The Stata Journal (2023)
23, Number 3, pp. 799–812 DOI: 10.1177/1536867X231196349

Estimating text regressions using txtreg_train

Carlo Schwarz
Bocconi University
Milano, Italy
carlo.schwarz@unibocconi.it

Abstract. In this article, I introduce new commands to estimate text regressions for continuous, binary, and categorical variables based on text strings. The command `txtreg_train` automatically handles text cleaning, tokenization, model training, and cross-validation for lasso, ridge, elastic-net, and regularized logistic regressions. The `txtreg_predict` command obtains the predictions from the trained text regression model. Furthermore, the `txtreg_analyze` command facilitates the analysis of the coefficients of the text regression model. Together, these commands provide a convenient toolbox for researchers to train text regressions. They also allow sharing of pretrained text regression models with other researchers.

Keywords: dm0112, txtreg_train, txtreg_predict, txtreg_analyze, text regressions, machine learning, text analysis

1 Introduction

In recent years, natural language processing (NLP) has risen to increased prominence in the social sciences. This rise was driven not only by increases in computing power and data availability but also by the enormous amount of previously unaccessible information that is contained in text form. A widely used approach in NLP is so-called text regressions (see Gentzkow, Kelly, and Taddy [2019] for a discussion of articles). Text regressions use the words in texts as predictors ($\mathbf{X}$) for an outcome (y). The great flexibility of this approach comes from its applicability to any form of text data and allows for the automatic prediction of outcome variables. Text regressions are also often used to impute variables in new datasets. For example, it is possible to hand code a subset of data and then extend the coding to the entire dataset. Alternatively, text regressions can learn relationships between an outcome and text in one dataset and create a new outcome variable for other data for which this information is lacking.

In this article, I introduce three commands for text regressions. First, `txtreg_train` trains text regression models and then stores them for later use or for sharing with other researchers. Second, the `txtreg_predict` command predicts outcomes based on pretrained models. Last, `txtreg_analyze` facilitates the analysis of the coefficients of the text regression model. This enables the investigation of relationships between words and the outcome variable that the text regression has derived.

In this way, the commands extend the NLP capabilities of Stata. Among others, Stata already can calculate the Levenshtein distance (Barker and Pöge 2012), perform latent Dirichlet allocation (Schwarz 2018), and perform latent semantic analysis (Schwarz 2019). The closest related commands are the `txttool` command by Williams

and Williams (2014) and the `ngram` command by Schonlau, Guenther, and Sucholutsky (2017), both of which can tokenize text for use in statistical analysis. The presented commands provide at least three advantages. First, they utilize Stata's Python integration, which was introduced in Stata 16 and enables the use of highly optimized Python packages.[1] Second, the commands provide an integrated pipeline that should be suitable for the most frequent uses of text regressions. Last and most importantly, the commands provide an easy way to store trained text regression models. Because the extrapolation of outcome variables beyond the training data is one of the most frequent uses of text regressions, this opens up new applications. The stored text regression models can also be shared with other researchers without sharing the underlying raw data.

The rest of the article proceeds as follows. Section 2 provides a short introduction to text regressions. Section 3 describes the commands. Section 4 illustrates the use of text regressions for the prediction of citations. Section 5 concludes.

2 Predicting numerical variables using text

Consider a setting in which a researcher wants to predict a dependent variable (y) based on information contained in a string variable. Text regressions approach this problem by using the words in the text strings as the predictor variables ($\mathbf{X}$). More specifically, text regressions minimize the mean squared deviation from a dependent variable y based on the observations $i \in N$,

$$\min_{\beta_j \in \mathbb{R}^V} \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{V} \beta_j \times w_{i,j} \right)^2 \tag{1}$$

where $w_{i,j}$ are the individual text features and β_j are the regression coefficients of the text regression. The text features are constructed from the text string by creating a bag-of-words representation, which represents the text as a document-n-gram-matrix $\mathbf{W}$. The term "n-grams" refers to continuous sequences of n-words from the original text string, where n indicates the length of the sequence. Hence, 1-grams (unigrams) are simply the sets of words that appear in the text. Similarly, 2-grams (bigrams) and 3-grams (trigrams) are the sets of two- and three-word phrases that appear in the text. The dimensions of the matrix $\mathbf{W}$ are $N \times V$, where N is the number of observations and V is the size of the vocabulary (that is, the number of unique n-grams). The entries $w_{i,v}$ in $\mathbf{W}$ represent the number of times an n-gram $v \in V$ appears in observation $i \in N$:

1. Because the commands use Stata's Python integration, they require at least Stata 16 to run. The commands further require a working Python installation. A Python installation that includes the required packages is, for example, Anaconda. In particular, the commands rely on the Numpy v1.21.5, Scikit v1.0.2 (Pedregosa et al. 2011), Pandas v1.4.4 (The Pandas Development Team 2020), and NLTK v1.21.5 (Bird, Klein, and Loper 2009) packages.

$$\underbrace{\mathbf{W}}_{N\times V} = \begin{pmatrix} w_{1,1} & \cdots & w_{1,V} \\ \vdots & \ddots & \vdots \\ w_{N,1} & \cdots & w_{N,V} \end{pmatrix}$$

Because the matrix **W** will form the main input for the text regression, it is important to decide which n-grams should be included in the matrix. The following three steps have become standard in text regressions. First, very frequent words (stopwords) that add little to the meaning of texts (for example, "I", "he", or "and") are removed upfront. Frequently, thresholds are also used for the minimum and maximum numbers of occurrences of n-grams across different observations. Second, words are reduced to their morphological roots using a process called stemming. For example, stemming will change "walking" and "walked" to "walk" so that these words represent the same unigram. Last, the matrix **W** is often reweighted using term frequency-inverse document frequency (tf–idf), which replaces $w_{i,v}$ by $\text{tf–idf}(w_{i,v}) = \{1+\log(w_{i,v})\} \times [\log\{(1+N)/(1+n_v)\}+1]$, where n_v is the number of observations in which the n-gram v appears. The tf–idf reweighting reduces the weights of n-grams that appear in many observations and therefore contribute less to the meaning of texts.

While we could estimate a text regression using (1), it should be apparent that the dimensions of matrix **W** can become very large because of the growing size of the vocabulary, particularly when trigrams are used. It is also common that $V > N$, which makes it impossible to estimate an ordinary least-squares regression. Even in cases where $V < N$, it might not be advisable to estimate an ordinary least-squares regression, because the regression might fit idiosyncratic features of the training data. To overcome these limitations, the machine learning literature uses regularization and k-fold cross-validation (see Hastie, Tibshirani, and Friedman [2009] for additional details). Regularization implies the inclusion of an additional term in the cost function C [see (1)], which incorporates a penalty for the size of the coefficients β_j. The most frequently used regularized regression models and their respective penalties are lasso ($\lambda \sum_{j=1}^{V} |\beta_j|$), ridge ($\lambda \sum_{j=1}^{V} \beta_j^2$), and elastic net ($\lambda_1 \sum_{j=1}^{V} |\beta_j| + \lambda_2 \sum_{j=1}^{V} \beta_j^2$). The user-chosen parameters λ_i specify the strength of regularization.[2] The optimal regularization strength, in turn, is chosen using k-fold cross-validation. This process involves randomly splitting the data into k slices. The model is then trained on $k-1$ slices, and the model's performance is evaluated on the remaining data slice, for example, the model's R^2. The model with the best out-of-sample performance determines the optimal regularization strength. In other words, the regularization strength is chosen such that the text regression model best explains the variation in unseen data.

2. When the dependent variable y is binary or categorical, it is similarly possible to include regularization terms in logistic or multinomial logistic regression models.

3 Implementation

This section describes how the `txtreg_train`, `txtreg_predict`, and `txtreg_analyze` commands automate the training of text regressions in Stata. If no pretrained model is used, the user should start with the `txtreg_train` command.

3.1 Training of text regression using txtreg_train

The `txtreg_train` command handles the training of text regression models. The user needs to specify the dependent variable and the predictor variable. The dependent variable can be continuous, binary, or categorical, while the predictor variable should always be a string variable. The rest of the training process is automatically handled by the command. In particular, the command performs the following steps:

1. The command cleans the text and creates the document-n-gram-matrix. If specified, the command will remove stopwords, stem the text, and use tf–idf reweighting.
2. Afterward, the data are split into a training sample (90%) and a test sample (10%).
3. The text regression model is trained on the training sample, and the optimal regularization strength is chosen using 10-fold cross-validation.
4. The command reports the optimal regularization strength and the out-of-sample performance of the text regression based on the test sample.

The provided options control the text preparation process and the training of the model.

3.1.1 Syntax

`txtreg_train` *depvar stringvar* `using` *filename* [`,` `model(`*string*`)`
`regularization(`*string*`)` `regu_range(`*string*`)` `seed(`*integer*`)` `tfidf` `stem`
`stem_lang(`*string*`)` `stopwords(`*string*`)` `ngrams(`*integer*`)` `min_freq(`*integer*`)`
`max_freq(`*real*`)` `max_voc(`*integer*`)`]

`using` *filename* specifies the path and the name for storing the trained text regression model. If `using` is not specified, a file called `txtreg_model.pkl` will be generated in the current working directory.

3.1.2 Options

`model(`*string*`)` specifies whether a least-squares or a logistic regression will be used to train the model. Least-squares regression is used by specifying `reg`. To use logistic

regression, specify `logit`. By default, a least-squares regression will be used. The command will automatically train a multinomial logistic model if the dependent variable is categorical and `logit` is specified.

`regularization(`*string*`)` specifies which form of regularization should be used. The options are `lasso`, `ridge`, or `elasticnet`. The default is `regularization(ridge)`.

`regu_range(`*string*`)` specifies which values of the regularization parameter will be tested in the 10-fold cross-validation. The string should give the start and end values separated by a comma. The default is `regu_range("0.1,1")` so that values between 0.1 and 1 in steps of 0.1 are used. In least-squares regressions, larger values imply stronger regularization. For logistic regression, smaller values imply stronger regularization.[3]

`seed(`*integer*`)` specifies the seed for the random-number generator, which, among others, determines the splits for the cross-validation. By default, the random-number generator starts from its previous state.

3.1.3 Text cleaning options

`tfidf` specifies whether tf–idf should be used. If so, the document-n-gram-matrix is reweighted by tf–idf before training the text regression model.

`stem` specifies whether the words should be stemmed before fitting the text regression model. This will reduce the words to their morphological roots (for example, "walked" to "walk"). The stemming implementation relies on Python's NLTK package (Bird, Klein, and Loper 2009).

`stem_lang(`*string*`)` specifies the language of the text strings so that the appropriate stemmer can be used. For a list of supported languages, see the NLTK website.[4] The default language is `stem_lang("english")`. This option is needed only if `stem` is used.

`stopwords(`*string*`)` specifies a list of words to exclude from the text regression. Usually, highly frequent words such as "I" and "you" are removed from the text because these words do not help with the classification of the documents. Predefined stopword lists for different languages are available online (see https://code.google.com/archive/p/stop-words/).

`ngrams(`*integer*`)` specifies which order of n-grams should be included in the text regression. For example, specifying `ngrams(2)` implies the use of unigrams and bigrams. `ngrams(3)` additionally uses trigrams. By default, only unigrams are used.

3. In the case of elastic-net regularization, the two λ parameters are set jointly through the use of a mixing parameter. The mixing parameter of lasso to ridge regularization is set to 0.5, such that the elastic-net penalty will be $0.5 \times \lambda \sum_{j=1}^{V} |\beta_j| + 0.5 \times \lambda \sum_{j=1}^{V} \beta_j^2$.
4. https://www.nltk.org/__modules/nltk/stem/snowball.html

`min_freq(`*integer*`)` allows the removal of words that appear in few documents. Words that appear in fewer documents than *integer* will be excluded from the text regression. The default is `min_freq(0)`.

`max_freq(`*real*`)` allows the removal of words that appear frequently in documents. Words that appear in more than a proportion of *real* $\in [0, 1]$ of the documents. The default is `max_freq(1)`.

`max_voc(`*integer*`)` specifies the maximum number of n-grams to be included in the text regression. By default, the vocabulary will not be restricted.

3.1.4 Output

The `txtreg_train` command generates a new file containing the trained text regression model. The filename and path are specified in `using`. If `using` is not specified, a file called `txtreg_model.pkl` will be generated in the current working directory.

3.2 Making use of trained text regression models with txtreg_predict

After one either trains a text regression model using `txtreg_train` or obtains a pretrained text regression model, one can use the `txtreg_predict` command to predict the outcome variable. For the prediction of the outcome, `txtreg_predict` uses text strings as the predictor variable. The advantage of this separation between the training and prediction processes is that it is possible to 1) apply the predicted citation model to a dataset different from the training data and 2) share the trained model with other researchers. The syntax of `txtreg_predict` is shown below.

3.2.1 Syntax

`txtreg_predict` *indepvar* `using` *filename* [`, name_new_var(`*string*`) stem stem_lang(`*string*`)`]

`using` *filename* specifies the location and the name of the pretrained text regression model.

3.2.2 Options

`name_new_var(`*string*`)` specifies the name of the variable created by `txtreg_predict`. The user should ensure that `name_new_var()` is not yet used in the dataset. The default is `name_new_var("predict_")`.

`stem` specifies whether the words should be stemmed before estimation of the text regression model. This will reduce the words to their morphological roots (for example, "walked" to "walk"). The option should be specified if the pretrained text regression model uses stemming.

`stem_lang(`*string*`)` specifies the language of the text strings. For a list of supported languages, see https://www.nltk.org/_modules/nltk/stem/snowball.html.

3.2.3 Output

`txtreg_predict` creates a new variable with the name specified by `name_new_var()` containing the predictions from the text regression model.

3.3 Analyzing text regression with txtreg_analyze

When one uses text regressions, it is important to ensure that the derived relationships between n-grams and the outcome are sensible. This can be achieved by analyzing the coefficients of the text regression model. This is facilitated by the `txtreg_analyze` command. The syntax is shown below.

3.3.1 Syntax

`txtreg_analyze using` *filename*

`using` *filename* specifies the location and the name of the pretrained text regression model.

3.3.2 Output

`txtreg_analyze` replaces the dataset that is currently in memory. The new dataset contains the n-grams and the estimated coefficients from the text regression.

4 Example: Predicting citations based on article titles

This section provides an example of the use of text regressions for the prediction of citations of scientific articles. This example is motivated by Iaria, Schwarz, and Waldinger (2022), who use predicted citations to control for observable differences between male- and female-authored scientific articles to measure gender gaps in citations. First, the authors train a text regression model based on the titles and the citations of articles written by men. Second, they create a measure of predicted citations for all articles, that is, how many citations we would expect an article to receive based on its title if it was written by a man. Finally, this predicted citation measure is used to control for differences in citations between male- and female-authored articles that occur because of the fact that women potentially work on different topics.

4.1 Model training

The example dataset consists of 100,000 observations. Each observation represents one scientific article with its title and the number of citations. First, the data are loaded, and the citation variable is transformed in two different ways (log and binary). Afterward, the `txtreg_train` command is called. The example shows the use of `txtreg_train` to train least-squares and logistic regressions with different regularization terms.

```
. ***********************************************************************
. **#* 1) Load data
. ***********************************************************************
. use wos_all_cite_prediction_example

. * generate log citations for regression
. generate WOS_TOTAL_ln = ln(1+WOS_TOTAL)

. * generate dummy for logit

. generate WOS_TOTAL_d = WOS_TOTAL>0

. * define list of stopwords
. global stopwords "a able about across after all almost also am among an and
> any are as at be because been but by can cannot could dear did do does either
> else ever every for from get got had has have he her hers him his how however
> i if in into is it its just least let like likely may me might most must my
> neither no nor not of off often on only or other our own rather said say says
> she should since so some than that the their them then there these they this
> tis to too twas us wants was we were what when where which while who whom
> why will with would yet you your"
. ***********************************************************************
. **#* 2) Train text regression using txtreg_train
. ***********************************************************************
. **#* 2a)

. txtreg_train WOS_TOTAL_ln title using "./models/predicted_citation_lasso.pkl",
> model("reg") regularization("lasso") regu_range(0.00005,0.0001) ngrams(2)
> seed(1502) tfidf stem stem_lang("english") stopwords("$stopwords")
> min_freq(3) max_freq(0.3) max_voc(5000)
Step 1/4 :Loading Data from Stata
Stemming text
Step 2/4 :Tokenizing Data
Tfidf is used
Step 3/4 :Training Model (This may take some time)
Score Fold 0 (Regularization=0.000050): 0.120290
Score Fold 1 (Regularization=0.000056): 0.132191
Score Fold 2 (Regularization=0.000061): 0.109037
Score Fold 3 (Regularization=0.000067): 0.117784
Score Fold 4 (Regularization=0.000072): 0.123905
Score Fold 5 (Regularization=0.000078): 0.119950
Score Fold 6 (Regularization=0.000083): 0.109066
Score Fold 7 (Regularization=0.000089): 0.113498
Score Fold 8 (Regularization=0.000094): 0.122939
Score Fold 9 (Regularization=0.000100): 0.117839
```

```
************************************************
************** Model Parameters ****************
************************************************
Parameters of tokenizer:
TfidfVectorizer(max_df=0.3, max_features=5000, min_df=3, ngram_range=(1, 2),
                stop_words={'a', 'able', 'about', 'across', 'after', 'all',
                            'almost', 'also', 'am', 'among', 'an', 'and', 'any',
                            'are', 'as', 'at', 'be', 'because', 'been', 'but',
                            'by', 'can', 'cannot', 'could', 'dear', 'did', 'do',
                            'does', 'either', 'else', ...},
                sublinear_tf=True)
************************************************
Dimensions of document-n-gram-matrix:
(100000, 5000)
************************************************
Parameters of trained model:
Lasso(alpha=5.555555555555556e-05, random_state=5057)
************************************************
Chosen regularization strength:
5.555555555555556e-05
Model Score: 0.122109
************************************************
Step 4/4 :Saving Model
```

```
. **#* 2b)

. txtreg_train WOS_TOTAL_ln title using "./models/predicted_citation_ridge.pkl",
> model("reg") regularization("ridge") regu_range(0.5,2) ngrams(2) seed(5184)
> tfidf stem stem_lang("english") stopwords("$stopwords") min_freq(3)
> max_freq(0.3) max_voc(5000)
```

(*output omitted*)

```
. **#* 2c)

. txtreg_train WOS_TOTAL_ln title using
> "./models/predicted_citation_elasticnet.pkl",
> model("reg") regularization("elasticnet") regu_range(0.00005,0.0001)
> ngrams(2) seed(7469) tfidf stem stem_lang("english") stopwords("$stopwords")
> min_freq(3) max_freq(0.3) max_voc(5000)
```

(*output omitted*)

```
. **#* 2d)

. txtreg_train WOS_TOTAL_d title using
> "./models/predicted_citation_logit_ridge.pkl",
> model("logit") regularization("ridge") regu_range(1,10)  ngrams(2)
> seed(2134) tfidf stem stem_lang("english") stopwords("$stopwords") min_freq(3)
> max_freq(0.3) max_voc(5000)
```

(*output omitted*)

During the training process, `txtreg_train` reports the individual steps that were performed. In the first step, the data are loaded from Stata, and the command reports whether the text strings are stemmed. The second step involves the creation of the document-n-gram-matrix. In each case, unigrams and bigrams are extracted from the text after the removal of stopwords and stemming.[5] Furthermore, the document-n-gram-matrix is reweighted using tf–idf. In this way, `txtreg_train` automatically handles the preparation of the titles for the machine learning.

5. To speed up the model training, I restricted the vocabulary to the 5,000 most frequent n-grams.

In step 3, `txtreg_train` reports the regularization strength that is used to train each fold and the resulting out-of-sample performance. If least-squares regressions are used, the performance score is the out-of-sample R^2 of the text regression. In the case of logistic regressions, the out-of-sample F1-score is reported. At the end of the model training, the command reports the parameters of the vectorizer, which generates the document-n-gram-matrix and the parameters of the final model, including the chosen regularization strength and the out-of-sample performance score. For logistic regressions, the command also reports the confusion matrix (a matrix showing the actual versus predicted outcomes). In the last step, the model is then saved in the specified location.

4.2 Making use of pretrained models

To show the potential of the `txtreg_train`, `txtreg_predict`, and `txtreg_analyze` commands for data sharing, I provide a text regression model that was pretrained on over 24 million articles in the fields of medicine, biology, physics, math, and chemistry for the years 1900–2010 based on the ISI Web of Science using a ridge regression. I use the following line of code:[6]

```
. txtreg_train WOS_TOTAL_ln title using "./models/predicted_citation_all.pkl",
> model("reg") regularization("ridge") regu_range("0.1,100") ngrams(2)
> seed(1502) tfidf stem stem_lang("english") stopwords("$stopwords")
> min_freq(100) max_freq(0.3) max_voc(200000)
```

The total training of this model took 2.5 hours. The model achieves an out-of-sample R^2 of 0.24. The `txtreg_predict` command can be used to load this model and generate a new variable containing the predicted citations. While the training of the model can take a long time and be computationally intensive, the prediction is usually far quicker. In the example, the prediction took less than a minute, most of which is needed to stem the text strings.

```
. ********************************************************************
. **#* 3) Predict outcome using txtreg_predict
. ********************************************************************
. txtreg_predict title using "./models/predicted_citation_all.pkl",
> name_new_var("predicted_citation") stem stem_lang("english")
Loading Data from Stata
Loading Model: ./models/predicted_citation_all.pkl
Stemming text
. * plot binscatter of predicted versus actual citations
. binscatter WOS_TOTAL_ln predicted_citation, n(100)
> xtitle("ln(1+predicted citations)", margin(top))
> ytitle("ln(1+citations)", margin(right)) graphregion(color(white))
> lcolor(dkorange) ylabel(,angle(0))
```

After the command runs, the `predicted_citation` variable is added to the dataset. It is easy to show that the predicted citations are highly correlated with the actual

6. This pretrained model is available on the author's website at https: // www.carloschwarz.eu / programming.

citations using a binscatter plot (see figure 1) created with the `binscatter` command (Stepner 2013). The strong positive relationship suggests that the trained model performs very well. In the example, the original data already contained information on the citations. The pretrained citation prediction model could alternatively be used to either generate predicted citations for datasets where citations are unknown or create proxies of scientific influence for other text data sources.

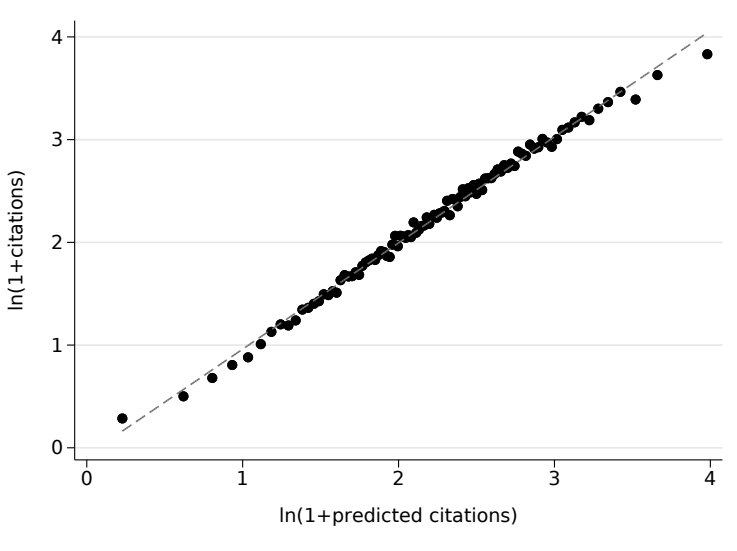

Figure 1. Predicted versus actual citations.
NOTES: This figure shows a binscatter of log(1+ number of predicted citations) and the log(1+ number of citations). The number of citations is based on the ISI Web of Science.

4.3 Analyzing model coefficients

Last, we can use the `txtreg_analyze` command to investigate if the model also identifies intuitive relationships between n-grams and citations, that is, which n-grams have the largest predictive power for the outcome variable. This is achieved by specifying `txtreg_analyze` and the model path. Note that `txtreg_analyze` will replace the dataset that is currently in memory. The new dataset has one row for each n-gram in the vocabulary V and a variable containing the regression coefficient that is associated with each n-gram. It is then straightforward to obtain the most predictive n-grams by simply sorting the coefficients by size.

```
. ********************************************************************
. **#* 4) Analyze coefficients using txtreg_analyze
. ********************************************************************
. txtreg_analyze using "./models/predicted_citation_all.pkl"
Loading Model: ./models/predicted_citation_all.pkl
  variable ngram was strL now str42
  (10,181,950 bytes saved)
. * list n-grams that are most predictive of high citations
. gsort -coef
. list ngram coef if _n<=10
```

	ngram	coef
1.	microrna	3.6021672
2.	graphen	3.5425722
3.	meta analysi	3.1363898
4.	random trial	3.1193311
5.	cut edge	3.0505555
6.	topolog insul	3.0193791
7.	organ framework	3.0116175
8.	arabidopsi	2.9907779
9.	energi harvest	2.9237333
10.	mice lack	2.8591241

The 10 n-grams with the largest coefficients provide insights into which articles receive many citations. For example, the word "meta-analysis" appears. This makes intuitive sense because meta-analyses are often highly cited. Interestingly, articles that use the n-gram "cutting edge" in their title receive more citations, as do articles using "randomized trials". Similarly, the n-grams "microRNA", "graphen", and "mice lack" all represent research topics of considerable interest. Because the model was trained for all subjects and years combined, the 10 most frequent words do not show the full scope of associations learned by the model. In Iaria, Schwarz, and Waldinger (2022), the predicted citation model is trained separately by cohort and subject, and the authors show that highly intuitive relationships emerge.

5 Conclusions

This article described new commands that train text regression models in Stata. Aside from enabling quick training of text regressions, the `txtreg_train`, `txtreg_predict`, and `txtreg_analyze` commands provide researchers with the opportunity to share their pretrained models. All models trained by `txtreg_train` can be loaded using the `txtreg_predict` and `txtreg_analyze` commands. The commands thereby further the use of text-based prediction across the community of Stata users. Note that text regressions obviously can suffer from the same endogeneity problems as any other regression model. The coefficients from the model therefore should not be interpreted as the "causal" effect of using a specific word. Further, `txtreg_train`, `txtreg_predict`, and `txtreg_analyze` are limited to penalized linear regression models. While these models have the great advantage that they are easy to train and that their coefficients

are easy to interpret, more complex deep-learning-based models usually achieve greater predictive accuracy.

6 Programs and supplemental materials

To install a snapshot of the corresponding software files as they existed at the time of publication of this article, type

```
. net sj 23-3
. net install dm0112     (to install program files, if available)
. net get dm0112         (to install ancillary files, if available)
```

7 References

Barker, M., and F. Pöge. 2012. strdist: Stata module to calculate the Levenshtein distance, or edit distance, between strings. Statistical Software Components S457547, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s457547.html.

Bird, S., E. Klein, and E. Loper. 2009. *Natural Language Processing With Python: Analyzing Text With the Natural language Toolkit*. Sebastopol, CA: O'Reilly.

Gentzkow, M., B. Kelly, and M. Taddy. 2019. Text as data. *Journal of Economic Literature* 57: 535–574. https://doi.org/10.1257/jel.20181020.

Hastie, T., R. Tibshirani, and J. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. New York: Springer. https://doi.org/10.1007/978-0-387-84858-7.

Iaria, A., C. Schwarz, and F. Waldinger. 2022. Gender gaps in academia: Global evidence over the twentieth century. http://doi.org/10.2139/ssrn.4150221.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12: 2825–2830. https://doi.org/10.48550/arXiv.1201.0490.

Schonlau, M., N. Guenther, and I. Sucholutsky. 2017. Text mining with n-gram variables. *Stata Journal* 17: 866–881. https://doi.org/10.1177/1536867X1801700406.

Schwarz, C. 2018. ldagibbs: A command for topic modeling in Stata using latent Dirichlet allocation. *Stata Journal* 18: 101–117. https://doi.org/10.1177/1536867X1801800107.

———. 2019. lsemantica: A command for text similarity based on latent semantic analysis. *Stata Journal* 19: 129–142. https://doi.org/10.1177/1536867X19830910.

Stepner, M. 2013. binscatter: Stata module to generate binned scatterplots. Statistical Software Components S457709, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s457709.html.

The Pandas Development Team. 2020. Python Pandas. https://doi.org/10.5281/zenodo.3509134.

Williams, U., and S. P. Williams. 2014. txttool: Utilities for text analysis in Stata. *Stata Journal* 14: 817–829. https://doi.org/10.1177/1536867X1401400407.

About the author

Carlo Schwarz is an assistant professor at Bocconi University. His research interests are in the fields of applied microeconomics and political economy, with a focus on text analysis and machine learning (https://carloschwarz.eu/).