Check for updates

The Stata Journal (2014)
14, Number 4, pp. 817–829

txttool: Utilities for text analysis in Stata

Unislawa Williams
Spelman College
Atlanta, GA
uwilliams@spelman.edu

Sean P. Williams
SunTrust Bank
Atlanta, GA
sean.williams.1000@gmail.com

Abstract. This article describes `txttool`, a command that provides a set of tools for managing free-form text. The command integrates several built-in Stata functions with new text capabilities. These latter functions include a utility to create a bag-of-words representation of text and an implementation of Porter's (1980, *Program: Electronic library and information systems* 14: 130–137) word-stemming algorithm. Collectively, these utilities provide a text-processing suite for text mining and other text-based applications in Stata.

Keywords: dm0077, txttool, text mining, Porter stemmer, bag of words, cleaning, stop words, subwords

1 Introduction

Stata users recently introduced new commands for increasingly sophisticated management of text data, in particular, `screening` and `kountry` (Belotti and Depalo 2010; Raciborski 2008). As these authors note, while text data can be challenging to analyze, they occur in many applications, including free-form electronic patient records, country identifiers in international relations and economics datasets, and open-ended survey responses. In addition, text mining, or the quantitative analysis of unstructured text data, has received increasing attention in the social sciences (Benoit, Laver, and Mikhaylov 2009; Lowe and Benoit 2013). These applications vary from categorizing and classifying legislative speeches to evaluating Russian military discourse (Laver, Benoit, and Garry 2003; Grimmer and Stewart 2013).

Generally, Stata lacks the text-management utilities to prepare text data for these kinds of applications. Although some steps in the data-preparation phase can be accomplished with built-in commands, using them can be tedious and can lead to errors. For example, removing punctuation, extra white spaces, and special characters from text could involve dozens of lines of code, and the code would need to be modified for different situations. On the other hand, some steps in the data-preparation phase, such as stemming words or representing instances of words as counts in numeric variables, are not available in Stata at all.

The `txttool` command fills this gap by providing utilities for several text-preparation tasks:

1. Cleaning: removing punctuation, special characters, and extra white spaces from text and converting it to lowercase.

© 2014 StataCorp LP

2. Stop-word removal: removing words that occur too frequently to discriminate outcomes or classes (that is, "the", "of", "and", etc.) or user-specified words that lead to better results in a particular context when removed.

3. Substitution: replacing large numbers of individual words to correct misspellings or variations or to insert user-defined categories for words when analytically useful.

4. Stemming: an implementation of Porter's (1980) stemming program that reduces a word to a stem or a root. For example, the words "programs", "programmed", and "programming" all reduce to the stem "program". The stem is not always a valid word, and it does not need to be. Rather, its purpose is to reduce the overall word count by grouping closely related words into the same stem.

5. Bag of words: converting a string into both a list of the unique words found in the string and a count of each unique word. This allows a variety of quantitative analyses, including discriminate analysis, clustering, and the creation of dictionaries, to represent different outcomes in other variables.

All of these features are described in more detail below. Afterward, extensions with other programs and with Mata are discussed.

2 The txttool command

2.1 Syntax

`txttool` *varname* [*if*] [*in*], {`generate(`*newvar*`)` | `replace`} [`stem` `stopwords(`*filename*`)` `subwords(`*filename*`)` `bagwords` `prefix(`*string*`)` `noclean` `nooutput`]

varname is the string variable containing the text to be processed.

2.2 Options

`generate(`*newvar*`)` creates a new variable, *newvar*, containing the processed text of *varname*. The *newvar* will be a copy of *varname* that has been stemmed, has had the stop words removed, has had words substituted, or has been cleaned, depending on the other options specified. Either `generate()` or `replace` is required.

`replace` replaces the text in *varname* with text that has been stemmed, has had the stop words removed, has had words substituted, or has been cleaned, depending on the other options specified. Either `generate()` or `replace` is required.

`stem` calls the Porter stemmer implementation to stem all the words in *varname*.

`stopwords(`*filename*`)` indicates that the program should remove all instances of words contained in *filename*. The *filename* is a list of words in a text file. Although a list of

frequently used English words is supplied with `txttool`, users can use different lists of stop words in different applications by specifying different filenames. Stop-word lists without punctuation are recommended.

`subwords(`*filename*`)` indicates that the program should substitute instances of words in *filename* with another word in *filename*. The *filename* is a tab-delimited text file, where the first column is the word to be replaced and the second column is the substitute text. Users can use different lists of words to substitute in different applications by specifying different filenames. Subword lists without punctuation are recommended.

`bagwords` tells `txttool` to create a bag-of-words representation of the text in *varname*. The bag-of-words representation consists of new variables, one for each unique word in *varname*, with the count of the occurrences of each word. The new variables are named with the convention *prefix_word*, where *prefix* is optionally supplied by the user, and *word* is the unique word in the text. The options `generate()` and `bagwords` can be used together to represent the processed text as one column with word counts.

`prefix(`*string*`)` supplies a prefix for the variables created in `bagwords`. The default is `prefix(w_)`. Supplying a prefix will automatically invoke the `bagwords` option. Note that `txttool` does not know what variables will be created before processing the text, so it cannot confirm the absence of variables already named with the specified prefix. Errors will therefore result if the chosen prefix matches an existing variable.

`noclean` specifies that the program should not remove punctuation, extra white spaces, and special characters from *varname*. By default, `txttool` will clean and lowercase *varname*. The `noclean` option is not allowed with `bagwords`. In addition, because the Porter stemmer does not stem punctuation and because the stop-words and subwords lists should not include punctuation, `noclean` should be used with caution.

`nooutput` suppresses the default output. By default, `txttool` reports the total number of words and the count of unique words before and after processing, as well as the time elapsed during processing. The `nooutput` option suppresses this output, which can save some time with large processing tasks.

2.3 Remarks

The options are processed in the following order: `noclean`, `subwords()`, `stopwords()`, `stem`, `generate()` or `replace`, and, finally, `bagwords`. Thus the `noclean` option is examined first, and if it is not specified, punctuation and special characters are removed. Then subwords are substituted and stop words removed; the remaining text is stemmed or bagged. The Porter stemmer algorithm does not recognize punctuation or non-English characters, and Stata does not allow variable names with punctuation and non-English characters, so cleaning must precede stemming and bagging.

Because cleaning comes first, the user-defined lists for `subwords()` and `stopwords()` are most effective when they are themselves "cleaned". Otherwise, they may reintroduce punctuation and other characters after cleaning. The `noclean` option allows users to process `subwords()` and `stopwords()` without first cleaning the original text, if this is required in a particular instance. However, a more effective approach is cleaning the stop-word and subword lists by reading the lists into Stata and processing the lists with `txttool stopwordlist, gen(stopwordlist2)` and `txttool subwordlist, gen(subwordlist2)` to obtain word lists that have had any punctuation and special characters removed. These lists can then be exported as text files to use in later applications with `txttool`.

The program's default behavior is to remove all characters except white space (American standard code for information interchange [ASCII] code 32), numerals (ASCII codes 48–57), and letters (ASCII codes 97–122 after lowercasing). Therefore, the default behavior is to remove punctuation, non-English characters, and nonprinting characters. Removing these characters is assumed by the Porter stemmer, which was created for English, and necessary for creating new variables of the unique words with the `bagwords` option because Stata does not allow special characters in variable names. While `bagwords` is not allowed with `noclean`, the `txttool` command allows the `stem` option when `noclean` is specified, although the stemmer may not function as expected, so users should examine the results carefully. Analysis of non-English text can still use the `stopwords()`, `subwords()`, `generate()`, and `replace` options even if `noclean` must be specified to accommodate the characters in a particular language.

3 Examples

3.1 Examples of options usage

Imagine we have text data in the form of open-ended answers to a survey on voter attitudes.

```
. use example_text
. list
```

	txtexample
1.	Unemployment is the major issue, but no one´s talking about it
2.	I´ve been looking and looking but i still can´t find a job
3.	I hear a lot of talk about the ecomony improving

It is best to show how to prepare the text for analysis step by step. First, cleaning removes punctuation and any special characters and lowercases the text.

```
. txttool txtexample, gen(cleaned)
Input:   28 unique words, 33 total words
Output:  27 unique words, 33 total words
Total time: .453 seconds
```

The researcher may wish to use a preexisting coding scheme and group terms such as "unemployment" and "jobs" into one term such as "employment." In addition, the researcher may correct common misspellings in a particular context, for example, replacing "ecomony" with "economy" in the third observation. Note also that the `subword` option can substitute phrases for words so that, for example, contractions in the text can be expanded. The researcher may define the `subword` list as the following tab-delimited text file:

```
unemployment    employment
job     employment
ecomony     economy
ive     i have
```

Using this list of substitutions produces the following text:

```
. txttool txtexample, gen(subbed) subwords("subwordexample.txt")
  (output omitted)
. list subbed
```

	subbed
1.	employment is the major issue but no ones talking about it
2.	i have been looking and looking but i still cant find a employment
3.	i hear a lot of talk about the economy improving

The `stopwords()` option will remove user-defined words. The `txttool` program is packaged with a list of common English words and contractions to be removed. Common words such as "I", "of", etc., generally have little discriminating power but increase memory requirements. Using the packaged list of stop words produces

```
. txttool txtexample, gen(stopped) subwords("subwordexample.txt")
> stopwords("stopwordexample.txt")
  (output omitted)
. list stopped
```

	stopped
1.	employment major issue talking
2.	looking looking still find employment
3.	hear lot talk economy improving

Stemming the text further reduces the words by removing all but the word stems.

```
. txttool txtexample, gen(stemmed) subwords("subwordexample.txt")
> stopwords("stopwordexample.txt") stem
```

(*output omitted*)

```
. list stemmed
```

	stemmed
1.	employ major issu talk
2.	look look still find employ
3.	hear lot talk economi improv

Note that "talking" and "employment" are now reduced to "talk" and "employ", respectively. Finally, the text is "bagged", and a variable with the prefix `"w_"` is produced for each remaining word, along with the count of the word in each line of text.

```
. txttool txtexample, gen(bagged) subwords("subwordexample.txt")
> stopwords("stopwordexample.txt") stem bagwords prefix(w_)
. list w_*
```

	w_employ	w_major	w_issu	w_talk	w_look	w_still	w_find
1.	1	1	1	1	0	0	0
2.	1	0	0	0	2	1	1
3.	0	0	0	1	0	0	0

	w_hear	w_lot	w_econ~i	w_improv
1.	0	0	0	0
2.	0	0	0	0
3.	1	1	1	1

3.2 Creating a dictionary

One practical text-mining task is creating a dictionary for specific values of a variable. A dictionary is a list of words, phrases, parts of speech, or other tokens that distinguish one value of a variable from another. In a customer-relationship management application, for example, a researcher may have data on products that customers returned as well as customer descriptions of the products. Using the customer descriptions, the researcher can find the words or phrases that distinguish returned and unreturned products. The list of words can then be studied to see what aspects of product design or manufacture lead to returns; it can also be used to "tag" future customer descriptions to track trends or used in a predictive model to classify large numbers of descriptions of competitors' products. Similar applications can be made to any text data with an associated outcome of interest.

The `bagwords` option makes it easy to create dictionaries of this kind. To illustrate, we will use the International Monetary Fund's monitoring of fund arrangements data on loan terms. The data include the International Monetary Fund's description of the conditions that a borrowing government must meet to maintain funding as well as a status variable that describes whether the country met the condition. We are specifically interested in the kinds of loan conditions that borrowers have the most difficulty meeting, which is a task for dictionary creation.[1] We set up the data as follows:

```
. import delimited using "Mona.csv", clear
(20 vars, 6549 obs)
. keep if status=="M" | status=="NM"
(2521 observations deleted)
. set seed 1234
. sample 500, count
(3528 observations deleted)
```

To begin, we assess how many unique words and total words are contained in the description of the loan conditions, named `descpt`.

```
. txttool descpt, gen(test1)
Input:   2798 unique words, 9706 total words
Output:  2095 unique words, 9707 total words
Total time: .896 seconds
```

After cleaning, 2,095 unique words remain out of 9,707 total words in only 500 descriptions of loan conditions. This is obviously a large number of words to parse by reading. We use the example list of packaged stop words with `txttool` to reduce the word count.

```
. txttool descpt, gen(test2) stopword("stopwordexample.txt")
Input:   2798 unique words, 9706 total words
Output:  2021 unique words, 6188 total words
Total time: .996 seconds
```

Adding the `stem` option further reduces the word count.

```
. txttool descpt, gen(test3) stopword("stopwordexample.txt") stem
Input:   2798 unique words, 9706 total words
Output:  1552 unique words, 6188 total words
Total time: 1.35 seconds
```

Thus, of the original 2,798 unique words, 703, or 25%, were removed (or made nonunique) through cleaning; another 74, or 3%, were removed with stop-word removal; and another 469, or 17%, were removed through stemming. Though greatly reduced, the new text description is still too large for manual parsing. By bagging the words, we create 1,552 new variables, one for each unique word, and we can turn to numeric methods.

```
. txttool descpt, gen(descpt2) stopword("stopwordexample.txt") stem bagwords
> prefix(w_)
```

(*output omitted*)

1. The data are available at http://www.imf.org/external/np/pdr/mona/index.aspx.

The next step is determining which of these words best distinguish conditions that were met (`status=="M"`) from those that were not met (`status=="NM"`). Although there are many ways to determine this, a simple approach is to use correlations between each word and the status variable. A loop through the word counts can list the words with particularly high or statistically significant correlations. Given the tabular nature of the data, we use tau-b correlations. Also we want to select only words that occur frequently enough that they can potentially describe more than one instance of meeting or failing to meet a condition, so we choose words that occur in at least 5% of the total words. The loop then outputs the words and the correlations.

```
. generate status_numeric = (status=="M")
. quietly foreach x of varlist w_* {
  2. summarize `x´, meanonly
  3. if r(mean) > .05 {
  4. tab `x´ status_numeric, all
  5. if abs(r(taub)) >.05 {
  6. noisily display "`x´" r(taub)
    }
  }
}
```

Note that positive correlations are associated more often with met conditions, while negative correlations distinguish the unmet conditions.

```
w_budget.08594184
w_plan.06356041
w_law.05894236
w_fund-.07997933
w_new-.06189476
w_account-.05247162
w_ministri.05705297
w_bank-.06719572
w_includ.06220605
w_implement-.07452835
w_adopt-.05802378
```

Some of the terms uncovered by the procedure, such as “budget”, “plan”, and “law”, are especially associated with met rather than unmet conditions. We can understand why by inspecting a few of the loan conditions with a particular word:

```
. list descpt if w_law>0
```

	descpt
4.	Revocation of amendments to Article 5(3) of the Anti-Money Laundering..
26.	Fiscal impact assessments evaluating the budgetary impact of all new ..
28.	Passage by Parliament of a new Law on Labor Relations

(*output omitted*)

On the other hand, several terms stand out for distinguishing conditions that are not met as often, especially those involving funding of programs ("fund"), accounting ("account"), central banking ("bank"), and following through on new programs ("implement", "adopt"). In fact, creating anything "new" seems to be a hard condition to meet, given the word's relatively high correlation with `NM` status. The means show that the word "new" appears almost twice as often in conditions that were not met versus those that were met.

```
. summarize w_new if status=="M"
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       w_new |        419    .0692124    .2633645          0          2
. summarize w_new if status=="NM"
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       w_new |         81    .1111111    .3162278          0          1
```

While we investigate the findings provided through the dictionary, we need to determine how well the dictionary distinguishes met and unmet conditions. To do so, we can use a simple linear discriminant analysis.

```
. discrim lda w_budget w_plan w_law w_fund w_new w_account w_ministri w_bank
> w_includ w_implement w_adopt, group(status_numeric)
Linear discriminant analysis
Resubstitution classification summary
```

Key
Number Percent

True status_numeric	Classified 0	1	Total
0	38	43	81
	46.91	53.09	100.00
1	105	314	419
	25.06	74.94	100.00
Total	143	357	500
	28.60	71.40	100.00
Priors	0.5000	0.5000	

The results show that the dictionary does a respectable job at discriminating the outcomes, including finding 38 of the 81 cases of unmet conditions. In further research, loan conditions can be tagged with the dictionary words to score their difficulty, and they can be used in more comprehensive models of a borrower's conformance that could include economic conditions, political relations, and other factors.

Different methods of identifying potentially distinguishing words, such as different limits on the minimum mean, different thresholds for significance, different correlations, and so on, will likely produce different results. Producing a workable dictionary requires several trials using different methods and validations before a researcher can find the best balance of discriminatory power, number of words, and insight. However, these trials are made easier with tools that appropriately reduce and represent the text.

4 Extensions

4.1 Extension with screening

Two commands, `kountry` and `screening` (Belotti and Depalo 2010; Raciborski 2008), offer powerful text data-management capabilities by performing the tagging operation previously described; that is, they apply a coding scheme against text variables. The articles describe examples of using this procedure to standardize country codes (in the case of `kountry`) or to apply standardized medical codes to electronic patient records (in the case of `screening`).

These commands are natural extensions of the dictionary-creation capabilities created by `txttool`. The `bagwords` option can be used to apply a dictionary and create it by simply counting the words in the dictionary found in the `bagwords` counts. However, with large datasets, bagging the words can bring substantial computational overhead to an otherwise straightforward problem of counting the occurrences of particular words.

The `screening` program provides a very convenient way to address the same problem without the additional computational overhead. Returning to the example in section 3.2, we identified a list of five words that were positively correlated with status (those that identified loan conditions that were easier for countries to meet) and six words that were negatively correlated with status (those conditions that were more difficult to meet). First, we prepare the data:

```
. import delimited using "Mona.csv", clear
(20 vars, 6549 obs)
. keep if status=="M" | status=="NM"
(2521 observations deleted)
. generate status_numeric = (status=="M")
. txttool descpt, gen(descpt2) stopword("stopwordexample.txt") stem
  (output omitted)
```

We can then quickly apply the dictionary from the previous example with the `screening` command:

```
. screening, sources(descpt2) keys(fund new account bank implement adopt)
> cases(negcases)
. screening, sources(descpt2) keys(budget plan law ministri includ)
> cases(poscases)
```

We then score the loan descriptions in the data and tabulate the results:

```
. egen totneg=rowtotal(negcases*)
. egen totpos=rowtotal(poscases*)
. summarize status_numeric if totpos>totneg
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
status_num~c |       926    .8650108    .3418967          0          1
. summarize status_numeric if totpos<totneg
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
status_num~c |      1147     .81517    .3883289          0          1
```

The results indicate that our dictionary distinguishes met and unmet loan conditions on the larger dataset. A logit model indicates that the counts of both positive and negative words are statistically significant.

```
. logit status_numeric totneg totpos, nolog
Logistic regression                             Number of obs   =       4028
                                                LR chi2(2)      =       9.14
                                                Prob > chi2     =     0.0104
Log likelihood = -1785.4032                     Pseudo R2       =     0.0026

------------------------------------------------------------------------------
status_nume~c |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
--------------+---------------------------------------------------------------
      totneg |  -.1403654   .0575889    -2.44   0.015    -.2532375   -.0274932
      totpos |   .1348363   .0675538     2.00   0.046     .0024333    .2672393
       _cons |   1.655704   .0602519    27.48   0.000     1.537613    1.773796
------------------------------------------------------------------------------
```

This example shows that `screening` can complement `txttool` by quickly applying dictionaries and scoring text data. But `txttool` can also complement the `screening` command, which can remove some special characters and matching keywords with varying numbers of letters (that is, not matching on the entirety of the keyword) to allow for varying spellings and word endings. The cleaning routine in `txttool`, on the other hand, is more comprehensive because it removes all special characters.

Furthermore, `stemming` can make `screening`'s matching keys more efficient by not requiring fewer letters, thus matching the same words without introducing false positives with fewer letters. For example, to match the keyword "flag" with four letters, `screening` will match instances of "flags" and "flagged" but will also introduce potential false positives by matching instances of "flagpole", "flagellate", and "flagellum". The words "flag", "flags", and "flagged" all stem to the single stem "flag", while "flagpole", "flagellate", and "flagellum" do not. Therefore, using `txttool` to clean, stem, and remove stop words from the text first can increase the accuracy and ease of matching.

4.2 Extension to Mata

Before version 13, Stata's limit on the size of text variables placed some restrictions on the types of text data that could be analyzed. However, Mata has no such limits on the size of text and is therefore much more suitable for analyzing larger text data, including longer open-ended survey responses, comment data, or even entire documents.

The options of `txttool` are written in Mata and can be used interactively in Mata to analyze larger text fields. The options are available in the following Mata routines:

Table 1. Mata commands

Option	Mata command
`clean`	`cleantxt(`*txtfield*`)`
`stem`	`stemcolumn(`*txtfield*`)`
`stopwords()`	`stopwords(`*txtfield*`,` *filename*`)`
`subwords()`	`subwords(`*txtfield*`,` *filename*`)`
`bagwords`	`wordbag(`*txtfield*`,` *prefix*`,` *touse*`)`

Where *txtfield* is a column vector of text data, *filename* is a string scalar indicating the filename of a stop-word or subword list; *prefix* is a string scalar indicating the prefix attached to the word count variables created by `bagwords`; and *touse* is a selection vector designating which observations to write to. In addition, the routine `porterstem(`*string*`)` can be used to stem words interactively or in user-written routines. The `porterstem()` function returns only the stem of the string used as an argument; for example, `porterstem(articles)` returns `articl`.

5 Conclusion

This article introduced the `txttool` command, a text data-management suite that integrates native Stata functionality for removing characters and substituting words into a simple command to clean text and remove or substitute unwanted words. `txttool` also adds an implementation of Porter's stemming algorithm to reduce words to more useful stems and as an option for creating variables to represent the counts of individual unique words. The command is a useful foundation for text-mining tasks such as creating dictionaries and predictive models based on word frequencies. In addition, the command extends, and is extended by, the functionality of other text-management commands such as `screening`.

6 References

Belotti, F., and D. Depalo. 2010. Translation from narrative text to standard codes variables with Stata. *Stata Journal* 10: 458–481.

Benoit, K., M. Laver, and S. Mikhaylov. 2009. Treating words as data with error: Uncertainty in text statements of policy positions. *American Journal of Political Science* 53: 495–513.

Grimmer, J., and B. M. Stewart. 2013. Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political Analysis* 21: 267–297.

Laver, M., K. Benoit, and J. Garry. 2003. Extracting policy positions from political texts using words as data. *American Political Science Review* 97: 311–331.

Lowe, W., and K. Benoit. 2013. Validating estimates of latent traits from textual data using human judgment as a benchmark. *Political Analysis* 21: 298–313.

Porter, M. F. 1980. An algorithm for suffix stripping. *Program: Electronic library and information systems* 14: 130–137.

Raciborski, R. 2008. kountry: A Stata utility for merging cross-country data from multiple sources. *Stata Journal* 8: 390–400.

About the authors

Unislawa Williams is an assistant professor of political science at Spelman College, Atlanta, GA. Her research interests include international relations and forecasting.

Sean P. Williams is senior vice president at SunTrust Bank.