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

Text mining with ngram variables

Matthias Schonlau
University of Waterloo
Waterloo, ON, Canada
schonlau@uwaterloo.ca

Nick Guenther
University of Waterloo
Waterloo, ON, Canada
nguenthe@uwaterloo.ca

Ilia Sucholutsky
University of Waterloo
Waterloo, ON, Canada
isucholu@uwaterloo.ca

Abstract. Text mining is the art of turning free text into numerical variables and then analyzing them with statistical techniques. We introduce the command `ngram` which implements the most common approach to text mining, "bag of words". An ngram is a contiguous sequence of words in a text. Broadly speaking, `ngram` creates hundreds or thousands of variables each recording how often the corresponding ngram occurs in a given text. This is more useful than it sounds. `ngram` is illustrated with the categorization of text answers from two open-ended questions.

Keywords: st0001, bag of words, sets of words, unigram, gram, statistical learning, machine learning

1 Introduction

Text mining is the art of turning free text into numerical variables and then analyzing them with statistical techniques. Because the number of variables can be very large and can exceed the number of observations, statistical learning or machine learning techniques play a large role in the analysis. (Statistical learning and machine learning are synonymous reflecting important contributions by both statisticians and computer scientists to the field. We use the term "statistical learning" from here on out.)

Text mining is useful in many contexts: For example, text mining can be used to categorize or cluster answers to open-ended questions in surveys. Text mining can be used to build spam filters for emails. For example, if you have ever been solicited for money by a Nigerian prince, you will agree the presence of the two-word sequence "Nigerian prince" is a strong indicator of SPAM. Text mining can be used for authorship attribution (Madigan et al. 2005) including plagiarism. For example, it would be suspicious if two texts contain the same long sequence of words.

The most popular approach to text mining is based on ngram variables. The approach is often called "bag of words" because it simply counts word occurrences and mostly [1] [] ignores word order. (A lessor used alternative name is "set of words" referring to recording only the presence or absence of a word rather than word frequency.)

The outline of this paper is as follows: Section 2 explain how text is "numericized", i.e., how text is turned into numerical variables. Sections 3 and 4 give to examples analyzing an open-ended questions. They each feature a different statistical learning algorithm. Finally, Section 5 concludes with a discussion.

1. Unigram (1-gram) variables ignore word order completely. Higher order ngram variables partially recover word order.

2 Text mining with ngram variables

This section explains how to turn text into numerical variables for analyses. A variable containing counts of how often a single word occurs in each text is called a unigram. For example, consider the four texts shown in Table 1. Each column represents an x-variable. Each cell gives the word frequency of a word in the given text. The last column, *n_token*, records the number of words in the text.

text	4	advice	daily	fun	funny	hike	hiked	hikes	hiking	john	machu	ooooh	picchu	take	times	n_token
Take my hiking advice: Hiking is fun.	0	1	0	1	0	0	0	0	2	0	0	0	0	1	0	7
John, ooooh John, take a hike!	0	0	0	0	0	1	0	0	0	2	0	1	0	1	0	6
John is funny: he hikes daily.	0	0	1	0	1	0	0	1	0	1	0	0	0	0	0	6
He hiked to Machu Picchu 4 times.	1	0	0	0	0	0	1	0	0	0	1	0	1	0	1	7

Table 1: Example: Turning text into unigrams (single-word variables)

The variables in Table 1 were generated with the command[2]:

```
ngram text, degree(1) threshold(1)
```

The option `degree(1)` refers to unigrams. That is, variables contain counts of individual words (rather than from multi-word sequences). When words appear too infrequently, corresponding unigrams are typically discarded. Option `threshold(1)` means that all resulting variables are retained even if an individual word only appears in a single text.

Very common words are called stopwords and their presence or absence in the text is unlikely to help differentiating among the texts. By default, `ngram` removes stopwords. Table 1 does not contain variables for the stopwords "he", "is", "to", and "my".

The words "hiked", "hikes" and "hike" all have the same word root. Because these words typically refer to the same meaning, it is often useful to create only a single variable for them. This is accomplished stemming each (different) word to its (identical) stem. There are many different stemming algorithms. We have implemented the Porter stemmer (Porter 1980), which is arguably the most popular stemmer. Adding option `stemmer`, the command becomes

```
ngram text, degree(1) threshold(1) stemmer
```

The result is shown in Table 2.

The Porter algorithm strips the ending of words or changes them slightly. For example "funny" is now listed as "funni" (so that "funnier" and "funny" reduce to the same stem). All three variables related to "hike" now have the same stem, "hike", and

2. ngram is available for Windows (64-bit), Mac (64-bit), and Linux (Ubuntu, 64-bit).

text	4	advic	daili	fun	funni	hike	john	machu	ooooh	picchu	take	time	n_token
Take my hiking advice: Hiking is fun.	0	1	0	1	0	2	0	0	0	0	1	0	7
John, ooooh John, take a hike!	0	0	0	0	0	1	2	0	1	0	1	0	6
John is funny: he hikes daily.	0	0	1	0	1	1	1	0	0	0	0	0	6
He hiked to Machu Picchu 4 times.	1	0	0	0	0	1	0	1	0	1	0	1	7

Table 2: Unigrams with stemming

only one instead of three variables is created.

We started out creating unigrams, that is, single-word variables. In addition, variables can be created for each two-word sequence. Such variables are called bigrams. Table 3 contains such bigrams and the variables were created with this command:

```
ngram text, degree(2) threshold(1) stemmer
```

4	advic	daili	fun	funni	hike	john	machu	ooooh	picchu	take	time	STX_hike	STX_john	STX_take	4_time	advic_hike	daili_ETX	fun_ETX	funni_hike	hike_ETX	hike_advic	hike_daili	hike_fun	hike_machu	john_funni	john_ooooh	john_take	machu_picchu	ooooh_john	picchu_4	take_hike	time_ETX	n_token
0	1	0	1	0	2	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0	0	0	0	1	0	7
0	0	0	0	0	1	2	0	1	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	1	0	6
0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	6
1	0	0	0	0	1	0	1	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	7

Table 3: Bigrams with stemming

As you can see, the number of variables increases quite rapidly. There are also some new variables starting with STX and ending in ETX. For example, STX_hike means the word stem "hike" appears as the first non-stopword in the text. Analogously, $time\_ETX$ means that the word stem "time" appears as the last non-stopword in a text.

A threshold value specifies in how many observations an ngram needs to occur before a variable is created. We increase the threshold value to 2:

```
ngram text, degree(2) threshold(2) stemmer
```

The result is shown in Table 4. In a more realistic example one might set this threshold to a larger value. Variables that are non-zero for only a couple of observations

are not useful for making predictions. By default, the threshold is set to 5.

text	hike	john	take	STX_john	take_hike	n_token
Take my hiking advice: Hiking is fun.	2	0	1	0	1	7
John, ooooh John, take a hike!	1	2	1	1	1	6
John is funny: he hikes daily.	1	1	0	1	0	6
He hiked to Machu Picchu 4 times.	1	0	0	0	0	7

Table 4: Bigrams with Threshold 2

When the texts are short, often it is sufficient to know whether an ngram is present or absent. The count is not actually needed. This can be accomplished with the option "binarize" as follows:

```
ngram text, degree(2) threshold(2) stemmer binarize
```

The result is shown in Table 5.

text	hike	john	take	STX_john	take_hike	n_token
Take my hiking advice: Hiking is fun.	1	0	1	0	1	7
John, ooooh John, take a hike!	1	1	1	1	1	6
John is funny: he hikes daily.	1	1	0	1	0	6
He hiked to Machu Picchu 4 times.	1	0	0	0	0	7

Table 5: Bigrams with Threshold 2 and Binarizing

We have described the core of a simple but powerful mechanism to turn text into variables. We next address how this works in foreign languages.

2.1 Internationalization

Language settings affect `ngram` for the removal of stopwords and for stemming. By default, `ngram` uses the language specified in the current locale. This can be reset to any other language. For example, to set the current locale to English specify

```
set locale_functions en
```

ngrams can be created for any language.

Stopwords

Stopwords can be removed for any language. We have included stopword files for all languages for which a stemmer is implemented. For other languages, lists of stopwords are readily available on the Internet and can easily be added by specifying the file `stopwords_la.txt` on the `adopath` (e.g. in the current folder) where "la" is the two letter language code of the current locale. If you want to see what words are considered stopwords, the default file for English is here: `C:\ado\plus\s\stopwords_en.txt`.

Stemming

An ideal stemmer stems all words that are derived from the same root to the same stem. Understemming refers to creating word groups that still have multiple roots, i.e., the groups are not separated enough. Overstemming refers to creating more than one group for words with the same root, i.e., groups are separated more than needed. The Porter stemmer is considered a light stemmer (Kraaij and Pohlmann 1994, e.g.). Typically, light stemming leads to better results than heavy stemming (Paice 1994).

In European languages, stemming can lead to large gains (Manning et al. 2008, p.32). The extent of the benefit varies by language and different evaluations have reported different gains, however. (Hollink et al. 2004) investigated Porter stemmers in eight European languages. While stemming always improves average precision, they found that there were much larger gains in some languages than in others. Noticeably, there were large gains for Finish (30%), Spanish (10.5%) and German (7.3%) , middling gains in English (4.9%) and Italian(4.9%) and smaller gains in Dutch (1.2%), French (1.2%) and Swedish (1.7%). (Savoy 2006, p.1034) reports a Porter stemmer improved the mean average precision on average by 30.5% for French, 7.7% for Portuguese, and 12.4% for German. (Gaustad and Bouma 2002) found that a Dutch stemming did not consistently improve prediction accuracy. Hull et al. (1996) conclude in English "some form of stemming is almost always beneficial".

Stemming is available for the following locales: da (Danish), de (German), en (English), es (Spanish), fr (French), it (Italian), nl (Dutch), no (Norwegian), pt (Portuguese), ro (Romanian), ru (Russian), and sv (Swedish). The language specific Porter stemmers we use are described on http://snowball.tartarus.org/ .

2.2 Additional program options

Option `n_token` adds a variable with the number of words of the original text (i.e., with stopwords). This variable is often very useful and is supplied by default.

All text is first converted into lower case by default. This behavior can be disabled

by specifying option `nolower`.

Option `punctuation` adds variables for punctuation. For example, a separate unigram is created for the symbol "?" if it appears. What is considered punctuation is language dependent.

2.3 Statistical/ Machine learning

After turning the text into numerical variables the number of variables are typically large and sometimes larger than number of observations. If so, linear and logistic regression will fail because they require estimating more parameters than observations. There is also a high degree of collinearity in the x-matrix as most values are 0: most words are not found in most texts. Because of these reasons we turn to statistical learning. In STATA there are at least two alternatives: boosting (Schonlau 2005) which implements the MART algorithm for gradient boosting (Hastie et al. 2005) and support vector machines (Guenther and Schonlau 2016) invented in the 90ies by Vapnik (Vapnik 2013) .

3 Beliefs about Immigrants

As part of their research on cross-national equivalence of measures of xenophobia, Braun et al. (2013) categorized answers to open-pended questions on beliefs about immigrants. The questions were part of the the 2003 International Social Survey Program (ISSP) questionnaire on National Identity. The authors investigated identical questions asked in multiple countries (in different languages); for simplicity we focus here on the German survey with 1006 observations. The questionnaire contained 4 statements about immigrants such as

> "Immigrants take jobs from people who were born in Germany".

Respondents were asked to rate this statement on a 5-point Likert scale ranging from "strongly disagree" to "strongly agree". After each of the four questions respondents were then probed with an open-ended question:

> "Which type of immigrants were you thinking of when you answered the question? The previous statement was: [text of the respective item repeated]."

This question is then categorized by (human) raters into the following categories (Braun et al. 2013):

- General reference to immigrants
- Reference to specific countries of origin/ethnicities (Islamic countries, eastern Europe, Asia, Latin America, sub-Saharan countries, Europe, and Gypsies)

- Positive reference of immigrant groups ("people who contribute to our society")
- Negative reference of immigrant groups ("any immigrants that[. . .] cannot speak our language")
- Neutral reference of immigrant groups "immigrants who come to the United States primarily to work")
- Reference to legal/illegal immigrant distinction ("illegal immigrants not paying taxes")
- Other answers ("no German wants these jobs")
- Nonresponse or incomprehensible / unclear answer ("its a choice")

3.1 Creating ngram variables

The text variable is called *probe_all*. We specify ngrams of degree 2 (unigrams and bigrams), stemming, and set the usual threshold (5):

```
. set locale_functions de
. ngram   probe_all, degree(2) threshold(5)  stemmer  binarize
```

The first line, `set locale_functions de`, signals that German stemming and the German stopword list are to be used.

`ngram` created 242 ngrams and a variable that counts the number of words, *n_token*. How much does the number of variables change as a function of degree? By changing the `degree` option and rerunning the program we can find out: There are 167 unigrams (degree(1)). As we already know, the number of variables rises to 242 when including bigrams (degree(2)). The number further rises to 256 when including trigrams (degree(3)).

Out of curiosity, we briefly assess the effect of stemming and stopwords on the number of variables for these data. Stemming reduced the number of variables only by 19 variables. When specifying bigrams (degree(2)) *without* stemming the number of variables reduces from 242 to 223. Removing stopwords has a substantial effect: When specifying bigrams with stemming but not removing stopwords (`stopwords(.)`), the number of variables increases from 242 to 423.

3.2 Statistical Learning

Next, we applied statistical learning techniques to the ngram variables. Here we use the STATA implementation (Schonlau 2005) of (gradient) boosting (Hastie et al. 2005). (The term "boosting" is used differently in statistics and computer science. In Statistics this term refers to the MART algorithm popularized by (Hastie et al. 2005). Computer scientists think of "boosting" ia a generic term for a large number of related algorithms).

Statistical learning models are black-box models and generally difficult to interpret. In boosting the concept of "influential variables" makes interpreting the black-box model a little easier. We choose 500 observations as training data identified by the variable *train* and compare several runs with and without stemming in German, removal of German stopwords, and binarizing of count variables.

```
boost  y t_* n_token if train, dist(multinomial) influence pred(pred) ///
     seed(12) interaction(3) shrink(.1)
```

n_token is the number of words in the text answer by default all ngrams have the prefix *t_* making it easy to specify all ngrams. The options `interaction(3) shrink(.1)` refer to boosting-specific tuning parameters that are not further explained here. The option `seed` passes a seed for the random generation of numbers, making results reproducible.

The accuracy of predicted categories on the test data is shown in Table 6. The first row represents typical choices: stemming, stopwords and binarizing. Surprisingly, the runs that do not remove stopwords have the highest accuracy. We will comment on this later.

German Stemming	Remove German Stopwords	binarize	Accuracy
yes	remove	yes	61.9 %
yes	remove	no	62.5 %
no	remove	yes	61.9 %
no	keep	yes	68.2%
yes	keep	yes	71.2%

Table 6: Performance metrics for boosting the Immigrant data for different options

We proceed with the run with the highest accuracy in the last row of Table 6. For this choice, Table 7 shows the breakdown of accuracy by response category. For the largest two categories, "general" and "type of foreigner" as well as for category "nonproductive" accuracy is very high. Accuracy is low for smaller categories ("positive", "negative", "legal-illegal", "neutral") and middling for "other".

y	Accuracy	N
positive	25%	36
negative	30%	20
neutral	40%	10
general	81%	144
type of foreigner	86%	146
legal /illegal	0%	1
other	63%	110
non-productive	82%	38

Table 7: Immigrant data: Accuracy by response category on the test data.

Each of the eight response categories in multinomial boosting is associated with a set of influential variables. Influences are measured in percentages and the influence of all variables sum to 100%. The influential variables for outcome "general" is shown in Figure 1. The most influential variable is "all" (with the same meaning in English). The second most influential variable, "allgemein", means "general" in English. The bigram "kein_bestimmt" translates to "no particular" as in "no particular type of foreigner". Several other influential variables refer to general groups of foreigners such as stemmed words of nationality, and foreigners.

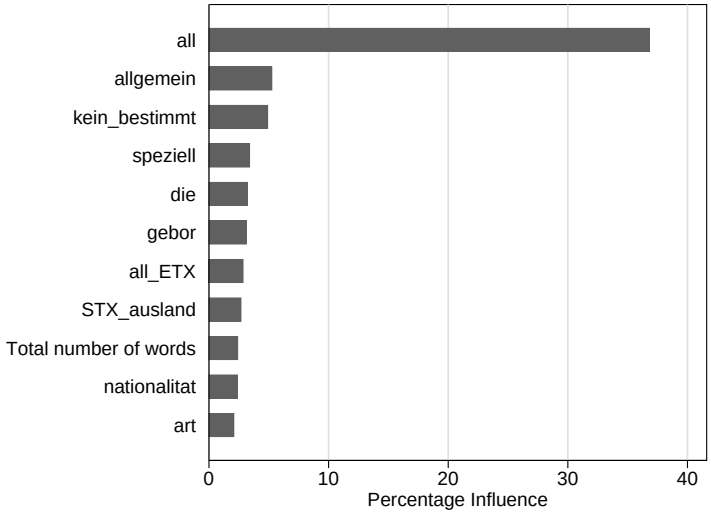

Figure 1: Influential variables for outcome "general". Variables with influence of less than 2% are not shown.

Influential variables for outcome "non-productive" are shown in Figure 2. Because non-productive answers are typically very short (a word or two), the most influential variable in determining whether the entry is non-productive is whether or not the line is empty: STX_ETX is the bigram where the beginning-of-line is followed by the end-of-line. The second most important variable is the number of words: nonproductive non-empty lines such as "-", "." and "???" have zero number of words. $kein\_ETX$ and $nicht\_ETX$ refer to the words "kein" (no, none) and "nicht" (not) appearing as the last word in the text. Looking at the stopword file it turns out that several influential words for this outcome were all stopwords: "kein" (no, none) , "nicht" (not) , "ich" (I) and "kann" (can) . This explains why it was a bad idea to exclude stopwords in this case. This is unusual. In most studies it is a good idea to remove stopwords.

In conclusion, with 8 different outcome categories prediction accuracy on a test data set was just over 70%. Retaining stopwords was important here.

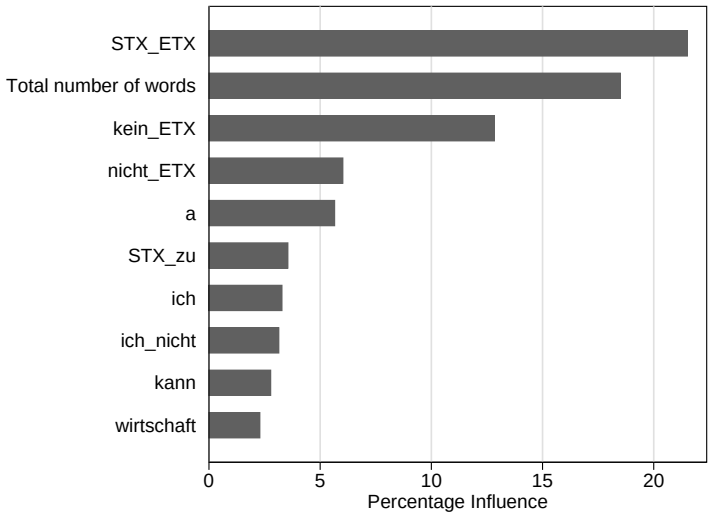

Figure 2: Influential variables for outcome "nonproductive". Variables with influence of less than 2% are not shown.

4 Example: Final comments in Surveys

Many surveys include an open-ended question near the end such as this: "Do you have any other comment?" A subset of such comments in the LISS panel and the Dutch Immigrant panel have been categorized into eight mutually exclusive types of comment: survey was difficult, survey contained an error, survey was too long, survey did not apply to me, survey or questions were unclear, negative comment (not otherwise categorized, positive comment, and neutral comment (Schonlau 2015). The survey was conducted in Dutch. The data consist of 699 categorized answers and 1400 uncategorized answers. The frequencies of the categorized answers are given in Table 8. Neutral comments dominate; there are three categories with low frequency ($\leq$ 11) making it harder to build a model to predict them. The goal is to a) create ngrams from the comments, b) use those ngrams to build a model, and c) use the model to predict the content categories on uncategorized comments.

We first create unigrams of words that occur in at least 5 answers. Dutch stopwords are removed, the words are stemmed to their Dutch root, and indicator variables (rather than word counts) are used to record when a text contains a word. The texts are substantially longer than in the previous example.

```
set locale_functions nl
ngram eva_comment, degree(1) threshold(5) stemmer   binarize
```

This creates about eight hundred unigrams as well as the variable `n_token` containing the number of words in the answer. Now we are ready to run a statistical/machine

Comment Category	**Freq**	**Percent(%)**
neutral comment	306	43.8
negative comment	196	28.0
survey difficult	73	10.4
questions unclear	72	10.3
does not apply	23	3.3
survey contains error	11	1.6
positive comment	10	1.4
survey too long	8	1.1
total	**699**	**100.0**

Table 8: Frequencies of categorized answers shown in descending order by frequency

learning algorithm on the data. Here we choose support vector machines (Guenther and Schonlau 2016):

```
svmachines category n_token t_*  if category!=., type(svc) kernel(linear) c(1)
predict  cat_pred
```

The dependent variable `category` contains 8 different values, one for each category. Uncategorized data are excluded with the `if` statement. Specifying a linear kernel for the support vector machines is usually sufficient in text mining. The linear kernel requires only one tuning parameter, c.

With learning algorithms it is important to find good values for tuning parameters. Splitting the data into random training data and test data sets, we try a wide range of choices for the c parameter: $100, 10, 1, 0.10.01, 0.001$. The c-parameter that gives the best performance on the test data set happens to be the default value $c = 1$. The accuracy with a training /test split of 500/199 is 0.608, i.e. 60.8% of predictions were correct. This is not a fabulous result but still quite a bit better than predicting the modal category "neutral comment" which would have had an accuracy of 43.8%. This highlights that it is important to establish for each application anew how well text mining is working.

We now look at predictions of the uncategorized 1400 observations. The distribution of the predicted categories is shown in Table 9. Relative to the distribution in Table 8, more common categories tend to be overpredicted at the expense of rare categories. It makes sense that in case of doubt the more common category is chosen. However, in the aggregate this distorts the distribution towards such categories.

The distortion in the distribution can be corrected by averaging the probabilities of predicting each category rather than the average number a category is predicted. Even if a rare category is never the most likely for any one answer, small probabilities add up. To obtain the probabilities SVM is rerun with the probability option for both model building and prediction:

```
svmachines category n_token t_*   category!=. , type(svc) kernel(linear) c(1) prob
predict p prob
summarize  p*
```

The average probabilities are shown in the last column of Table 9. The distribution based on the sum of probabilities is strikingly similar to the distribution of the random sample in Table 8.

Comment Category	**Freq**	**Predictions(%)**	**Av. prob (%)**
neutral comment	740	52.9	42.0
negative comment	412	29.4	27.9
survey difficult	119	8.5	10.7
questions unclear	78	5.6	10.4
does not apply	32	2.3	3.8
survey too long	9	0.6	1.5
survey contains error	7	0.5	1.7
positive comment	3	0.2	1.7
total	**1400**	**100**	**100**

Table 9: Predicted categories of 1400 previously uncategorized answer texts.

Sometimes the categorizations will be used as x-variables in a subsequent regressions. McCaffrey and Elliott (2008) addressed a similar problem in the context of multiple imputation. Their recommendation was to use estimated probabilities rather than indicator variables of predicted categories as x-variables. The same recommendation applies here.

5 Discussion

The ngram approach to text mining is not a panacea. It tends to work better for shorter texts because in shorter texts the presence and absence of words is more remarkable than in longer texts containing many words.

A related STATA program, `txttool` (Williams and Williams 2014), also creates unigrams, removes stopwords and uses the Porter algorithm for stemming. The program introduced here, `ngram`, is more comprehensive than `txttool` in that it also allows for higher order ngram variables (i.e., bigrams and trigrams in addition to unigrams), stemming in languages other than English, and the use of punctuation (e.g. question marks) as ngram variables. However, unlike `ngram`, the `txttool` program is written in Mata.

We hope we have provided a useful entry point to text mining using this approach. Text mining extends far beyond what we have been able to cover. There are many book length treatments. Manning and Schütze (1999) is a classic book written by computer scientists. Manning et al. (2008) and Büttcher et al. (2016) are two popular books in information retrieval (e.g. searches in a search engine such as Google). Ignatow and Mihalcea (2016) is specifically aimed at social scientists. Jockers (2014) gave extensive

examples in *R* aimed at students of literature.

Acknowledgment

This research was supported by the Social Sciences and Humanities Research Council of Canada (SSHRC # 435-2013-0128 and # 430-2013-0301) . This paper uses data from the LISS (Longitudinal Internet Studies for the Social sciences) panel administered by CentERdata (Tilburg University, The Netherlands). Dr. Dorothée Behr kindly provided us with the GESIS immigrant data. We are grateful for the data. We thank an anonymous referee for his/her comments.

6 References

Braun, M., D. Behr, and L. Kaczmirek. 2013. Assessing cross-national equivalence of measures of xenophobia: Evidence from probing in web surveys. *International Journal of Public Opinion Research* 25(3): 383–395.

Büttcher, S., C. L. Clarke, and G. V. Cormack. 2016. *Information retrieval: Implementing and evaluating search engines*. Cambridge, Massachusetts: MIT Press.

Gaustad, T., and G. Bouma. 2002. Accurate stemming of Dutch for text classification. *Language and Computers* 45(1): 104–117.

Guenther, N., and M. Schonlau. 2016. Support Vector Machines. *The Stata Journal* 16(4): 917–937.

Hastie, T., R. Tibshirani, and J. Friedman. 2005. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. 2nd ed. Heidelberg: Springer.

Hollink, V., J. Kamps, C. Monz, and M. De Rijke. 2004. Monolingual document retrieval for European languages. *Information retrieval* 7: 33–52.

Hull, D. A., et al. 1996. Stemming algorithms: A case study for detailed evaluation. *Journal of the American Society of Information Science* 47(1): 70–84.

Ignatow, G., and R. F. Mihalcea. 2016. *Text Mining: A Guidebook for the Social Sciences*. Thousand Oaks, California: SAGE.

Jockers, M. L. 2014. *Text Analysis with R for Students of Literature*. Heidelberg: Springer.

Kraaij, W., and R. Pohlmann. 1994. Porters stemming algorithm for Dutch. In *Wetenschappelijke bijdragen aan de derde STINFON Conferentie, Tilburg, 1994*, ed. N. LGM and de Vroomen WAM, 167–180.

Madigan, D., A. Genkin, D. D. Lewis, S. Argamon, D. Fradkin, and L. Ye. 2005. Identification on the large scale. In *Proc. of the Meeting of the Classification Society of North America*.

Manning, C., P. Raghavan, and H. Schütze. 2008. *Introduction to Information Retrieval*. Cambridge, England: Cambridge University Press.

Manning, C. D., and H. Schütze. 1999. *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts: MIT Press.

McCaffrey, D. F., and M. N. Elliott. 2008. Power of Tests for a Dichotomous Independent Variable Measured with Error. *Health Services Research* 43(3): 1085–1101.

Paice, C. D. 1994. An evaluation method for stemming algorithms. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, 42–50. Springer.

Porter, M. F. 1980. An algorithm for suffix stripping. *Program* 14(3): 130–137.

Savoy, J. 2006. Light stemming approaches for the French, Portuguese, German and Hungarian languages. In *Proceedings of the 2006 ACM symposium on Applied computing*, 1031–1035. ACM.

Schonlau, M. 2005. Boosted regression (boosting): An introductory tutorial and a Stata plugin. *The Stata Journal* 5(3): 330–354.

———. 2015. What do web survey panel respondents answer when asked "Do you have any other comment?". *Survey Methods: Insights from the Field* http://surveyinsights.org/?p=6899.

Vapnik, V. 2013. *The Nature of Statistical Learning Theory*. Heidelberg: Springer.

Williams, U., and S. P. Williams. 2014. txttool: Utilities for text analysis in Stata. *Stata Journal* 14(4): 817–829.

About the authors

Matthias Schonlau, Ph.D., is professor in the Department of Statistics and Actuarial Sciences at the University of Waterloo in Canada. His interests include survey methodology and text mining of open-ended questions.

Nick Guenther is an undergraduate in the Department of Statistics and the School of Computer Science at the University of Waterloo in Canada. His interests include feature learning, functional programming, and applying machine learning towards new technologies.

Ilia Sucholutsky is an undergraduate student in the Department of Statistics at the University of Waterloo in Canada. His interests include deep learning and discovering the impact it may have when applied to new problems.