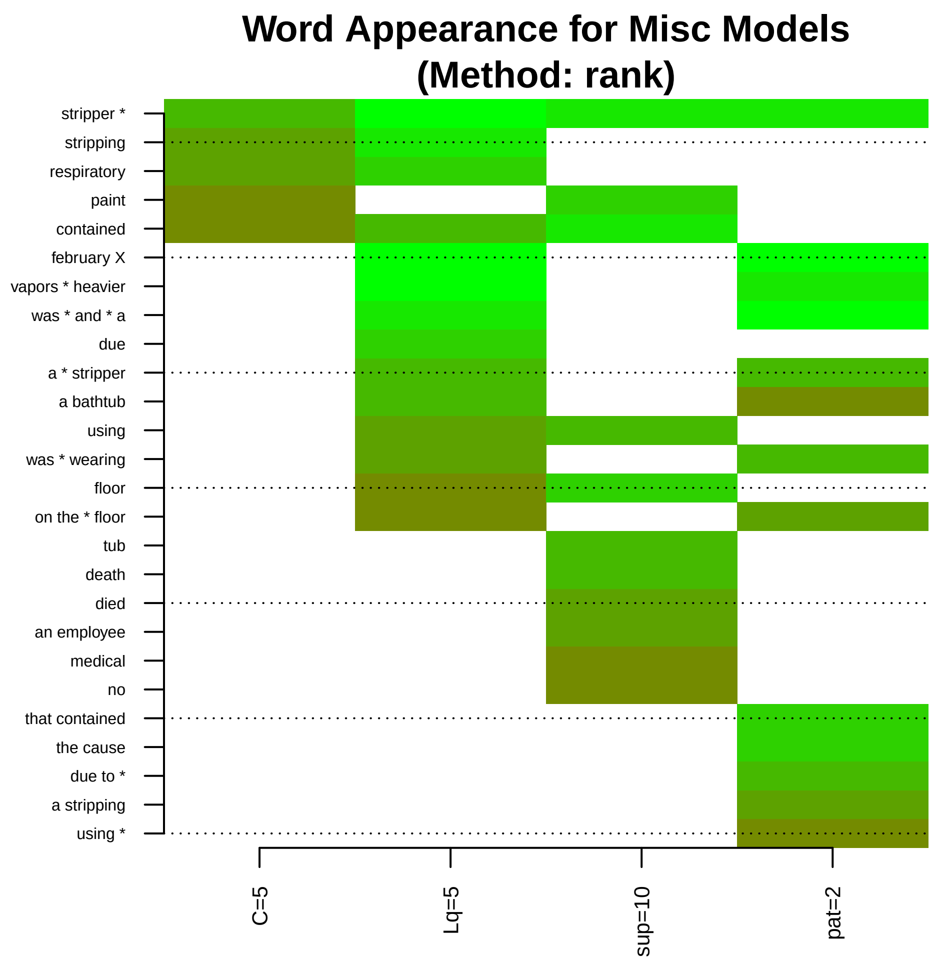
```
library(xtable)
lst = list( rs5, rsLq5, rsMinSup10,rsMinPat2 )
names(lst) = c("C=5", "Lq=5","sup=10","pat=2")
tbl = make.list.table( lst, topic="Misc Models" )
print( xtable( tbl, caption="Table from the make.list.table call" ),
       latex.environments="tiny" )
```

	phrase	C=5	Lq=5	sup=10	pat=2	num.phrase	num.reports	num.tag	per.tag	per.phrase
1	stripper *	5.00	12.00	11.00	11.00	14	10	10	100.00	59.00
2	stripping	4.00	10.00			9	8	7	88.00	41.00
3	respiratory	3.00	8.00			6	5	5	100.00	29.00
4	paint	2.00		9.00		21	12	9	75.00	53.00
5	contained	1.00	5.00	10.00		15	13	10	77.00	59.00
6	february X		14.00		12.00	2	2	2	100.00	12.00
7	vapors * heavier		13.00		10.00	2	2	2	100.00	12.00
8	was * and * a		11.00		14.00	2	2	2	100.00	12.00
9	due		9.00			7	7	5	71.00	29.00
10	a * stripper		7.00		7.00	4	4	4	100.00	24.00
11	a bathtub		6.00		2.00	9	9	5	56.00	29.00
12	using		4.00	7.00		22	22	9	41.00	53.00
13	was * wearing		3.00		5.00	7	7	5	71.00	29.00
14	floor		2.00	8.00		20	16	4	25.00	24.00
15	on the * floor		1.00		4.00	3	3	2	67.00	12.00
16	tub			6.00		20	8	5	62.00	29.00
17	death			5.00		10	9	5	56.00	29.00
18	died			4.00		17	17	7	41.00	41.00
19	an employee			3.00		19	19	4	21.00	24.00
20	medical			2.00		29	25	8	32.00	47.00
21	no			1.00		17	16	5	31.00	29.00
22	that contained				9.00	3	3	3	100.00	18.00
23	the cause				8.00	4	4	4	100.00	24.00
24	due to *				6.00	6	6	4	67.00	24.00
25	a stripping				3.00	2	2	2	100.00	12.00
26	using *				1.00	21	21	8	38.00	47.00

Table 1: Table from the make.list.table call

See latex table for results.
You can also plot this side-by-side table

```
list.table.chart( tbl )
```



1.5 Selecting C

C is the main tuning parameter for a regularized regression. In the above we just used a default $C = 4$, which is just large enough to drop singleton phrases that are "perfect predictors." Better choices are possible. One way is to select one via obtaining a permutation distribution on this parameter under a null of no connection between text and labeling. Do so as follows:

```
Cs = find.threshold.C( bathtub, mth.lab, banwords, R = 100, gap=1, min.support = 5,
                       verbosity=0, convergence.threshold=0.00001 )
Cs[1]

## [1] 9.901

summary( Cs[-1] )

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    3.73    4.63    5.22    5.30    5.96    7.65

C = quantile( Cs, 0.95 )
C

##   95%
## 6.853
```

The $Cs[1]$ term gives you the penalty needed to get no selected phrases (a null model) on your original labeling. If this is much larger than the permutation distribution, you know you have a real connection between the text and the labeling, even after dropping banned words and phrases outside the specified support.

Important: The `find.threshold.C` function shares the parameters of the `textreg` function. By using the same parameters in both calls, you will find the appropriate null distribution given the phrases allowed by the other parameters such as `min.pattern` and so forth.

1.6 Dropping documents

You can drop documents from the regression by setting the corresponding label to 0 instead of +1 or -1. For example

```
mth.lab.lit = mth.lab
mth.lab.lit[20:length(mth.lab)] = 0
rs.lit = textreg( bathtub, mth.lab.lit, banwords, C = 4, gap=1, min.support = 1, verbosity=0 )
rs.lit

## textreg Results
##     C = 4 a = 1 Lq = 2
##     min support = 1  phrase range = 1-100 with up to 1 gaps.
##     itrs:  17 / 40
##
## Banned phrases: 'methylene', 'chloride'
##
## Label count:
## -1  1
## 11  8
##
```

```
## Final model:
##         ngram    beta     Z support totalDocs posCount negCount
##   *intercept* -0.2933 1.000      19        19        8       11
##     contained  0.2588 2.646       7         7        6        1
##      stripper  0.5936 2.236       5         5        5        0
##     stripping  0.2643 2.828       6         5        5        0

rs.lit$labeling

##  [1] -1 -1  1 -1  1  1 -1 -1 -1 -1 -1  1  1 -1 -1 -1  1  1  1
```

Note how we can get the subset labeling from the `textreg.result` object. This can be useful for some of the text exploratory calls that take a result object and a labeling.

2 Exploring the Text

The textreg package also offers a variety of ways to explore your text. Some of these methods work with objects returned from the `textreg()` command, and some just extend the capability of the `tm` package and are generally useful.

2.1 Finding Where Phrases Appear

It is easy to see which selected features are in which positive documents by generating the "phrase matrix" which is effectively the design matrix of the regression (with all unimportant columns dropped). Here we look at the phrase matrix for the full bathtub regression, above, and the one limited to the subset in the "Dropping Documents" section, above.

```
hits = phrase.matrix( rs )
dim( hits )

## [1] 127   9

t( hits[ 1:10, ] )

##                   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## *intercept*          1    1    1    1    1    1    1    1    1     1
## contained            0    1    1    0    1    1    0    0    0     0
## due                  0    0    0    0    0    1    0    0    0     1
## paint                0    0    5    0    1    2    1    0    0     2
## respiratory          0    0    2    0    1    0    0    0    0     0
## stripper *           0    0    0    0    1    1    0    0    0     0
## stripping            0    0    0    0    1    0    0    0    0     0
## vapors * heavier     0    0    0    0    0    0    0    0    0     0
## was * and * a        0    0    0    0    0    0    0    0    0     0

hits.lit = phrase.matrix( rs.lit )
dim(hits.lit)

## [1] 19  4
```

Note the transpose, above, making the rows the phrases and the columns documents. This is just for ease of printing.
Also note that, in the `rs.lit` case, since documents were dropped by the labeling, this method will also drop them from the phrase matrix. (See above about dropping documents.)

Once you have your phrase matrix, you can calculate the number of "important" phrases in each document, or the total number of times a phrase appears (these numbers are already in the result object, however).

```
apply( hits[ mth.lab == 1, ], 1, sum )

##  [1] 9 6 6 5 4 4 4 5 6 1 7 5 6 6 3 3 2

apply( hits[ mth.lab == 1, ], 2, sum )

##      *intercept*        contained              due            paint
##               17               11                5               17
##      respiratory        stripper *        stripping vapors * heavier
##                6               14                8                2
##    was * and * a
##                2
```

2.2 Independent Search Methods

We provide several methods that allow you to directly explore text without a `textreg.result` object. You can use these even if you are not using the regression function of this package at all. For example, to look at the appearance pattern of individual terms try the following:

```
tt2 = phrase.count( "tub * a", bathtub )
head( tt2 )

## [1] 0 0 0 0 0 0

table( tt2, dnn="Counts for tub * a" )

## Counts for tub * a
##   0   1   3
## 124   2   1
```

You can further investigate the appearance patterns for phrases by making a table of which documents have which phrases. Again, you can look for any phrases you want using these methods, even if they are not part of your original CCS summary.

```
tab = make.phrase.matrix( c( "bathtub", "tub * a" ), bathtub )
head( tab )

##      bathtub tub * a
## [1,]       1       0
## [2,]       1       0
## [3,]       1       0
## [4,]       1       0
## [5,]       1       0
## [6,]       2       0

table( tab[,2] )

##
##   0   1   3
## 124   2   1
```

Note the tally numbers are the same as above.
You can also just get total counts of the phrases in the corpus. The only advantage of this is you can check phrases that were not returned in your textreg result object.

```r
ct = make.count.table( c( "bathtub", "tub * a", "bath" ), mth.lab, bathtub )
ct

##         n.pos  n per.pos per.tag
## bathtub     5 16      31      29
## tub * a     6 10      60      35
## bath        7 18      39      41
```

2.3 Finding Phrases' Contexts

You can grab snippits of text that include phrases quite easily. For example, here are the first three appearances of "bathtub":

```r
tmp = grab.fragments( "bathtub", bathtub, char.before=30, char.after=30, clean=TRUE )
tmp[1:3]

## $`1`
## [1] "teriors inc were installing a BATHTUB surround in the bathroom of a"
##
## $`2`
## [1] "ng down a pallet containing a BATHTUB kit and knocked down a storag"
##
## $`3`
## [1] "remove the old coating from a BATHTUB the employee was found dead a"
```

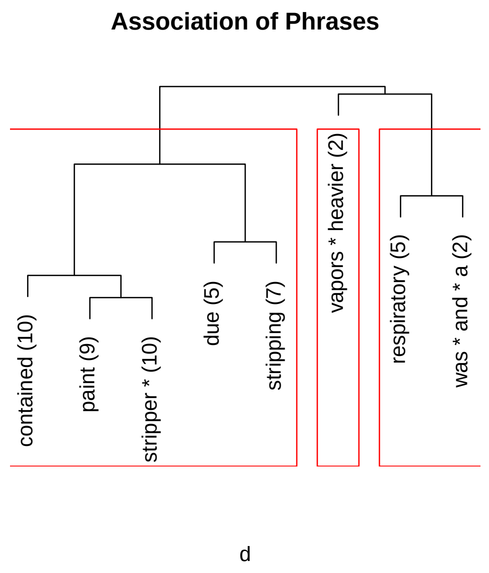
If a document has a phrase multiple times, you will get multiple results for that document. Here is where "tub * a" comes from, divided into positive and negative classes by the passed labeling:

```r
frags = sample.fragments( "tub * a", mth.lab, bathtub, 20, char.before=30, char.after=30 )
frags

## $`tub * a`
##
## Profile of Summary Phrase: 'tub * a'
## Positive: 3/17 = 17.65
## Negative: 0/110 = 0.00
## Appearance of 'tub * a' in positively marked documents:
## * the tub and spread around the TUB WITH A brush it was allowed to react
## * ping paint off of a porcelain TUB USING A methylene chloride based stri
## * aptistery similar to a sunken TUB USING A push broom and a chemical cal
##
## Appearance of 'tub * a' in baseline documents.
```

2.4 Relationships between phrases

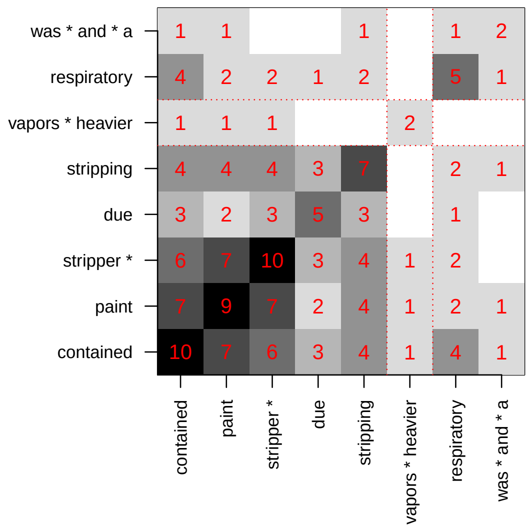
Sometimes, especially for summaries of many positive documents, there are multiple aspects that are being summarized with clusters of phrases. We have two vizualizations that help understand how phrases interact.
The first is a simple clustering of the phrases based on their appearance in the positively marked documents only. This is only meaningful if the negative, baseline documents are to be construed only as a backdrop. Clustering is based on a scaled overlap statistic.

```
cluster.phrases( rs, num.groups=3 )
```



The num.groups parameter is how many clusters to make.
The second vizualization is a heat-map of the pairwise correlations of all selected phrases. You can plot the number of documents shared by each pair as well.

```
make.phrase.correlation.chart( rs, count=TRUE, num.groups=3 )
```



The count=TRUE means use raw counts (easier to interpret) rather than the scaled overlap statistic.

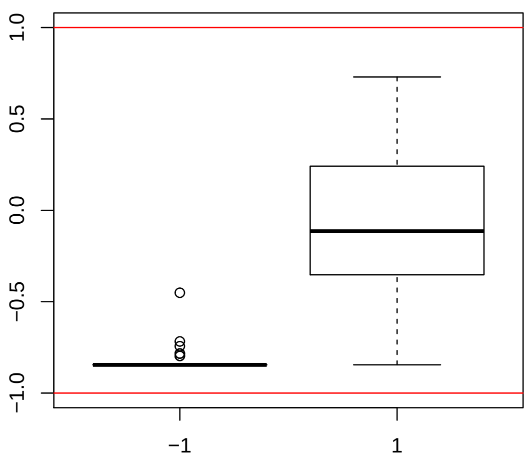
3 Prediction

You can use the phrases to predict the labeling both for your original documents or new out-of-sample documents if you wish. First, you can obtain an overall measure of how well one can predict the labeling with the phrases:

```
calc.loss( rs )

## tot.loss     loss  penalty
##    46.40    25.02    21.38
```

This might be useful for selecting C based on cross-validated prediction accuracy or similar. You can also, if you wish, examine the prediction ability of phrases on individual documents.

```
pds = predict( rs )
labs = rs$labeling
table( labs )

## labs
##  -1   1
## 110  17

boxplot( pds ~ labs, ylim=c(-1,1) )
abline( h=c(-1,1), col="red" )
```



Note many of the predictions for positively marked documents remain very negative. This is typical when there are few positive examples. Also note the `lab=rs$labeling` line—this will give you the final labeling used by `textreg` after any 0s have been dropped.

3.1 Out of Sample Prediction

Here we split the sample and train on one part and test on the other.

```
  smp = sample( length(bathtub), length(bathtub)*0.5 )
        rs = textreg( bathtub[smp], mth.lab[smp], C = 3, gap=1, min.support = 5,
              verbosity=0, convergence.threshold=0.00001, maxIter=100 )
        rs

## textreg Results
##     C = 3 a = 1 Lq = 2
##     min support = 5  phrase range = 1-100 with up to 1 gaps.
##     itrs:  24 / 100
##
## Banned phrases: ''
##
## Label count:
## -1  1
## 56  7
##
## Final model:
##        ngram    beta     Z support totalDocs posCount negCount
##  *intercept* -0.8966 1.000      63        63        7       56
##     chloride  2.2244 6.000      14         7        7        0
##    contained  1.0814 2.828       6         5        5        0

        train.pred = predict( rs )
        test.pred = predict( rs, bathtub[-smp] )

        train.loss = calc.loss( rs )
        train.loss

## tot.loss     loss  penalty
##   16.542    6.625    9.917

        test.loss = calc.loss( rs, bathtub[-smp], mth.lab[-smp] )
        test.loss

## tot.loss     loss  penalty
##   24.991   15.073    9.917
```

You might want to think carefully about how to do this if the negative documents far outweigh the positive ones.

3.2 Cross Validation

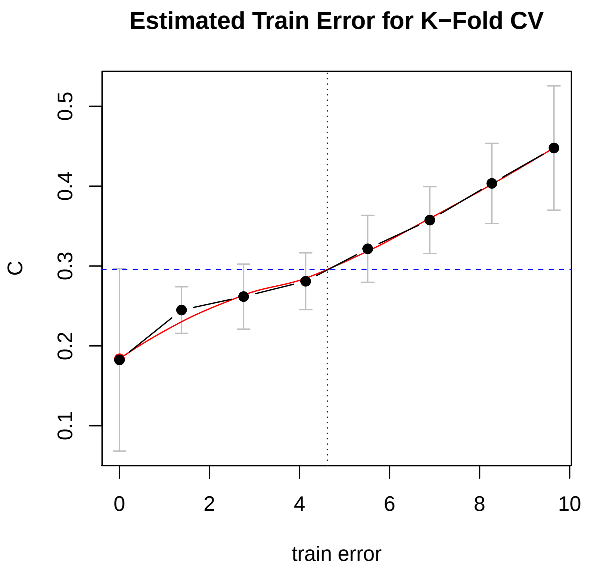
We can find an optimal C via cross-validation as follows:

```
  tbl = find.CV.C( bathtub, mth.lab, c("methylene","chloride"), 4, 8, verbosity=0 )
  print( round( tbl, digits=3 ) )

##      Cs train.err test.err std_err
## 1 0.000     0.000    0.182   0.114
## 2 1.379     0.031    0.245   0.029
## 3 2.757     0.106    0.262   0.041
## 4 4.136     0.186    0.281   0.036
## 5 5.514     0.263    0.322   0.042
## 6 6.893     0.348    0.358   0.042
## 7 8.272     0.422    0.403   0.050
## 8 9.650     0.454    0.448   0.078
```

This is 4-fold cross validation evaluated at 8 different values of C ranging from no regularization ($C = 0$) to full regularization (C just large enough to give a null model). We get a

table of test error. We would then typically pick a C that has a test error one SE larger than the minimum.
You can get this via the rather clumsy `make.CV.chart` method, which returns such a C:

```
  rs = make.CV.chart( tbl )
```



```
  rs

## $minimum
## [1] 4.616
##
## $test.err
## [1] 0.2956
```

4 Cleaning Text and Stemming

You can easily clean dirty text and stem it.

```
data( dirtyBathtub )
strwrap( dirtyBathtub$text[[1]] )

## [1] "Two employees of Unique Interiors Inc., were installing a bathtub"
## [2] "surround in the bathroom of a single family home. While applying"
## [3] "contact cement to the back of the tub surround, the vapors from the"
## [4] "cement ignited the pilot light in a gas powered water heater. Because"
## [5] "of the lack of ventilation, the vapors and the heater created a flat"
## [6] "tire. Employee #1 was burned over 90 percent of his body and died."
## [7] "Employee #2 was burnt over 15 percent of his body and he was"
## [8] "hospitalized."

bc = Corpus( VectorSource( dirtyBathtub$text ) )

bc.clean = clean.text( bc )
strwrap( bc.clean[[1]] )
```

```
## [1] "two employees of unique interiors inc were installing a bathtub"
## [2] "surround in the bathroom of a single family home while applying contact"
## [3] "cement to the back of the tub surround the vapors from the cement"
## [4] "ignited the pilot light in a gas powered water heater because of the"
## [5] "lack of ventilation the vapors and the heater created a flat tire"
## [6] "employee X was burned over XX percent of his body and died employee X"
## [7] "was burnt over XX percent of his body and he was hospitalized"

bc.stem = stem.corpus(bc.clean, verbose=FALSE)
strwrap( bc.stem[[1]] )

## [1] "two employe+ of uniqu+ interior+ inc were instal+ a bathtub surround+"
## [2] "in the bathroom of a singl+ famili+ home+ while appli+ contact+ cement"
## [3] "to the back+ of the tub surround+ the vapor+ from the cement ignit+ the"
## [4] "pilot light+ in a gas power+ water heater becaus+ of the lack of"
## [5] "ventil+ the vapor+ and the heater creat+ a flat tire employe+ X was"
## [6] "burn+ over XX percent of his bodi+ and die+ employe+ X was burnt over"
## [7] "XX percent of his bodi+ and he was hospit+"
```

Everything else works. For the textreg package, the "+" are automatically turned into wildcards when doing phrase search in the original (cleaned but not stemmed) text. We need updated banwords to account for the stemming, but other than that, everything is the same; we are doing business as usual on the transformed text:

```
  res.stm = textreg( bc.stem, mth.lab, c("chlorid+", "methylen+"), C=4, verbosity=0 )
  res.stm

## textreg Results
##     C = 4 a = 1 Lq = 2
##     min support = 1  phrase range = 1-100 with up to 0 gaps.
##     itrs:  40 / 40
##
## Banned phrases: 'chlorid+', 'methylen+'
##
## Label count:
##  -1   1
## 110  17
##
## Final model:
##          ngram     beta     Z support totalDocs posCount negCount
##    *intercept* -0.83022 1.000     127       127       17      110
##            due  0.02126 2.646       7         7        5        2
##          paint  0.89595 7.280      21        12        9        3
##   respiratori+  0.32978 2.828       6         5        5        0
##         strip+  0.99358 6.856      19        11       10        1
##       stripper  2.16583 5.385      15        10       10        0
##  that contain+  0.22084 1.732       3         3        3        0

  sample.fragments( "that contain+", res.stm$labeling, bc.stem, 5, char.before=10 )

## $`that contain+`
##
## Profile of Summary Phrase: 'that contain+'
## Positive: 3/17 = 17.65
## Negative: 0/110 = 0.00
## Appearance of 'that contain+' in positively marked documents:
## *  stripper THAT CONTAIN+ at least
## * an strip+ THAT CONTAIN+ XX XX pe
## * ip+ agent THAT CONTAIN+ methylen
##
## Appearance of 'that contain+' in baseline documents.

  sample.fragments( "that contain+", res.stm$labeling, bc.clean, 5, char.before=10 )
```

```
## $`that contain+`
##
## Profile of Summary Phrase: 'that contain+'
## Positive: 3/17 = 17.65
## Negative: 0/110 = 0.00
## Appearance of 'that contain+' in positively marked documents:
## *  stripper THAT CONTAINED at least
## * ean strip THAT CONTAINED XX XX per
## * ing agent THAT CONTAINED methylene
##
## Appearance of 'that contain+' in baseline documents.
```

This vastly increases the ease of understanding a stemmed phrase or word.
Future work would be to be able to retrieve phrases in the original "dirty" text; that would be a useful addition. It will mostly work now, but dropped punctuation, etc., can mess up phrase retrieval.
A final note is if generating the cleaned corpus is time consuming, there is a small helper function `save.corpus.to.files` that will write out your corpus to a text file and a `Rda` file. The text file's name can then be passed to textreg, thus avoiding the need to load the corpus into R's memory. This is recommended to avoid a lot of copying of large objects back and forth in memory.