

Isemantica: A Stata Command for Text Similarity based on Latent Semantic Analysis

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

Abstract. The Isemantica command, presented in this paper, implements Latent Semantic Analysis in Stata. Latent Semantic Analysis is a machine learning algorithm for word and text similarity comparison. Latent Semantic Analysis uses Truncated Singular Value Decomposition to derive the hidden semantic relationships between words and texts. Isemantica provides a simple command for Latent Semantic Analysis in Stata as well as complementary commands for text similarity comparison.

Keywords: st0001, Isemantica, 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 plagiarisms. There are two main problems when attempting to identify documents with similar meaning. First, the same word can have different meanings in different context. Secondly, different words can have the same meaning in other contexts. As a result, 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.

Latent Semantic Analysis (LSA) provides a solution to this problem, by providing a reliable measure of semantic similarity for words and texts. 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 accurately judge the relationships and meanings of words, since just using the query terms often leads to unsatisfying search results. LSA improves search results, by taking into account the relationships and potential multiple meanings of words. It is this property that makes LSA applicable for a variety of different tasks among others:

1. Similarity of Words (Landauer et al. 1998)

LSA in Stata


5. Assessing Discourse Coherence (Foltz 2007)

For all this application 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. In a similar vein, Truncated Singular Value Decomposition makes it possible for 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 makes use of word co-occurrences, LSA can infer much deeper relations between words.

Landauer (2007) compares the workings of LSA to 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 $B$. By combining 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: $\text{meaning document} = \sum (\text{meaning word}_1; \ldots; \text{meaning word}_n)$. For a computer, also in this context, it is not possible to infer the meaning of the words based on one document alone, but LSA is able to “learn” the meaning of words using the large set of simultaneous equations provided by all documents in the corpus.

As an illustration of the capabilities of LSA, Landauer (2007) provides the following example. On the one hand the text passages “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 text passage “music of the spheres” is judged as dissimilar by LSA, despite using similar words. As a result, text similarity comparison using LSA is preferable to just using the raw word counts in each document, since 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).

This paper introduces the lsemantica command. lsemantica 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 to calculate the Levenshtein edit distance with the strdist command (Barker 2012) and the txttool command (Williams and Williams 2014) facilitates the cleaning and tokenizing of text data. Moreover, Schwarz (2017) and the ldagibbs command makes it possible to run Latent Dirichlet Allocation in Stata. While ldagibbs allows to 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 Latent Semantic Analysis

This section describes how Truncated Singular Value Decompositions allows LSA to retrieve connections between words. As the first step, lsemantica creates a so called bag-of-words representation of the text data. In this process lsemantica creates a document-term-matrix \( A \). The matrix \( A \) contains a row for each document \( d \in D \) and a column for each unique term, i.e. words, in the vocabulary \( V \). Each cell in \( A \) contains \( f_{d,v} \) the number of times term \( v \) appears in document \( d \):

\[
A_{D \times V} = \begin{pmatrix}
  f_{1,1} & \cdots & f_{1,d} \\
  \vdots & \ddots & \vdots \\
  f_{d,1} & \cdots & f_{d,v}
\end{pmatrix}
\]

The second step of lsemantica is to reweigh the word frequencies \( f_{d,v} \) in the matrix \( A \) by their term-frequency-inverse-document-frequency (tf-idf). In this step \( f_{d,v} \) is replaced by \( tf-idf(f_{d,v}) = (1 + \log(f_{d,v})) \cdot \left( \log \left( \frac{1 + d_v}{1 + D} \right) + 1 \right) \), where \( d_v \) is the number of documents term \( v \) appears in at least once. The tf-idf reweighting reduces the weights of words that appear in many documents, since these words are usually less important for the overall meaning of documents. After the tf-idf reweighting the matrix \( A \) contains:

\[
A_{D \times V} = \begin{pmatrix}
  tf-idf(f_{1,1}) & \cdots & tf-idf(f_{1,d}) \\
  \vdots & \ddots & \vdots \\
  tf-idf(f_{d,1}) & \cdots & tf-idf(f_{d,v})
\end{pmatrix}
\]

As the final step, lsemantica applies Singular Value Decomposition to the reweighted matrix \( A \). Singular Value Decomposition transforms \( A \), of rank \( R \), into three matrices such that \( A = U \Sigma W^T \), where \( U \) is a \( D \times R \) orthogonal matrix, \( W^T \) is a \( R \times V \) orthogonal matrix, and \( \Sigma \) is a \( R \times R \) diagonal matrix. Afterwards, lsemantica truncates the resulting matrices by removing the rows and columns associated with the smallest eigenvalues in the matrix \( \Sigma \). This truncation process reduces the dimensions of the matrices to a user-chosen number of components \( C \), such that \( U \) becomes \( U_C \) of dimension \( D \times C \), \( \Sigma \) becomes \( \Sigma_C \) of dimension \( C \times C \), and \( W^T \) becomes \( W^T_C \) of dimension \( C \times V \). The truncation process is represented in Figure 1.

The number of components \( C \) is usually chosen based on the size of the vocabulary. Martin and Berry (2007) suggest \( 100 \leq C \leq 1000 \). During this truncation process, the entries of the original matrix \( A \) change as the number of components is reduced. In the end, this process results in the best rank-\( C \) approximation of the original matrix \( A \) called \( A_C \). The truncation process of lsemantica is of utmost importance, since it reduces the components of the semantic space to the \( C \) most important ones.

The output of lsemantica is then based on \( U_C \cdot \Sigma_C \), 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
Figure 1: LSA Graphical Representation

$$A_{D \times V} = U_{D \times C} \Sigma_{C \times C} W_{C \times V}^T$$

$$U_c \cdot \Sigma_c : \text{document-component-matrix}$$

(used to compare similarity of documents)

semantic space. These components capture the semantic relationships between the individual documents. Moreover, lsemantic can save the matrix $W_{C}^T$ that allows 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 lsemantic command allows running LSA in Stata. To use lsemantic, each observation in the data set should represent one of the documents the user wants to use for LSA. If documents consist of thousands of words, users can split the documents into smaller parts to classify each part separately. In any cases, the text strings for the classification should be contained in one variable. Furthermore, non-alphanumerical characters should be removed from the text strings, since they can potentially interfere with the classification.

The lsemantic command only requires the variables containing the text strings as an input. The individual options of the command allow for the modification of LSA. For convenience lsemantic also includes some text cleaning capabilities to remove stopwords and short words from the data. For more advanced text cleaning options, see the txttool command (Williams and Williams 2014). lsemantic also provides two option to reduce the size of the vocabulary. These options are helpful in cases where Truncated Singular Value Decomposition requires a large amount of time due to the size of the vocabulary.

3.1 Syntax

lsemantic varname[, components(integer) tfidf min_char(integer) stopwords(string) min_freq(integer) max_freq(real) name_new_var(string)]
3.2 Options

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

\texttt{tfidf} specifies if term-frequency-inverse-document-frequency reweighting should be used before applying the Truncated Singular Value Decomposition. In most cases tf-idf reweighting will improve the results.

\textbf{Text Cleaning Options}

\texttt{min_char(integer)} allows the removal of short words from the texts. Words with less characters than \texttt{min_char(integer)} will be excluded from the sampling algorithm. The default is \texttt{min_char(0)}.

\texttt{stopwords(string)} specifies a list of words to exclude from \texttt{lsemantica}. Usually highly frequent words such as "I", "you", etc. are removed from the text, since these words contribute little to the meaning of documents. Predefined stopword lists for different languages are available online.

\texttt{min_freq(integer)} allows the removal of words that appear in few documents. Words that appear in fewer documents than \texttt{min_freq(integer)} will be excluded from the sampling algorithm. The default is \texttt{min_freq(0)}.

\texttt{max_freq(real)} allows the removal of words that appear very frequently in documents. Words that appear in a share of more than \texttt{max_freq(real)} documents will be excluded from the sampling algorithm. The default is \texttt{max_freq(1)}.

\textbf{Output Options}

\texttt{name_new_var(string)} specifies the name of the output variable created by \texttt{lsemantica}. These variables contain the topic assignments for each document. The user should ensure that \texttt{name_new_var(string)} is unique in the data set. If nothing is specified, the name of the variable is \texttt{name_new_var("component_.")}, such that the names of the new variables will be \texttt{component_1-component_C}, where the C is the number of the components.

\texttt{mat_save} specifies if the word-component matrix should be saved. This matrix describes semantic relationships between words. By default, the matrix will not be saved.

\texttt{path(string)} sets the path where the word-component matrix is saved.
3.3 Output

lsemantica generates $C$ new variables. These variables describe the components generated by the Truncated Singular Value Decomposition of each document. As described in the previous sections, these components capture the semantic relationships of documents and allow 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 $d_1$ and $d_2$ and their respective document-component vectors $\delta_{d_1}$ and $\delta_{d_2}$ is defined as:

$$cosine_{sim}(d_1, d_2) = \frac{\sum_{c=1}^{C} (\delta_{d_1,c} \cdot \delta_{d_2,c})}{\sqrt{\sum_{c=1}^{C} \delta_{d_1,c} \cdot \sqrt{\sum_{c=1}^{C} \delta_{d_2,c}}}}$$

The cosine similarity is hence the un-centered version of the correlation coefficient. The cosine similarity is 1 for perfect similarity documents and -1 for completely dissimilar documents. When using LSA, the cosine similarity usually lies within the unit interval. Only for highly dissimilar documents the cosine similarity will 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 in the corpus and stores it in Mata. 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 data set contains the title of 41,349 papers published in economic journals in the years 1980 until 2016. After loading the data, non-alphanumerical characters are removed from the title strings in preparation of LSA.

```
. use "$path/Data/example_data.dta", clear
. * combine title and summary
. gen text_strings = title
. *remove non alpha numerical 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, ":", " ", .)
```

1. The cosine similarity matrix is stored in Mata since it is likely that the dimensions of the Matrix exceed the limits of Stata.
Latent Semantic Analysis is then started by simply calling the \texttt{lsemantica} command. As the first step \texttt{lsemantica} prepares the documents and produces the document-term-matrix. During this preparation process \texttt{lsemantica} also removes words shorter than 4 characters, words that appear in less than 10 documents or more than half of all documents from the data. Furthermore, stopwords are removed from the data. The resulting document-term-matrix is then reweighed using tf-idf. The command reports every time when 500 documents have been processed.

\begin{verbatim}
. global stopwords "a able about across after all almost also am among an and a
> ny 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 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 whom wh
> y will with would yet you your"
.
. lsemantica text_strings , components(300) min_char(4) min_freq(10) max_freq(0
> .5) tfidf stopwords("$stopwords") mat_save path("$path")

*********** Latent Semantic Analysis ***********
Number of Components: 300
Minimal Word Length: 4
\end{verbatim}
Minimal Word Frequency: 10
Maximal Word Frequency: .5

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

Processing Dokument: 500
Processing Dokument: 1000
(output omitted)
Processing Dokument: 40500
Processing Dokument: 41000

If documents do not have any words left after the text cleaning lsemantica will remove these observations from the data since 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. The calculation of the Truncated Singular Value Decomposition is computationally intensive and can take some time. The time required increases with the size of the document-term-matrix and hence with the number of documents and the size of the vocabulary.

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

<table>
<thead>
<tr>
<th></th>
<th>240</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>683</td>
</tr>
<tr>
<td>3</td>
<td>1121</td>
</tr>
<tr>
<td>4</td>
<td>1219</td>
</tr>
<tr>
<td>5</td>
<td>1233</td>
</tr>
<tr>
<td>6</td>
<td>1506</td>
</tr>
<tr>
<td>7</td>
<td>1551</td>
</tr>
<tr>
<td>8</td>
<td>1841</td>
</tr>
<tr>
<td>9</td>
<td>1858</td>
</tr>
<tr>
<td>10</td>
<td>2086</td>
</tr>
<tr>
<td>11</td>
<td>2088</td>
</tr>
<tr>
<td>12</td>
<td>2408</td>
</tr>
<tr>
<td>13</td>
<td>2455</td>
</tr>
<tr>
<td>14</td>
<td>2742</td>
</tr>
</tbody>
</table>

(output omitted)

Size of Vocabulary: 3024

*************************************
**** Initializing Truncated SVD ****
*************************************
After \texttt{lsemantica} is finished running, one can begin to analyze the similarity of documents by calculating the cosine similarity between the component vectors using the \texttt{lsemantica\_cosine} command. The resulting cosine similarity matrix is only stored in Mata due to its dimensions. \texttt{lsemantica\_cosine} allows to calculate the average similarity as well as the maximal and minimal similarity to other paper titles.

\begin{verbatim}
. lsemantica\_cosine component\_1-component\_300, mean\_cosine min\_cosine max\_cos
> ine find\_similar(10) find\_similar\_cosine(10)
. sum mean\_similarity, d

mean\_similarity

<table>
<thead>
<tr>
<th>Percentiles</th>
<th>Smallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>.0044088</td>
</tr>
<tr>
<td>5%</td>
<td>.0064275</td>
</tr>
<tr>
<td>10%</td>
<td>.0078005</td>
</tr>
<tr>
<td>25%</td>
<td>.0106047</td>
</tr>
<tr>
<td>50%</td>
<td>.0144794</td>
</tr>
<tr>
<td>75%</td>
<td>.0189268</td>
</tr>
<tr>
<td>90%</td>
<td>.0232651</td>
</tr>
<tr>
<td>95%</td>
<td>.0260069</td>
</tr>
<tr>
<td>99%</td>
<td>.0316009</td>
</tr>
</tbody>
</table>

Percentiles Smallest
1% .00020578
5% .00020981
10% .0022984
25% .0023423
50% .015114
75% .0413696
90% .0430038
95% .0432377
99% .0453161

> ine sum max\_similarity, d

max\_similarity

<table>
<thead>
<tr>
<th>Percentiles</th>
<th>Smallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>.5657686</td>
</tr>
<tr>
<td>5%</td>
<td>.6288624</td>
</tr>
<tr>
<td>10%</td>
<td>.6645111</td>
</tr>
<tr>
<td>25%</td>
<td>.7309251</td>
</tr>
<tr>
<td>50%</td>
<td>.8170741</td>
</tr>
<tr>
<td>75%</td>
<td>.9474309</td>
</tr>
<tr>
<td>90%</td>
<td>.9942282</td>
</tr>
<tr>
<td>95%</td>
<td>.9985675</td>
</tr>
<tr>
<td>99%</td>
<td>1</td>
</tr>
</tbody>
</table>

Percentiles Smallest
1% .3213424
5% .3456575
10% .3466017
25% .3641868
50% .8245778
75% .122637
90% .1324817
95% .212745
99% 1

> ine sum min\_similarity, d

min\_similarity

<table>
<thead>
<tr>
<th>Percentiles</th>
<th>Smallest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>-.2534644</td>
</tr>
<tr>
<td>5%</td>
<td>-.1904553</td>
</tr>
<tr>
<td>10%</td>
<td>-.1521353</td>
</tr>
<tr>
<td>25%</td>
<td>-.1236439</td>
</tr>
<tr>
<td>50%</td>
<td>-.0937783</td>
</tr>
<tr>
<td>75%</td>
<td>-.0742603</td>
</tr>
<tr>
<td>90%</td>
<td>-.0625094</td>
</tr>
<tr>
<td>95%</td>
<td>-.0570256</td>
</tr>
<tr>
<td>99%</td>
<td>-.0486044</td>
</tr>
</tbody>
</table>

Percentiles Smallest
1% -.3954274
5% -.3954274
10% -.3914352
25% -.3914352
50% -.1047754
75% -.0279486
90% -.0279073
95% -.0254906
99% -.0253161
Furthermore, `lsemantica_cosine` can find the most similar papers for each of the papers in the data. In the example, the 10 most similar papers are calculated. Afterwards, the 5 most similar paper titles for the first paper in the data are listed. One can see that LSA accurately identified highly similar papers all discussing questions of labor supply.

```
list most_similar_* cosine_most_similar_* if _n ==1
1.                513                29462                27261                6285                34261                28462
   most_s-7                38844                29299                29316                38926                .83112737                .81672134
   cosine_-3                .67756183                .67717427                .67447494                .66636836                .66595972
   cosine_-8                .66420832                .66420832                .66381491
list title if _n==1 | _n==513 | _n==28462 | _n==27261 | _n==6285 | _n==34261
```

`lsemantica` makes it possible to calculate the number of papers the original paper is highly similar to. In the example, a cut-off for the cosine similarity of 0.75 was chosen. The Mata code generates a new variable called `high_sim_papers` containing the number of papers that have a cosine similarity above this cut-off. Estimating a regression of this newly created variable on the number of citations each paper received reveals that there is a significant positive relationship between the number of similar papers and the number of citations a paper receives.

```
sort pub_year
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 )'
```
C. Schwarz

```stata
> 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
.
.
. reg citations high_sim_paper i.pub_year

Source | SS df MS Number of obs = 41,176
---------|-----------|------|------------------------------|
Model | 34064693.9 37 920667.402 Prob > F = 0.0000
Residual | 476007139 41,138 11570.984 R-squared = 0.0668
Total | 510071833 41,175 12387.9012 Root MSE = 107.57

| Coef. Std. Err. | t | P>|t| | [95% Conf. Interval] |
---------|----------------|-----|----------------|----------------------|
high_sim_paper | 0.095991 | 0.0454516 | 2.11 | 0.035 | 0.0069049 | 0.185077 |
pub_year 1981 | 18.96728 | 15.03029 | 1.26 | 0.207 | -10.49241 | 48.42698 |
(output omitted)
pub_year 2016 | -43.50222 | 10.76639 | -4.04 | 0.000 | -64.60459 | -22.39986 |
_cons | 47.0978 | 10.51544 | 4.48 | 0.000 | 26.48731 | 67.70829 |
```

Finally, `lsemantica` makes it possible to 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’ as well as ‘monopsony’ are related to each other.

```stata
. lsemantica_word_comp using "$path/word_comp"
.
. lsemantica_cosine component_1-component_300, find_similar(10) find_similar_cosine(10)
.
. list most_similar_* cosine_most_similar_* if _n ==1
1. | most_s-1 | most_s-2 | most_s-3 | most_s-4 | most_s-5 | most_s-6 | most_s-7 | most_s-8 | most_s-9 | most_s-10 | cosine_-1 | cosine_-2 |
---------|----------|----------|----------|---------|---------|---------|----------|----------|---------|-----------|-----------|-----------|
2028 | 1663 | 1801 | 1165 | 444 | 1412 | 1412 | 1412 | 1412 | 1412 | 1412 | 1412 | 1412 |
```

## LSA in Stata

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2873</td>
<td>1415</td>
<td>1312</td>
<td>1583</td>
<td>.51891912</td>
<td>.46223733</td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cosine_3</td>
<td>cosine_4</td>
<td>cosine_5</td>
<td>cosine_6</td>
<td>cosine_7</td>
<td>cosine_8</td>
</tr>
<tr>
<td>.33289342</td>
<td>.30647979</td>
<td>.26266823</td>
<td>.23580692</td>
<td>.23367881</td>
<td>.21901752</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cosine_9</td>
<td>cosin_10</td>
</tr>
<tr>
<td>.20074844</td>
<td>.1972368</td>
</tr>
</tbody>
</table>

```
.list word if _n==1 | _n==2028 | _n==1663 | _n== 1801 | _n==1165 | _n==444

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>labor</td>
</tr>
<tr>
<td>444.</td>
</tr>
<tr>
<td>force</td>
</tr>
<tr>
<td>1165.</td>
</tr>
<tr>
<td>segmented</td>
</tr>
<tr>
<td>1663.</td>
</tr>
<tr>
<td>division</td>
</tr>
<tr>
<td>1801.</td>
</tr>
<tr>
<td>frictional</td>
</tr>
<tr>
<td>2028.</td>
</tr>
<tr>
<td>monopsony</td>
</tr>
</tbody>
</table>
```

## 5 References

Barker, M. 2012. STRDIST: Stata module to calculate the Levenshtein distance, or edit distance, between strings. *Statistical Software Components*.


**About the authors**

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. (www.carloschwarz.eu)