



The World's Largest Open Access Agricultural & Applied Economics Digital Library

This document is discoverable and free to researchers across the globe due to the work of AgEcon Search.

Help ensure our sustainability.

Give to AgEcon Search

AgEcon Search

<http://ageconsearch.umn.edu>

aesearch@umn.edu

*Papers downloaded from **AgEcon Search** may be used for non-commercial purposes and personal study only. No other use, including posting to another Internet site, is permitted without permission from the copyright owner (not AgEcon Search), or as allowed under the provisions of Fair Use, U.S. Copyright Act, Title 17 U.S.C.*

No endorsement of AgEcon Search or its fundraising activities by the author(s) of the following work or their employer(s) is intended or implied.

lsemantica: A command for text similarity based on latent semantic analysis

Carlo Schwarz
University of Warwick
Coventry, UK
c.r.schwarz@warwick.ac.uk

Abstract. In this article, I present the `lsemantica` command, which implements latent semantic analysis in Stata. Latent semantic analysis is a machine learning algorithm for word and text similarity comparison and uses truncated singular value decomposition to derive the hidden semantic relationships between words and texts. `lsemantica` provides a simple command for latent semantic analysis as well as complementary commands for text similarity comparison.

Keywords: `st0552`, `lsemantica`, machine learning, latent semantic analysis, latent semantic indexing, truncated singular value decomposition, text analysis, text similarity

1 Introduction

The semantic similarity of two text documents is a highly useful measure. Knowing that two documents have a similar content or meaning can, for example, help to identify documents by the same author, measure the spread of information, or detect plagiarism. There are two main problems when attempting to identify documents with similar meaning. First, the same word can have different meanings in different context. Second, different words can have the same meaning in other contexts. Thus, using just the words counts in documents as a measure of similarity is unreliable. The alternative of reading and hand-coding documents already becomes prohibitive for a few hundred documents.

By providing a reliable measure of semantic similarity for words and texts, latent semantic analysis (LSA) provides a solution to this problem. LSA was developed by Deerwester et al. (1990) for the task of automated information retrieval in search queries. In search queries, it is important to understand the relationships and meanings of words because just using the query terms often leads to unsatisfying search results. LSA improves results by accounting for the relationships and potential multiple meanings of words. It is this property that makes LSA applicable for a variety of different tasks, including the following:

1. Similarity of words (Landauer, Foltz, and Laham 1998)
2. Similarity of texts (Gomaa and Fahmy 2013; Iaria, Schwarz, and Waldinger 2018)
3. Computer-guided summary writing (Wolfe et al. 1998; Franzke et al. 2005)

4. Automated essay grading (Kintsch 2002; Miller 2003)
5. Assessing discourse coherence (Foltz 2007)

For all of these applications, LSA derives the hidden semantic connection between words and documents by using truncated singular value decomposition, the same transformation used in principal component analysis. Principal component analysis uses truncated singular value decomposition to derive the components that explain the largest amount of variation in the data. Similarly, truncated singular value decomposition enables LSA to “learn” the relationships between words by decomposing the “semantic space”. This process allows LSA to accurately judge the meaning of texts. While LSA uses word cooccurrences, LSA can infer much deeper relations between words.

Landauer (2007) compares the workings of LSA with a problem of simultaneous equations. For two equations $A+2B = 8$ and $A+B = 5$, neither equation alone is enough to infer the values of A or B . When one combines the two equations together, it becomes straightforward to calculate the respective values for A and B . Similarly, the meaning of a document is based on the sum of the meaning of its words: $\text{meaning_document} = \sum(\text{meaning_word}_1, \dots, \text{meaning_word}_n)$. For a computer, it is not possible to infer the meaning of the words based on one document alone, but LSA can “learn” the meaning of words using the large set of simultaneous equations provided by all documents in the corpus.

To illustrate the capabilities of LSA, Landauer (2007) provides the following example. On the one hand, the phrases “A circle’s diameter” and “radius of spheres” are judged by LSA to have similar meaning, despite having no word in common; on the other hand, the phrase “music of the spheres” is judged as dissimilar by LSA, despite using similar words. Thus, text similarity comparison using LSA is preferable to just using the raw word counts in each document because word frequencies completely ignore multiple meanings of words. Furthermore, LSA also outperforms more recent machine learning algorithms when it comes to document similarity comparison (Stevens et al. 2012).

In this article, I introduce the `lsemantica` command, which provides a solution for using LSA in Stata. `lsemantica` further facilitates the text-similarity comparison in Stata with the `lsemantica_cosine` command. In this way, `lsemantica` further improves the text analysis capabilities of Stata. Stata already allows one to calculate the Levenshtein edit distance with the `strdist` command (Barker and Pöge 2012), and the `txttool` command (Williams and Williams 2014) facilitates the cleaning and tokenizing of text data. Moreover, Schwarz’s (2018) `ldagibbs` command allows one to run latent Dirichlet allocation in Stata. While `ldagibbs` can group documents together by similar topics, `lsemantica` is preferable in cases where one is predominately interested in how similar documents are.

2 Decomposing the semantic space using LSA

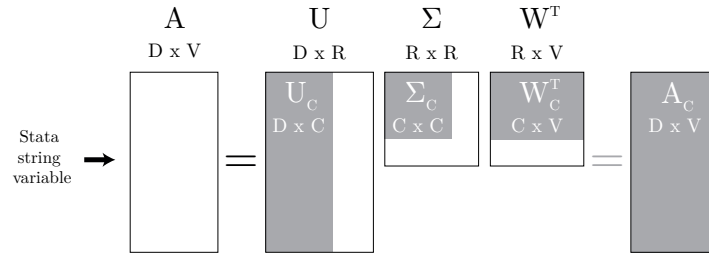
This section describes how truncated singular value decompositions allows LSA to retrieve connections between words. First, `lsemantica` creates a so-called bag-of-words representation of the texts in the string variable. In this process, `lsemantica` creates a document-term matrix \mathbf{A} . This matrix contains a row for each document $d \in D$ and a column for each unique term, that is, words, in the vocabulary V . Each cell in \mathbf{A} contains $f_{d,v}$, the number of times term v appears in document d :

$$\underbrace{\mathbf{A}}_{D \times V} = \begin{pmatrix} f_{1,1} & \cdots & f_{1,v} \\ \vdots & \ddots & \vdots \\ f_{d,1} & \cdots & f_{d,v} \end{pmatrix}$$

Second, `lsemantica` reweighs the word frequencies $f_{d,v}$ in the matrix \mathbf{A} by their term-frequency-inverse-document-frequency (TFIDF). Here $f_{d,v}$ is replaced by $\text{TFIDF}(f_{d,v}) = \{1 + \log(f_{d,v})\} \times [\log\{(1+D)/(1+d_v)\} + 1]$, where d_v is the number of documents in which term v appears. The TFIDF reweighting reduces the weights of words that appear in many documents because these words are usually less important for the overall meaning of documents. After the TFIDF reweighting, the matrix \mathbf{A} contains

$$\underbrace{\mathbf{A}}_{D \times V} = \begin{pmatrix} \text{TFIDF}(f_{1,1}) & \cdots & \text{TFIDF}(f_{1,v}) \\ \vdots & \ddots & \vdots \\ \text{TFIDF}(f_{d,1}) & \cdots & \text{TFIDF}(f_{d,v}) \end{pmatrix}$$

Finally, `lsemantica` applies singular value decomposition to the reweighted matrix \mathbf{A} . This process is represented in figure 1. Singular value decomposition transforms \mathbf{A} , of rank R , into three matrices such that $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{W}^T$, where \mathbf{U} is a $D \times R$ orthogonal matrix, \mathbf{W}^T is a $R \times V$ orthogonal matrix, and $\mathbf{\Sigma}$ is a $R \times R$ diagonal matrix (black boxes in figure 1). Afterward, `lsemantica` truncates the resulting matrices by removing the rows and columns associated with the smallest eigenvalues in the matrix $\mathbf{\Sigma}$ until only the user-chosen number of eigenvalues C remains. This truncation thereby reduces the dimensions of the matrices to C components such that \mathbf{U} becomes \mathbf{U}_C of dimension $D \times C$, $\mathbf{\Sigma}$ becomes $\mathbf{\Sigma}_C$ of dimension $C \times C$, and \mathbf{W}^T becomes \mathbf{W}_C^T of dimension $C \times V$ (shaded areas in figure 1).

Figure 1. LSA graphical representation¹

The number of components C that should be preserved during truncation is usually chosen based on the size of the vocabulary. Martin and Berry (2007) suggest that $100 \leq C \leq 1000$ usually leads to good results. During truncation, the entries of the original matrix \mathbf{A} change as the number of components is reduced. In the end, this process results in the best rank- C approximation of the original matrix \mathbf{A} , called \mathbf{A}_C . The truncation process is of utmost importance because it reduces the components of the semantic space to the C most important ones.

The output of `lsemantica` is then based on $\mathbf{U}_C \times \mathbf{\Sigma}_C$, a $D \times C$ document-component matrix that can be used to compare the similarity of documents. The individual components of the document-component matrix represent the reduced dimensions of the semantic space. These components capture the semantic relationships between the individual documents. Moreover, `lsemantica` can save the matrix \mathbf{W}_C^T , which allows one to compare the similarity of words. As illustrated by the example in the introduction, even documents that contain completely different words can be judged to be similar by LSA if the words appear together in similar semantic contexts.

3 Stata implementation

This section describes how the `lsemantica` command allows running LSA in Stata. With `lsemantica`, each document the user wants to use for LSA should be one observation in the dataset. If documents consist of thousands of words, users can split the documents into smaller parts, for example, paragraphs, to process each part separately. In any case, only one variable should be used for the text strings.² Furthermore, nonalphanumeric characters should be removed from the strings.

1. This figure is a modified version of a figure in Martin and Berry (2007) and visualizes the workings of LSA. First, the string variable is transformed into a document-term matrix \mathbf{A} . Next, this matrix is decomposed such that $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{W}^T$. The shading represents truncation, which derives the best rank- C approximation of \mathbf{A} .

2. Since Stata 13, `strL` variables allow one to store text strings up to 2,000,000,000 characters. Hence, even very long documents can be stored in a single variable.

The `lsemantica` command is started by specifying the variable that contains the text. There is an option to choose the number of components for truncated singular value decomposition. The options of `lsemantica` also allow one to remove stop words and short words from the text string. The `txttool` command (Williams and Williams 2014) provides more advanced text-cleaning options, if required. `lsemantica` also provides two options to reduce the size of the vocabulary. These options are helpful in cases where truncated singular value decomposition requires a long time because of the size of the vocabulary.

3.1 Syntax

```
lsemantica varname [, components(integer) tfidf min_char(integer)
  stopwords(string) min_freq(integer) max_freq(real) name_new_var(string)
  mat_save path(string) ]
```

3.2 Options

`components(integer)` specifies the number of components the semantic space should be reduced to by `lsemantica`. The number of components is usually chosen based on the size of the vocabulary. The default is `components(300)`.

`tfidf` specifies whether TFIDF reweighting should be used before applying the truncated singular value decomposition. In most cases, TFIDF reweighting will improve the results.

`min_char(integer)` allows the removal of short words from the texts. Words with fewer characters than `min_char(integer)` will be excluded from LSA. The default is `min_char(0)`.

`stopwords(string)` specifies a list of words to exclude from `lsemantica`. Usually, highly frequent words such as “I”, “you”, etc., are removed from the text because these words contribute little to the meaning of documents. Predefined stop word lists for different languages are available online.

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

`max_freq(real)` allows the removal of words that appear frequently in documents. Words that appear in a share of more than `max_freq(real)` documents will be excluded from LSA. The default is `max_freq(1)`.

`name_new_var(string)` specifies the name of the output variable created by `lsemantica`.

These variables contain the components for each document. The user should ensure that `name_new_var(string)` is unique in the dataset. By default, the name of the variable is `component_`, so the names of the new variables will be `component_1–component_C`, where C is the number of components.

`mat_save` specifies whether the word-component matrix should be saved. This matrix describes semantic relationships between words. By default, the matrix will not be saved.

`path(string)` sets the path where the word-component matrix is saved.

3.3 Output

`lsemantica` generates C new variables, which are the components generated by the truncated singular value decomposition. As described in the previous sections, these components capture the semantic relationships of documents and allow one to calculate the similarity between documents.

The similarity of documents based on LSA is usually measured by the cosine similarity of the component vectors of each document. The cosine similarity of two documents $d1$ and $d2$ and their respective document-component vectors δ_{d1} and δ_{d2} is defined as

$$\text{cosine_sim}_{(d1,d2)} = \frac{\sum_{c=1}^C (\delta_{d1,c} \times \delta_{d2,c})}{\sqrt{\sum_{c=1}^C \delta_{d1,c}^2} \times \sqrt{\sum_{c=1}^C \delta_{d2,c}^2}}$$

The cosine similarity is hence the uncentered version of the correlation coefficient. The cosine similarity is 1 for highly similar documents and -1 for completely dissimilar documents. When one uses LSA, the cosine similarity usually lies within the unit interval. Only for highly dissimilar documents will the cosine similarity be negative.

`lsemantica` further provides the `lsemantica_cosine` command to facilitate the analysis of the cosine similarity. `lsemantica_cosine` calculates the cosine similarity for all documents and stores it in Mata.³ Furthermore, `lsemantica_cosine` can provide summary statistics for the cosine similarity and find highly similar documents. A separate help file explains the syntax of `lsemantica_cosine`.

4 Example

The example dataset contains the title of 41,349 articles published in economic journals from 1980 to 2016. After one loads the data, nonalphanumeric characters are removed from the title strings in preparation for LSA.

3. The cosine similarity matrix is stored in Mata because it is likely that the dimensions of the matrix exceed the limits of Stata.

```

. use example_data.dta, clear
. generate text_strings = title
. *remove nonalphanumeric characters
. replace text_strings=strlower(text_strings)
(41,347 real changes made)
. replace text_strings = subinstr( text_strings, ".", " ", .)
(936 real changes made)
. replace text_strings = subinstr( text_strings, "!", " ", .)
(32 real changes made)
. replace text_strings = subinstr( text_strings, "?", " ", .)
(4,407 real changes made)
. replace text_strings = subinstr( text_strings, ":", " ", .)
(12,193 real changes made)
. replace text_strings = subinstr( text_strings, ";", " ", .)
(18 real changes made)
. replace text_strings = subinstr( text_strings, ",", " ", .)
(5,434 real changes made)
. replace text_strings = subinstr( text_strings, "(", " ", .)
(291 real changes made)
. replace text_strings = subinstr( text_strings, ")", " ", .)
(291 real changes made)
. replace text_strings = subinstr( text_strings, "&", " ", .)
(211 real changes made)
. replace text_strings = subinstr( text_strings, `""`, " ", .)
(274 real changes made)
. replace text_strings = subinstr( text_strings, " ", " ", .)
(18,125 real changes made)
. replace text_strings = stritrim(text_strings)
(281 real changes made)
. list text_strings if _n<=10

```

| | text_strings |
|----|---|
| 1. | what is labor supply and do taxes affect it |
| 2. | tax rules and the mismanagment of monetary policy |
| 3. | a consistent characterization of a near-century of price behavior |
| 4. | comparison of interwar and postwar business cycles monetarism recon.. |
| 5. | trade policy as an input to development |

(output omitted)

LSA is then started by simply calling the `lsemantica` command. First, `lsemantica` prepares the documents and produces the document-term matrix. `lsemantica` also removes words shorter than 4 characters and words that appear in fewer than 10 documents or in more than half of all documents from the data. Furthermore, stop words are removed from the data. The resulting document-term matrix is then reweighed using TFIDF. The command reports every time when 10% of the vocabulary has been processed.


```
. 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"

. lsemantica text_strings, components(300) min_char(4) min_freq(10)
> max_freq(0.5) tfidf stopwords("$stopwords") path("$path") mat_save
*****
**** Latent Semantic Analysis ****
*****
Number of Components: 300
Minimal Word Length: 4
Minimal Word Frequency: 10
Maximal Word Frequency: .5
*****

*****
***** Preparing Documents *****
*****

*** Creating Document-Word-Matrix ***
Processing Vocabulary:
10% done
Processing Vocabulary:
20% done
(output omitted)
Processing Vocabulary:
90% done
Processing Vocabulary:
100% done
```

If documents do not have any words left after the text cleaning, `lsemantica` will remove these observations from the data because they interfere with the truncated singular value decomposition. `lsemantica` reports which documents have been removed from the data as well as the size of the vocabulary. In the example, 167 documents are removed. The removal of documents is mainly due to the option `min_freq(10)`. The dataset, for example, contains an article simply titled “Notches”. Because the word “Notches” appears only in the title of three articles, it is not included in the vocabulary.

Next, `lsemantica` calculates the truncated singular value decomposition, which is computationally intensive and can take some time. In some cases, Stata becomes unresponsive during this process. The time required for the truncated singular value decomposition increases with the size of the document-term matrix and hence with the number of documents and the size of the vocabulary.⁴

4. For a very large corpus, the time required might become impractical. In this case, the user can attempt to speed up the singular value decomposition by reducing the size of the vocabulary using the `min_freq()` and `max_freq()` options.

The following observation were removed from the data, since they did not
> have any remaining words:

```

1
1 240
2 683
(output omitted)
167 41324

```

Size of Vocabulary:
3029

Now moving to SVD: this may take a while!

```

*****
***** Running Truncated SVD *****
*****

```

After `lsemantica` is finished running, one can begin to analyze the similarity of documents by calculating the cosine similarity between the component vectors using the `lsemantica_cosine` command.⁵ The resulting cosine similarity matrix is stored in Mata because of its dimensions. If required, the full cosine similarity matrix can be retrieved from Mata using `getmata (cosin.*) = cosine_sim` after the `lsemantica_cosine` command. `lsemantica_cosine` also allows one to calculate the average similarity as well as the maximal and minimal similarity to other article titles.

```

. lsemantica_cosine component_1-component_300, mean_cosine min_cosine
> max_cosine find_similar(10) find_similar_cosine(10)
. summarize mean_similarity, detail

```

| mean_similarity | | | | |
|-----------------|----------|----------|-------------|----------|
| Percentiles | | Smallest | | |
| 1% | .0044491 | .0018937 | | |
| 5% | .0064587 | .0021306 | | |
| 10% | .0078202 | .0021863 | Obs | 41,182 |
| 25% | .0106267 | .0023644 | Sum of Wgt. | 41,182 |
| 50% | .0144946 | | Mean | .0151279 |
| | | Largest | Std. Dev. | .0060271 |
| 75% | .0189352 | .0414024 | | |
| 90% | .0232734 | .0423894 | Variance | .0000363 |
| 95% | .0260268 | .0429952 | Skewness | .5782946 |
| 99% | .0315863 | .0432296 | Kurtosis | 3.186935 |

5. `lsemantica_cosine` is memory intensive because the command multiplies the document-component matrix with itself.

```
. summarize max_similarity, detail
```

| max_similarity | | | | | |
|----------------|----------|----------|-------------|-----------|--|
| Percentiles | | Smallest | | | |
| 1% | .5652832 | .3455565 | | | |
| 5% | .6286974 | .3464915 | | | |
| 10% | .6642829 | .3655767 | Obs | 41,182 | |
| 25% | .7306858 | .3674786 | Sum of Wgt. | 41,182 | |
| 50% | .816731 | | Mean | .8242296 | |
| | | Largest | Std. Dev. | .1226226 | |
| 75% | .946547 | 1 | | | |
| 90% | .9941671 | 1 | Variance | .0150363 | |
| 95% | .9985264 | 1 | Skewness | -.1298808 | |
| 99% | 1 | 1 | Kurtosis | 2.128631 | |

```
. summarize min_similarity, detail
```

| min_similarity | | | | | |
|----------------|-----------|-----------|-------------|-----------|--|
| Percentiles | | Smallest | | | |
| 1% | -.2512143 | -.4081201 | | | |
| 5% | -.1909725 | -.4081201 | | | |
| 10% | -.1618612 | -.4010201 | Obs | 41,182 | |
| 25% | -.1235881 | -.4010201 | Sum of Wgt. | 41,182 | |
| 50% | -.0936972 | | Mean | -.1047622 | |
| | | Largest | Std. Dev. | .0433816 | |
| 75% | -.0743099 | -.0296275 | | | |
| 90% | -.0625703 | -.0294599 | Variance | .001882 | |
| 95% | -.0569326 | -.0284645 | Skewness | -1.634287 | |
| 99% | -.0480057 | -.0281307 | Kurtosis | 7.055067 | |

Furthermore, `lsemantica_cosine` can find the most similar articles for each of the articles in the data. In the example, the 10 most similar articles are calculated. Afterward, the five most similar article titles for the first article in the data are listed. One can see that LSA accurately identified highly similar articles all discussing questions of labor supply.

```
. list most_similar_* cosine_most_similar_* if _n ==1
```

| | | | | | | |
|----|------------------------|------------------------|------------------------|-------------------------|------------------------|------------------------|
| 1. | most_s-1 28468 | most_s-2 513 | most_s-3 6288 | most_s-4 27267 | most_s-5 34267 | most_s-6 26707 |
| | most_s-7 38850 | most_s-8 29305 | most_s-9 29322 | most_s-10 38932 | cosine_~1 .81473917 | cosine_~2 .81221156 |
| | cosine_~3 .67660235 | cosine_~4 .67439987 | cosine_~5 .67347484 | cosine_~6 .66513775 | cosine_~7 .66482448 | cosine_~8 .66331657 |
| | cosine_~9 .66331657 | | | cosine_~10 .66203841 | | |

```
. list title if _n==1 | _n==513 | _n==28468 | _n==27267 | _n==6288 | _n==34267
```

| | title |
|--------|--|
| 1. | What Is Labor Supply and Do Taxes Affect It? |
| 513. | Family Labor Supply with Taxes |
| 6288. | The Effect of Taxes on Labor Supply in the Underground Economy |
| 27267. | Low-Skilled Immigration and the Labor Supply of Highly Skilled Women |
| 28468. | Labor Supply and Taxes: A Survey |
| 34267. | Worktime Regulations and Spousal Labor Supply |

`lsemantica` can calculate the number of articles to which the original article is highly similar. In the example, a cutoff for the cosine similarity of 0.75 was chosen. The Mata code generates a new variable called `high_sim_articles` containing the number of articles published afterward that have a cosine similarity above this cutoff. The example data also contain the number of citations for each article. Hence, one can estimate a regression of the number of similar articles on the number of citations. The regression reveals a significant positive relationship between the two variables.

```
. sort pub_year
. mata
----- mata (type end to exit) -----
: pub_year = st_data(., "pub_year")
: high_sim_paper= J(0,1,.)
: for (y=1980 ; y<=2016 ; y++){
>
>     cosine_submat = select(cosine_sim, pub_year==y)
>     cosine_submat = select(cosine_submat',pub_year:>=y )'
>
>     high_sim = rowsum(( cosine_submat:>=J(rows(cosine_submat),
> cols(cosine_submat), 0.75) ))
>
>     high_sim_paper = high_sim_paper \ high_sim
> }
: var = st_addvar("double", "high_sim_paper")
: st_store(., "high_sim_paper", high_sim_paper)
: end
-----
```

```
. regress citations high_sim_paper i.pub_year
```

| Source | SS | df | MS | Number of obs | = | 41,182 |
|----------|------------|--------|------------|---------------|---|--------|
| Model | 34061173.2 | 37 | 920572.248 | F(37, 41144) | = | 79.57 |
| Residual | 476018071 | 41,144 | 11569.5623 | Prob > F | = | 0.0000 |
| | | | | R-squared | = | 0.0668 |
| | | | | Adj R-squared | = | 0.0659 |
| Total | 510079244 | 41,181 | 12386.2763 | Root MSE | = | 107.56 |

| citations | Coef. | Std. Err. | t | P> t | [95% Conf. Interval] | |
|------------------|-----------|-----------|-------|-------|----------------------|-----------|
| high_sim_pa-r | .0982082 | .0456843 | 2.15 | 0.032 | .008666 | .1877504 |
| pub_year | | | | | | |
| 1981 | 19.001 | 15.02938 | 1.26 | 0.206 | -10.45691 | 48.4589 |
| (output omitted) | | | | | | |
| 2016 | -43.48352 | 10.7657 | -4.04 | 0.000 | -64.58452 | -22.38252 |
| _cons | 47.07542 | 10.51478 | 4.48 | 0.000 | 26.46623 | 67.68461 |

Finally, `lsemantica` can compare semantic relationships and the similarity of words. Using `lsemantica_word_comp`, one can import the word-component matrix stored by `lsemantica`. Again, `lsemantica_cosine` can be used to calculate the cosine between the words in the data and find the most similar words. The example shows that `lsemantica` identifies that the words “labor”, “force”, “segmented”, “division”, “frictional”, and “monopsony” are related to one another.

```
. lsemantica_word_comp using "word_comp.mata"
. lsemantica_cosine component_1-component_300, find_similar(10)
> find_similar_cosine(10)
. * "labour": _n ==1516
. list most_similar* cosine_most_similar* if _n ==1516
```

| | | | | | | |
|-------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|
| 1516. | most_s-1 1768 | most_s-2 812 | most_s-3 1144 | most_s-4 2465 | most_s-5 1111 | most_s-6 291 |
| | most_s-7 1394 | most_s-8 1309 | most_s-9 2253 | most_s-10 2232 | cosine_~1 .53823032 | cosine_~2 .46088074 |
| | cosine_~3 .33327173 | cosine_~4 .30747563 | cosine_~5 .26425962 | cosine_~6 .2354069 | cosine_~7 .23372151 | cosine_~8 .21986212 |
| | cosine-9 .2002355 | | | cosine-10 .19779614 | | |

```
. list word if _n==1516 | _n==812 | _n==1144 | _n==2465 | _n==1111 | _n==1768
```

| | words |
|-------|------------|
| 812. | division |
| 1111. | force |
| 1144. | frictional |
| 1516. | labor |
| 1768. | monopsony |
| 2465. | segmented |

5 Conclusion

LSA has been proven to be a versatile text analysis algorithm. `lsemantica` brings LSA to Stata and can help researchers to more accurately measure the similarity of text documents with the `lsemantica_cosine` command. Thereby, `lsemantica` opens up new ways to analyze text data in Stata.

6 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>.
- Deerwester, S., S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science and Technology* 41: 391–407.
- Foltz, P. W. 2007. Discourse coherence and LSA. In *Handbook of Latent Semantic Analysis*, ed. T. K. Landauer, D. S. McNamara, S. Dennis, and W. Kintsch, 167–184. New York: Routledge.
- Franzke, M., E. Kintsch, D. Caccamise, N. Johnson, and S. Dooley. 2005. Summary Street[®]: Computer support for comprehension and writing. *Journal of Educational Computing Research* 33: 53–80.
- Gomaa, W. H., and A. A. Fahmy. 2013. A survey of text similarity approaches. *International Journal of Computer Applications* 68: 13–18.
- Iaria, A., C. Schwarz, and F. Waldinger. 2018. Frontier knowledge and scientific production: Evidence from the collapse of international science. *Quarterly Journal of Economics* 133: 927–991.
- Kintsch, W. 2002. The potential of latent semantic analysis for machine grading of clinical case summaries. *Journal of Biomedical Informatics* 35: 3–7.

- Landauer, T. K. 2007. LSA as a theory of meaning. In *Handbook of Latent Semantic Analysis*, ed. T. K. Landauer, D. S. McNamara, S. Dennis, and W. Kintsch, 3–34. New York: Routledge.
- Landauer, T. K., P. W. Foltz, and D. Laham. 1998. An Introduction to Latent Semantic Analysis. *Discourse Processes* 25: 259–284.
- Martin, D. I., and M. W. Berry. 2007. Mathematical foundations behind latent semantic analysis. In *Handbook of Latent Semantic Analysis*, ed. T. K. Landauer, D. S. McNamara, S. Dennis, and W. Kintsch, 35–56. New York: Routledge.
- Miller, T. 2003. Essay assessment with latent semantic analysis. *Journal of Educational Computing Research* 29: 495–512.
- Schwarz, C. 2018. ldagibbs: A command for topic modeling in Stata using latent Dirichlet allocation. *Stata Journal* 18: 101–117.
- Stevens, K., P. Kegelmeyer, D. Andrzejewski, and D. Buttler. 2012. Exploring topic coherence over many models and many topics. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Jeju Island, Korea: Association for Computational Linguistics.
- Williams, U., and S. P. Williams. 2014. txttool: Utilities for text analysis in Stata. *Stata Journal* 14: 817–829.
- Wolfe, M. B. W., M. E. Schreiner, B. Rehder, D. Laham, P. W. Foltz, W. Kintsch, and T. K. Landauer. 1998. Learning from text: Matching readers and texts by latent semantic analysis. *Discourse Processes* 25: 309–336.

About the author

Carlo Schwarz is a PhD student at the University of Warwick. His research interests are in the field of applied microeconomics and political economy, with a focus on text analysis and machine learning (<http://www.carloschwarz.eu>).