The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–14

textfind: A Data-driven Text Analysis Tool for Stata

Andre Assumpcao
Department of Public Policy
The University of North Carolina
Chapel Hill, NC, USA
aassumpcao@unc.edu

Abstract. `textfind` is a data-driven tool to identify and classify textual information in Stata. It makes three main contributions beyond existing text programs: (i) it uses regular expressions and Unicode encoding for complex, non-English search patterns; (ii) it produces six measures of textual match quality; and (iii) it generates variables following search patterns so that they can be used in further quantitative analysis. By using `textfind`, researchers can simultaneously employ complex search patterns and report data-driven match quality measures.

Keywords: st0001, screening, moss, ngram, txttool, term frequency-inverse document frequency, tf-idf, bag of words, topic analysis.

1 Introduction

Despite growing interest in the use of text as data across the social sciences, the low availability of text programs, the lack of consensus on analysis models, and the few solutions that integrate unsupervised machine learning and causal inference are but a few of the existing obstacles to more widespread use of text in quantitative analysis. In addition to these issues, the translation of text data into quantitative variables has suffered from unsystematic, vague classification methods that are discipline-specific and prevent their use across the social sciences.

In this article, I address some of these issues by implementing the `textfind` program. The command takes in user-defined *n*-grams and searches for such patterns in one or more variables provided to Stata. The first key feature about `textfind` is that the user can define a number of complex search patterns, such as exclusion *n*-grams and case-specific, exact, or partial match criteria – all in one line of code. The program searches substrings using regular expressions and Unicode encoding so that users are not constrained by misspelling of *n*-grams nor language-specific characters.

Secondly, `textfind` produces six statistics reporting the quality of the match, moving away from more unsystematic classification methods. For instance, instead of focusing on whether a left-leaning political group should be identified by descriptions containing the unigrams 'liberal' or 'progressive,' the researcher can actually test how much better the identification is when using two unigrams rather than one. She can rely on the six `textfind` measures to serve as data-driven evidence of her choice.

Finally, the command generates new variables containing the information from five out of the six statistics so that they can be used for further quantitative analysis, such as displayed in summary statistics or regression tables, for example.

In the following sections, I first discuss common pitfalls of using text in data analysis and how `textfind` complex search and match patterns advance analysis beyond other available solutions. In section III, I describe the syntax of the program. Finally, in section IV, I apply solutions to one hypothetical dataset and two actual datasets (one from experimental research in Malawi and another from observational research in Brazil) to show the versatility of `textfind` when working with different languages, strings containing errors, and many misspelled words. Section V concludes.

2 Classification Problems

There are many useful solutions to text analysis problems already available in Stata: `txttool` processes strings for bag of words analyses (Williams and Williams 2014); `ngram` extracts grams from strings sequences (Schonlau et al. 2017); `moss` returns counts and positions of grams following user-defined search patterns (Cox 2011); `screening` screens variables for keywords and then optionally tabulates results and generates variables containing the match criteria (Belotti and Depalo 2010). None of these programs, however, specifically deals with all problems below, which are the main contribution of `textfind`.

2.1 Case-Sensitive Match

Problem number one is when n-gram case-sensitiveness matters for classification of observations. Suppose you are using data from an open-ended survey of Chicago residents containing a question on how long does their morning commute usually take. It is entirely plausible that someone will reply "I take the Red Line to work, so about 40 minutes when it is running on schedule" while someone else will answer "I drive fast so if I leave on time and there are only a few red lights along the way, I am at work in less than 15 minutes."

Suppose further that you do not have access to a question on the type of transport, either because it was not included in the survey or the question was not provided to you; as an attentive researcher, however, you know it would be essential to control travel times by transport type. In such case, inferring transport from the matches of unigram 'red' in the answers might be a terrible idea. Unless you preserve its letter case, 'red' could refer to the Chicago Transit Authority (CTA) elevated train line or the stop sign in traffic lights across the city – meaning the respondent uses the car to commute. Thus, you would be mistakenly grouping together 15- and 40-minute commutes on different means of transportation.

Though the above example might seem silly, consider the well-established practice in political science of analyzing political text (Grimmer and Stewart 2013). A case-

insensitive search for word 'veteran' in a congressional speech in the United States could point to both 'Veteran Affairs', a government department, or an individual who has served in the U.S. armed forces. In such case, relying on the unigram 'veteran' to classify speeches in reference to the U.S. Department of Veteran Affairs (VA) without specifying that 'V' should be capitalized is again a bad idea. `textfind` solves these problems by defaulting to case-sensitive match – overriden by the `nocase` option if letter case is meaningless.

While you could use `screening` to pick up on these subtle classification differences, there are two limitations that make the program only a second best: (i) the user has to manually ask for a tabulate or count report; (ii) it does not work on Unicode text. `ngram` does have the option for case-sensitive search, but it does not match keywords; rather, it only preserves letter case of *n*-grams when processing text. In fact, these are not significant improvements over a combination of `tab` and `ustrregexm` (or any other function from the regular expression Unicode family).

2.2 Additive Match

The second common pitfall occurs when the search pattern takes in two or more grams, i.e. when we want to identify bigrams, trigrams, or any other *n*-grams. In the Chicago example from subsection 2.1, this means the researcher would like to identify the mode of transportation by requiring that pairs ('Red,' 'Line') or ('red,' 'light') both be searched for in survey answers.

There are a few ways to do this using existing programs, but they all have small shortfalls that make them less than optimal solutions. The best existing alternative is feeding a bigram, such as 'Red Line,' into `moss`. The problems of doing so are: (i) you have to know what is going on between the two unigrams you are using, such as the number of spaces and tabs; in other words, you have to focus on something else other than your main search pattern; (ii) or, in order to avoid this problem, you should be versed in regular expressions and specify option `regex` and use "`( )+`," for instance, to get around any whitespace between unigrams; (iii) finally, you would have to take an additional step to generate match result (binary) variables using `generate`. Last, (iv) if the user is working on a dataset in another language with special characters (e.g. Portuguese's *ç* or Spanish's *ñ*), she will have to specify the `unicode` option since these characters do not exist in ASCII – the default option in `moss`.[1]

When using more than one gram in the search criteria in `textfind`, the program automatically runs an additive search (e.g. 'keyword1' and 'keyword2'), in Unicode, and generates both binary and position variables if keywords are found in observation. Thus, the user does not have to worry about any whitespace or special characters in the string variable; she focuses on the search pattern and what variables she wants the

1. `screening` searches multiple keywords independently, so it cannot run additive or alternative matches. By default, `textfind` reports all independent matches and multiple keyword matches (if specified). Users could use bigrams (or any other *n*-grams) as keywords in `screening`, such as they would with `moss`, but they would still be subject to problems (i), (ii), and (iv) described above.

program to return. In fact, this is also a safer option than using bigrams because there are fewer constraints on the order and distance between both unigrams. They can come in a different order or away by as many words as possible in the textual observation.[2]

2.3 Exact Match

Another issue `textfind` helps solve is the distinction between exact and partial matches. Suppose you are working with the political speech database from subsection 2.1. This time, however, you find out that there is a table of keywords, much like in academic papers, indexing text such that military-related speeches could be simultaneously indexed by 'army,' 'air force,' 'marines,' 'Veteran Affairs,' 'veteran,' etc. Since you want to keep researching matters of the U.S. Department of Veteran Affairs (VA) exclusively but members of Congress could refer to all of these keywords in their speech, a substring match (which I call partial match here) on 'Veteran Affairs' will not be extremely helpful as you would be picking up military speeches in general instead of VA-exclusive speeches in particular.

Though (i) `strmatch("`*string*`")` and (ii) `regexm("^(`*string*`)+$")` both perform an exact match, running `textfind` is a much more efficient command; it automatically searches exact substrings in Unicode, avoiding pitfalls of non-English datasets from running (i); and it works even when the user is not familiar with regular expressions, such as it is required for running (ii). `moss` does allow for exact substring match, but it requires enabling its regular expression option and the use of search patterns from (ii), therefore it is not necessarily helpful for novice text analysts which `textfind` hopes to reach.

2.4 "Find but Exclude" Match

The last important identification feature in the command is a combined "find but exclude" match. This criterion is useful for complex classification rules that involve finding one or more grams but which, in the presence of another gram (called "exclusion" gram), should not be matched.

Let us go back to the hypothetical database of political text. Now you want to identify speeches in which the author specifically refers to the U.S. Department of Veteran Affairs, and in particular to cases where she is addressing expenditure problems. Furthermore, you want to hone in on capital expenditures (i.e. investments) but not on human resources spending. None of the string functions summarized here would help you; in fact, you would have to define various individual search and exclude criteria and concatenate them by using multiple and/or logical operators.

Loosely speaking, you would need one alternative and partial find criterion for government department ('Veteran Affairs' or 'VA'); another alternative and partial find criterion for investments ('investment' or 'capital expenditures' or 'capex'); and lastly

2. Nevertheless, for completeness purposes, `textfind` can also perform alternative matches by using option `or`, which overrides its additive match pattern.

an alternative and partial exclusion criterion ('human resources' or 'HR'). Criteria 1 and 3 should be case-sensitive because of the two abbreviations so that the search algorithm skips the sequence of characters 'va' or 'hr' in other words and only matches the appropriate abbreviations in upper case. If you are familiar with regular expressions, your code could look something like this:[3]

```
ustrregexm(varname, "([Vv]eteran [Aa]ffairs)|"VA", 0) & ///
ustrregexm(varname, "investment|(capital expenditure)|capex", 1) & ///
ustrregexm(varname, "([Hh]uman [Rr]esources)|HR", 0) == 0
```

In addition to the criteria, you would also have to define the action to take when a match is found, i.e. `tab` or `generate`. Even if you do write your search criteria in another way, you are prone to making many coding mistakes that are very time-consuming. You would have to type out three string functions, their variables, their substrings, and define case-sensitiveness three times in addition to the action you want Stata to take. With `textfind`, you only write everything out once and that is it. Moreover, the program produces a table with results on the quality of the match, allowing you to go back and forth with keywords, exclusions, and options before you settle on the best match according to statistics provided by the program.

3 Program Description

3.1 Syntax

`textfind` *varlist* [*if*] [*in*] [, `keyword(`*"string" ...*`)` `but(`*"string" ...*`)` `nocase` `exact or notable tag(`*newvarname*`) nfinds length position tfidf`]

3.2 Options

`keyword(`*"string" ...)* is the main search criteria. It looks up multiple substrings in each observation of *varlist*, where the substring could be text, numbers, or any other `ustrregexm()` search criteria.

`but(`*"string" ...)* is the main exclusion criteria. It looks up multiple exclusion substrings in each observation of *varlist*, where the substring could be text, numbers, or any other `ustrregexm()` search criteria, and removes observations that were previously matched by `keyword()`.

At least one of these options above has to be stated by the user, since there is no way to search for something that was not defined.

3. Did you notice that I have omitted the equal to one part in the first two criteria? This is also a source of problems when your code gets longer and longer.

`nocase` performs a case-insensitive search.

`exact` performs an exact search of `keyword()` in varlist and only matches observations that are entirely equal to *"string."*

`or` performs an alternative match for multiple entries in `keyword()`. The default is an additive search when the number of substrings is greater than or equal to two (e.g. *"string1"* and *"string2"*).

`notable` asks Stata not to return the table of summary statistics.

`tag(`*newvarname*`)` generates one variable called *newvar* marking all observations that were found under search criteria.

`nfinds` generates one variable per substring in `keyword()` containing the number of occurrences of *"string"* in each observation. Default variable names are *myvar1_nfinds*, *myvar2_nfinds*, ...

`length` generates new variable *myvar_length* containing the word length of each variable in varlist for which search criteria is found.

`position` generates one variable per substring in `keyword()` containing the position where *"string"* was first found in each observation. Default variable names are *myvar1_pos*, *myvar2_pos*, ...

`tfidf` generates one variable per substring in `keyword()` containing the term frequency-inverse document frequency statistic for each *"string"* in each observation. Default variable names are *myvar1_tfidf*, *myvar2_tfidf*, ...

3.3 Output

`textfind` generates six statistics displaying the quality of the search criteria defined by the user. They are:

Total Finds (exclusions): returns the number of observations found by `keyword()` or excluded by `but()`. If both have been specified, then it will find all observations for which substrings in `keyword()` were found but for which no substrings from `but()` were found. Thus, `but()` removes observations from the sample identified by `keyword()`.

Average Finds (exclusions): returns the average number of occurrences of substrings from `keyword()` [or exclusions from `but()`] for all observations in which the search criteria found substring `keyword(`*"string"*`)`.

Average Length: returns the average length (in words) of text in observations where `keyword()` [`but()`] were [not] found.

Average Position: returns the average position in which `keyword()` or `but()` were found for all observations.

Average TF-IDF: returns the average tf-idf statistic for all observations in which

`keyword()` or `but()` were found.

Means test: returns the *p*-value of a *t*-test on the difference of means across two immediate samples. It is a measure of improvement of using *n* vs. *n-1* substrings when identifying a subsample of the textual observations.

4 textfind Examples

In this section, I implement `textfind` on three different datasets to demonstrate its capabilities in addressing the pitfalls discussed in section II. The first dataset is a hypothetical collection of government functions in Neverland. I am focusing here on the ease with which we conduct text identification compared to Stata's existing solutions. Next, I move over to a real dataset of job functions in the government of Malawi collected after the "Cashgate" corruption scandal, where we experience additional classification problems from different combinations of job descriptions. Finally, I implement the command on a sample of government spending tasks in Brazil for the period 2004-2010 and use it to group expenditures as "procurement" or "public works" related. Such application demonstrates the power of `textfind` when used in non-English, error-ridden text.

4.1 Civil Servants in Neverland

In this hypothetical dataset, 5,000 civil servants hold 10 different positions in the government of Neverland. These positions are self-reported job titles, so the problem here is the identification of similar functions across government which nevertheless have different titles. For example, an 'analyst' and an 'officer' are likely performing the same function, but perhaps 'analyst' is a job title used in the Ministry of Health whereas 'officer' is used in the Ministry of Education. The output below contains a quick description of all positions in the government of Neverland.

```
. tab post
```

post	Freq.	Percent	Cum.
Analyst	527	10.54	10.54
Senior Hook Security Officer	525	10.50	21.04
analist	501	10.02	31.06
analyst	476	9.52	40.58
fairy analyst	512	10.24	50.82
fairy officer (senior level)	480	9.60	60.42
officer	504	10.08	70.50
piracy analyst	492	9.84	80.34
senior manager	507	10.14	90.48
senior piracy analyst	476	9.52	100.00
Total	5,000	100.00	

Before checking for keywords 'analyst' and 'officer,' we should note that these job titles are not always reported in the same letter case, that there are a few misspelled words, and that a combination of any of the unigrams with word 'senior' differentiates

positions even further. Therefore, using the knowledge from section 2, we would want: (i) a case-insensitive search, since letter case comes from the position of each unigram in the job title and position, here, is a meaningless information; (ii) an alternative match, because both 'analyst' or 'officer' define the same function; (iii) a "find but exclude" match, since the word 'senior,' combined with either unigram, picks up government officials in a different hierarchy than that of just 'analyst' or 'officer.' After accounting for these factors, the command below would identify the sample of interest:

```
. tab post if (ustrregexm(post, "anal[yi]st", 1) == 1 | ustrregexm(post, "officer", 1
> ) == 1) & ustrregexm(post, "senior", 1) == 0
```

post	Freq.	Percent	Cum.
Analyst	527	17.50	17.50
analist	501	16.63	34.13
analyst	476	15.80	49.93
fairy analyst	512	17.00	66.93
officer	504	16.73	83.67
piracy analyst	492	16.33	100.00
Total	3,012	100.00	

`textfind`, on the other hand, would build the following table:

```
. textfind post, key("anal[yi]st" "officer") but("senior") or nocase

Searching keyword "anal[yi]st" in variable post ...
Searching keyword "officer" in variable post ...
Searching exclusion "senior" in variable post ...

The following table displays the keyword(s) and exclusion(s) criteria used in
the search and returns six statistics for each variable specified:

Total finds:           the number of observations when criterion is met.
Average finds per obs: the average occurrence of word when criterion is met.
Average length:        the average word length when criterion is met.
Average position:      the average position of match when criterion is met.
Average tf-idf:        the average term frequency-inverse document frequency
                       when the criterion is met.
Means test p-value:    the p-value for a means comparison test across samples
                       identified by the different criteria.
```

Summary Table

```
variable:   post
n: 5000                                        Average                        Means
                    Total   ---------------------------------------        test
keyword(s)          Finds      Finds     Length   Position     TF-IDF    p-value
------------------------------------------------------------------------------------
anal[yi]st           2984         .7     1.6555     1.6555    .262111    2.0e-43
officer              1509          1    3.63419    2.36183    .567835          0
------------------------------------------------------------------------------------
Total                3012          .    1.33333          .          .    1.e-151
------------------------------------------------------------------------------------

exclusion(s):
"senior"
```

The first feature of `textfind` is the ease with which the user can search for n-grams in textual observations. The command is more efficient, more intuitive, and less

vulnerable to coding errors when compared to the combination of native commands `tab` and `ustrregexm()`. For example, if the user is not familiar with regular expressions, she could still have gotten the same results by using 'analyst' and 'analist' as different substrings to identify both correctly spelled and misspelled keywords.

In addition to total finds, the user would also have five other statistics summarizing the quality of the search criteria. She could compare keywords' average number of occurrences and textual length to get a sense of the relative importance of each keyword across job titles descriptions.

Moreover, she could even check whether the identification of a job function improves when she uses more unigrams rather than less, or when she uses exclusions words compared to when she does not. This is the information in the last column of the summary table, where `textfind` reports the p-value for a t-test across immediate sample means identified by different search criteria. Table 1 contains the (row-wise) questions that column seven helps to answer.

Table 1: Means test column interpretation

Row 1	Is the sample identified by 'anal[yi]st' the same as the original sample?
Row 2	Is the sample identified by 'anal[yi]st' or 'officer' the same as the sample identified only by 'anal[yi]st'?
Row 3	Is the sample identified by 'anal[yi]st' or 'officer' where no 'senior' is found the same as the sample identified only by 'anal[yi]st' or 'officer'?

4.2 Malawi Government Officials

The second dataset I test is from a survey of government officials in Malawi in 2016. They were asked about their corruption perceptions after corruption scandal "Cashgate" broke out in 2013 (The Economist 2014).

I focus on one variable which, much like the hypothetical Neverland dataset, contains information on officials' job function. A simple tabulation of this variable reveals that case-sensitiveness matters ('ict' and 'ICT'; 'district' and 'District') and that a common unigram is shared by different job titles ('principal human resources management' vs. 'principal economist'). These are common issues when enumerators have to transcribe respondents' answers in survey experiments. Thus, we need to use the full capability of `textfind` to properly identify job functions, since using case-sensitive and additive match both matter for the identification of job positions within the Malawian government.

Let us address case-sensitiveness first. The substring 'ICT' is an abbreviation for Information and Communications Technology. However, it is also part of unigram 'district;' therefore, if we ignore letter case in a partial search of 'ICT,' we will end up grouping together two different job functions as one.

```
. tab post if ustrregexm(post, "ict", 1) == 1
```

What is your post?	Freq.	Percent	Cum.
Assistant district registration office	1	10.00	10.00
Chief ICT Officer	1	10.00	20.00
District Commission	1	10.00	30.00
District Commissioner	1	10.00	40.00
assistant director of ICT	1	10.00	50.00
assistant director of ICT management ..	1	10.00	60.00
chief ICT officer	1	10.00	70.00
district AIDS coordinator	1	10.00	80.00
district aids coordinator	1	10.00	90.00
district lands officer	1	10.00	100.00
Total	10	100.00	

With `textfind`, we can easily do this by specifying option `but(`*"district"*`)` or using the default case-sensitive search. The two alternatives are shown below.[4] Besides the versatility of `textfind`, there are five other measures of match quality. A high TF-IDF, for instance, means that 'ICT' is an important word in the definition of a job title.

```
. textfind post, key("ICT") but("district") nocase

                                Summary Table
```

variable: post n: 524		Average				Means test
keyword(s)	Total Finds	Finds	Length	Position	TF-IDF	p-value
ICT	10	1	3.3	1.9	1.31964	2.e-296
Total	4	1	4	3	1.32037	0

```
exclusion(s):
"district"

. textfind post, key("ICT")

                                Summary Table
```

variable: post n: 524		Average				Means test
keyword(s)	Total Finds	Finds	Length	Position	TF-IDF	p-value
ICT	4	1	4	3	1.32037	0
Total	4	1	4	3	1.32037	.

```
exclusion(s):
```

Finally, we return to `textfind` for an additive search. In looking for 'principal economist' job titles, we could ask that textfind performs a search on 'principal' and 'economist,' on 'principal(.)+economist' (using regular expressions), or find all other

4. From here on, I edit out table headers to make it easy to visualize `textfind` results.

words that should be removed from a match on 'principal.' They all yield the same result, the difference being on the other quality measures reported by the program.

```
. textfind post, key("principal" "economist") nocase

                              Summary Table
─────────────────────────────────────────────────────────────────────────────
variable:   post
n: 524                                  Average                      Means
                    Total   ──────────────────────────────────────       test
keyword(s)          Finds      Finds     Length   Position     TF-IDF    p-value
─────────────────────────────────────────────────────────────────────────────
principal              88          1    3.27273    1.01136    .608404    4.5e-68
economist              10          1          2          2    2.11142    2.3e-21
─────────────────────────────────────────────────────────────────────────────
Total                   5          .          2          .          .    .025207
─────────────────────────────────────────────────────────────────────────────
exclusion(s):

. textfind post, key("principal(.)+economist") nocase

                              Summary Table
─────────────────────────────────────────────────────────────────────────────
variable:   post
n: 524                                  Average                      Means
                    Total   ──────────────────────────────────────       test
keyword(s)          Finds      Finds     Length   Position     TF-IDF    p-value
─────────────────────────────────────────────────────────────────────────────
princi..)+e~t           5   .863636          2          .    2.00884          0
─────────────────────────────────────────────────────────────────────────────
Total                   5   .863636          2          .    2.00884          .
─────────────────────────────────────────────────────────────────────────────
exclusion(s):

. textfind post, key("principal") but("account|management|officer|administ|planning|an
> alyst|system|secretary|policy|nutrition|in[dt]|human|finan|audit|insp|comm|infor|sc"
> ) nocase

                              Summary Table
─────────────────────────────────────────────────────────────────────────────
variable:   post
n: 524                                  Average                      Means
                    Total   ──────────────────────────────────────       test
keyword(s)          Finds      Finds     Length   Position     TF-IDF    p-value
─────────────────────────────────────────────────────────────────────────────
principal              88          1    3.27273    1.01136    .608404    4.5e-68
─────────────────────────────────────────────────────────────────────────────
Total                   5          1          2          1    2.32603    4.1e-33
─────────────────────────────────────────────────────────────────────────────
exclusion(s):
"account|management|officer|administ|planning|analyst|system|secretary|policy|nutritio
> n|in[dt]|human|finan|audit|insp|comm|infor|sc"
```

4.3 Government Spending in Brazil

Lastly, I implement `textfind` on a database of local government expenditure tasks between 2004-2010 in Brazil. There is a lot of variation in the description of these

expenditure tasks: some are long; some are single-word; some contain numbers and strings; etc. They also contain non-English characters, such as Portuguese's *ç*, *ã*, *ê*, etc.[5] In addition to these features, the expenditure tasks are often long text entries summarizing policy programs or activities and, as such, contain multiple errors, e.g. many whitespaces between words and paragraph/line breaks. Fortunately, however, `textfind` solves all of these issues by focusing exclusively on Unicode regular expression search criteria.

In this analysis, I am interested in assigning these expenditure tasks to one of three categories: public procurement, public works, or neither. The search criteria should take in *n*-grams associated with procurement and public works, remove observations from the each sample in which exclusion keywords are found, and generate categorical variables containing the match result. For instance, unigram 'acquisition' would belong to procurement group; 'construction' would be included in the public works group.

Should I just throw in keywords that scholars claim are associated with procurement or public works? This is precisely where one of `textfind`'s greatest contribution lies. Its six quality measures serve to guide the user on the *n*-grams that best identify each group. In particular, the means test in the last column demonstrates the incremental benefit of using more keywords rather than less for the identification of each group (as summarized in section 4.1). The lists produced below are a result of a number of interactions and words have been stemmed to their lowest possible unique root.

Table 2: Assignment Keywords

Procurement	**Public Works**
Find	**Find**
"aquisi"	"constru"
"execu"	"obra"
"ve[íi]culo"	"implant"
"despesa"	"infra(.)*estrut"
"medicamento(.)*peaf"	"amplia"
"compra"	"abasteci(.)*d(.)*[áa]gua"
"pnate"	"reforma"
"transporte(.)*escola"	"esgot"
"kit"	"m[óo]dul(.)*sanit[áa]rio"
"adquir"	"(melhoria)+(.)*(f[íi]sica)+"
	"benfeit"
	Exclude
	"psf"

5. It means I could not pre-process the strings using `txttool` because of its ASCII nature.

The output produced by `textfind` is reported below.[6] Note that each n-gram I included for identification of the procurement group improves sample identification significantly over the previous group of keywords. For the works group, if I had asssumed statistical significance only at 5%, I would have dropped the last keyword since it would not significantly identify a larger group (p-value = .083265 > .05).

```
. textfind soDescription, key(`purchase´) or nocase

                              Summary Table
------------------------------------------------------------------------------
variable:   soDescription
n: 12399                                  Average                       Means
                     Total   ---------------------------------------       test
keyword(s)           Finds      Finds     Length   Position     TF-IDF    p-value
------------------------------------------------------------------------------
aquisi                3716    1.05248     27.757    4.64909    .083934          0
execu                 2261    1.19018    47.6617    13.1729    .075388    9.e-303
ve[íi]culo             717    .712692    38.9693    11.9484    .094192    4.1e-14
despesa                667      1.006    40.8561    19.4738    .110395    5.0e-32
medica..)*p~f          570    3.36704    13.1667          .    .794461    5.5e-92
compra                 449    1.00223    5.17817    2.32294    2.30464    2.9e-96
pnate                  283          1    22.5442    21.4841    2.18641    9.0e-40
transp..)*e~a          201     1.3602    18.4925          .      .4108    1.9e-13
kit                    134    1.06716    7.83582    3.58955    1.29168    4.7e-28
adquir                  68    1.33824    29.0147      17.25    .355008    5.3e-11
------------------------------------------------------------------------------
Total                 6443          .    28.3553          .          .          0
------------------------------------------------------------------------------
exclusion(s):

. textfind soDescription, key(`works´) but("psf") or nocase

                              Summary Table
------------------------------------------------------------------------------
variable:   soDescription
n: 12399                                  Average                       Means
                     Total   ---------------------------------------       test
keyword(s)           Finds      Finds     Length   Position     TF-IDF    p-value
------------------------------------------------------------------------------
constru                954    1.02306    21.8218    4.28302    .261454          0
obra                   877    1.00342    12.7537    7.02281    1.65799    3.e-192
implant                767    1.02086     50.811     4.0013    .073898    3.e-133
infra(.)*es~t          614    .859392    88.8941         22      .0549    3.8e-73
amplia                 366          1    39.1093    6.61475    .143862    1.6e-20
abaste..)*d~a          333    .996246    31.1562          .    .175142    5.8e-23
reforma                307    1.02932    14.7036    6.31596    .429128    2.5e-08
esgot                  255    1.02353    37.0353    31.4118    .186959    .000107
m[óo]du..)*s~o         254    .752625     36.811          .    .175488    .045496
(melho..)*(~+           57    3.34373     49.193          .    .301284    .004674
benfeit                  3          1         12          3    .702908    .083265
------------------------------------------------------------------------------
Total                 2447          .     34.812          .          .          0
------------------------------------------------------------------------------
exclusion(s):
"psf"
```

6. I fed both lists to Stata using locals 'purchase' and 'works.'

5 Conclusion

In this article, I implement program `textfind` to address common pitfalls of textual data classification. It significantly improves over existing solutions in Stata by using regular expression patterns, by setting Unicode as the default search pattern, by producing six measures of match quality, and finally by building in variable generation capability so that the match results can then be fed into quantitative analysis.

`textfind`, however, is an ongoing project and I hope to improve it over time. The most important development will be reporting quality measures even when complex regular expressions are used either as `keyword()` or `but()`, which can be seen in the results in section 4.3. Second, I should work further on making it faster. The search in section 4.3, with at least nine keywords and 12,399 observations, some of which with 2556 character length 2556, took about five minutes to run. Finally, although `textfind` stores all results in matrix form for later use in word processors, a solution integrating `textfind` and `esttab` would be much more efficient.

6 References

Belotti, F., and D. Depalo. 2010. Translation from Narrative Text to Standard Codes Variables with Stata. *Stata Journal* 10(3): 458–481.

Cox, N. J. 2011. Stata Tip 98: Counting Substrings within Strings. *Stata Journal* 11(2): 318–320.

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(3): 267–297.

Schonlau, M., N. Guenther, and I. Sucholutsky. 2017. Text Mining with N-Gram Variables. *Stata Journal* 17(4): 866–881.

The Economist. 2014. The $32m Heist. *The Economist* .

Williams, U., and S. P. Williams. 2014. Txttool: Utilities for Text Analysis in Stata. *Stata Journal* 14(4): 817–829.

Acknowledgement

I thank Ciro Biderman and Brigitte Seim for providing valuable advice and text analysis problems that motivated this paper. I would also like to thank the Center of Politics and Public Economics at the Getúlio Vargas Foundation (CEPESP-FGV), the U.K. Department for International Development – Malawi office, Gerhard Anders, and Fidelis Kanyongolo for providing the data for examples in this paper. Cox (2011) created the code for n-gram position and count variables; I have only tweaked it to make it Unicode-friendly.