VecText: Converting Documents to Vectors
František Dařena, Member, IAENG

Abstract—This paper introduces a software application VecText that is used to convert raw text data into a structured format suitable for various data mining tools. VecText supports most of the common operations needed for text data preprocessing as well as not very usual functions. Its graphical user interface enables user-friendly software employment without requiring specialized technical skills and knowledge of a particular programming language together with its library names and functions. The command line interface mode, where the options are specified using the command line parameters, enables incorporating the application into a more complicated data mining process integrating several software packages or performing multiple conversions in a batch. Besides introducing the tool, the paper also summarizes various techniques that are being applied when deriving a structured representation of texts in the form of a document-term matrix and compares several popular text mining frameworks and tools.

Index Terms—VecText, text mining, natural language processing, vector space model, document preprocessing, document-term matrix.

I. INTRODUCTION

THE discipline concerned with mining useful knowledge from large amounts of text data, known as text mining, has gained great attention as the volume of available text data from many sources has been significantly increasing. Text mining involves general tasks such as text categorization, information extraction, single- or multi-document document summarization, clustering, association rules mining, or sentiment analysis [1], [2].

As the manual processing of the data is usually not feasible, automated artificial intelligence, machine learning, or statistical methods are used to solve numerous tasks. Examples of specific applications include categorization of newspaper articles or web pages, e-mail filtering, organization of a library, customer feedback handling, extracting information from lawsuit documents, competitive intelligence, extraction of topic trends in text streams, discovering semantic relations between events, or customer satisfaction analysis [3], [4], [5], [6], [7], [8], [9].

Many of the algorithms used to accomplish the tasks require the data to be converted to a structured format. Effective and efficient text mining thus heavily relies on the application of various preprocessing techniques. Their goal is to infer or extract structured representations from unstructured plain textual data [10].

A widely used structured format is the vector space model proposed by Salton [11]. Every document is represented by a vector where individual dimensions correspond to the features (usually the terms) and the values are the weights (importance) of the features. All vectors then form so-called document-term matrix where the rows represent the documents and the columns correspond to the terms in the documents. Very often, the features correspond to the words contained in the documents. Such a simple approach, known as the bag-of-words approach, is popular because of its simplicity and straightforward process of creation while providing satisfactory results [12].

Preprocessing of texts, i.e., a conversion to a structured representation, is a procedure having a significant impact on the process and results of text mining. The procedure can consist of just a few simple steps or can contain a series of advanced processing phases ordered in a particular sequence. Preprocessing methods include, e.g., online text cleaning, white space removal, case folding, spelling errors corrections, abbreviations expanding, stemming, stop words removal, negation handling, or feature selection [13], [14], [15]. Some of the natural language processing techniques, such as tokenization, stemming, part-of-speech tagging, syntactical or shallow parsing, are a subset of these methods requiring knowledge of the language to be processed [10].

What will work best for a given knowledge discovery task is not known in advance and strongly depends not only on the data but also on the preprocessing operations. Thus, a possibility to create different structured text data representations and make them ready for experimenting and finding an optimal solution is often essential.

The application of a specific preprocessing technique requires, of course, familiarity with the technique. Besides understanding the purpose and principle, one needs to know what subroutine/class/method/tool in a programming language or data mining framework where the data is analyzed to use. In the case some functions or methods in a programming language are used, their parameters and return values need to be known together with the syntax of the given language. The individual preprocessing steps also need to be arranged in a proper sequence which makes this task quite difficult. For some experiments, especially when finding an acceptable set of preprocessing techniques and their parameters, a possibility of automating the entire process with varying parameters is useful as well.

Text analytics is becoming more and more attractive not only for many commercial companies in order to, for example, get insights on customers and markets [16]. Text mining and natural language processing have become an integral part of many computer science study programs in the entire world. A tool that facilitates some basic text data processing tasks, that can be efficiently adopted by researchers and enthusiasts and that can be easily incorporated into an educational process is attractive.

This paper introduces VecText, a software package built on open-source technologies, that provides extensive functionality related to converting raw texts to a structured representation. Compared to existing tools, VecText provides sub-
stantially more preprocessing possibilities and is not bound to any particular data mining framework or programming language. VecText has been used in research activities of the author as well as in the course focused on text mining at Mendel University in Brno. Besides introducing the software tool, the paper also summarizes various approaches to derive a structured representation of texts in the form of a document-term matrix to be used by various data mining tools in a knowledge discovery process.

II. THE GENERAL PROCESS OF THE CONVERSION

The documents to be converted need to be located, read, and possibly filtered so only desired documents (e.g., documents from specified classes or documents containing relevant information) further processed. Long or semi-structured documents might be segmented into smaller pieces, such as sentences or elements delimited by tags of a markup language. Then, unwanted characters and their sequences (e.g., digits, punctuation, and other special symbols) are removed and each document or its pieces is broken down into individual tokens (useful units for processing). The tokens might be somehow transformed (e.g., stemmed, replaced by an alternative, converted to lower/upper case), assembled (into n-grams that are sequences of n successive tokens), or removed according to given rules (minimal or maximal length, local/global/document frequency, presence in a list of unwanted tokens or absence in a list of allowed tokens). The filtered or derived features, referred to as terms, later form the features of a structured representation of the documents.

Subsequently, the weights of individual terms in the documents are quantified. The weight \( w_{ij} \) of every term \( i \) in document \( j \) is determined by three components – a local weight \( lw_{ij} \), representing the frequency of term \( i \) in document \( j \), a global weight \( gw_{i} \), reflecting the discriminative ability of term \( i \), based on the distribution of the term in the entire document collection, and a normalization factor \( n_{j} \), given by the properties of document \( j \) and correcting the impact of different document lengths [11]:

\[
  w_{ij} = \frac{lw_{ij} \times gw_{i}}{n_{j}}
\]

The calculation formulae of some commonly used local and global weights and normalization factors can be found in tables I–II. The calculated values of \( w_{ij} \) then comprise the components of the document-term matrix that is being generated.

After performing some of the above-mentioned steps and calculating the values of the document-term matrix components, the data is ready for further analysis. The data is in an internal form (operational memory) of the given software package performing the preprocessing stage and some data mining algorithms might be applied to it (for example, a clustering or classification algorithm). When another data mining software will be used, the data usually needs to be stored in a file with a certain structure (format) required by the software. Besides the document-term matrix, additional files with supplementary information are sometimes required (e.g., attribute names in the c5 package [17] or class labels for cluster quality evaluation in CLUTO [18]).

III. EXISTING TOOLS

There are several tools enabling conversions of raw data into a structured form in current popular data mining frameworks available. The R programming language provides a framework for text mining applications – the tm package. This package enables loading and filtering the documents, some basic transformations (whitespace stripping, lowercase conversion, stemming, stop words removal, phrases replacement, or punctuation and numbers removal), and creating document-term matrices [19].

In Matlab, the Text to Matrix Generator (TMG) toolbox can be used for text mining tasks. The toolbox provides, besides a relatively simple basic document-term matrix creation, also other modules implementing data mining algorithms, like clustering, classification, dimensionality reduction, and others. Most of TMG is written in MATLAB, but a large segment of the indexing phase is written in Perl [20].

The StringToWordVector filter in Weka converts text strings into a set of attributes representing word occurrences in the strings. The tokens are determined according to the supplied tokenizing algorithm. The filter supports a few weighting schemes (boolean word presence, word counts, logarithm, inverse document frequency), filtering based on total word frequency (in every class/entire data set), stop-words removal, and stemming [21].

A simple script doc2mat written in Perl is used to infer a document-term matrix from a text file containing a document on every row. Porter’s stemming, stop words removal (using an internal or user-supplied list of English stop words), removing words containing numeric digits, and filtering out non-alphanumeric characters and short terms might be applied. The output is a document-term matrix in a matrix format compatible with CLUTO application [18] together with the dictionary (column labels), class labels (when applicable), and a token representation of each document after performing the tokenization and preprocessing when desired.

The NLTK library for Python provides functions for tokenization, stemming, lemmatization, sentence segmentation, parsing, or part-of-speech tagging [22]. Methods from the scikit-learn library [23] can be used for stripping accents, lowercasing, stop words filtering, composing n-grams, using an existing dictionary, filtering words according to their relative document frequency, and calculating a few weights to form a document-term matrix.

The available tools are often bound to a certain programming language or software package. This means that a data engineer needs to understand the syntax of the given language and must be able to write a code implementing the desired steps. Familiarity with the names of various modules/libraries/packages, their functions, input parameters, return values etc. is also required. Some of the tools provide only a limited number of preprocessing steps and parameters setting. It is often necessary to specify the preprocessing steps using a proper sequence of commands, too, while the user-friendliness is modest.

IV. SOFTWARE DESCRIPTION

As an alternative to existing tools, VecText provides the functionality that is not bound to specific software or programming language. It focuses only on the preprocessing
The data might be alternatively stored in directories in a
contains one original document in the specified encoding.
converted to vectors is stored in a text file where every row
B. Software Functionalities
available.
https://sourceforge.net/projects/vectext/
where the necessary
complicated data mining process integrating several software
munication enables incorporating the application into a more
mode, all preprocessing options must be specified using
into logically related blocks. In the command line interface
are specified using common graphical elements organized
of libraries and their functions, etc. All preprocessing actions
and knowledge of a particular programming language, names
The graphical user interface enables user-friendly software
be used in the text preprocessing phase from a user.
To run VecText a user needs a
interpreter installed.
There are two interfaces to the VecText core which ensures
the conversion itself. Both serve for obtaining the parameters
to be used in the text preprocessing phase from a user. The
graphical user interface enables user-friendly software
employment without requiring specialized technical skills
and knowledge of a particular programming language, names
of libraries and their functions, etc. All preprocessing actions
are specified using common graphical elements organized
into logically related blocks. In the command line interface
mode, all preprocessing options must be specified using
command line parameters. This way of non-interactive
communication enables incorporating the application into a more
complicated data mining process integrating several software
packages or performing multiple conversions in a batch.
The entire project is hosted at
https://sourceforge.net/projects/vectext/ where the necessary
resources, including documentation and a user manual, are
available.
B. Software Functionalities
The application requires that the input text data to be
converted to vectors is stored in a text file where every row
contains one original document in the specified encoding.
The data might be alternatively stored in directories in a
specified location. Then, all files in these folders will be
processed (one file is one document). The directories’ names
will be used as the first tokens in each document and could
be later used as, e.g., class labels, see Fig. 2.
A few leading tokens (pieces of text separated by spaces,
commas, or semicolons) containing, e.g., document labels,
might be skipped and not included in the further processing.
One of such tokens might represent a class label for the
document which is needed, e.g., for classification or supervised
feature selection problems, see Fig. 3. A user might also
specify what classes of documents should be later processed.
When a user wants to work with just a subset of the data,
the desired number of documents from the entire collection
might be randomly selected.
If the text of the documents is marked by an SGML
based markup language [25] only the content of selected
elements (e.g., the <text> element used in the Reuters
dataset [26]) might be processed. A user might also request
splitting the documents into sentences. At this moment,
sentences boundaries are simply determined by occurrences
of given characters (typically .?!); these characters might be
enumerated by the user.
The application performs case folding as desired (no case
folding, converting to upper or lower case) and filters out
non-alphanumeric characters, while optionally keeping user-
provided symbols (e.g., abbreviations), numbers, or emoti-
cons. If the user provides a list of stop words [1] these words
are excluded from further processing. When a dictionary (a
list of allowed words) is supplied the words that are not in
it are eliminated. This is useful, e.g., when creating a test
data set that must have the same attributes as the training
set, or when it makes sense to use a dictionary from a given
domain.
When supplied, rules for replacing some text with another
(e.g., “European Union” → “EU” are applied. Another
technique used to modify the original words is stemming.
A stemming procedure is based on Snowball, a language
designed for creating stemming algorithms [27], and stemming
rules implemented for any natural language might be used.

Fig. 1. The architecture of VecText.
The price is toooo <b>high</b>&lt;/b>!!! It exceeds $ 100 \rightarrow 
\begin{align*}
&\text{replacement rules applied (e.g., $ \rightarrow USD)} \\
&\text{The price is toooo high!!! It exceeds USD} \rightarrow \\
&\text{special characters and numbers replaced} \\
&\text{The price is toooo high!!! It exceeds USD sad} \\
&\text{collapsing repeating characters} \\
&\text{The price is tooo high} \rightarrow \\
&\text{replacing short words (minimal length = 3)} \\
&\text{The price too high} \rightarrow \\
&\text{removing stop words} \\
&\text{price high exceeds USD sad} \\
&\text{stemming} \\
&\text{price high exceed USD sad} \\
&\text{converting to lower case} \\
&\text{price high exceed usd sad}
\end{align*}

Fig. 4. An example of a process of raw text transformation. Pieces of the text that are subject to modification are emphasized before the transformation.

Words with the length longer or shorter than desired might be filtered out. Single words might be combined into sequences of successive words, known as n-grams. A user might specify the value of n and thus generate 2-grams, 3-grams, etc., including their combinations (e.g., generating 2- and 3-grams together).

An example of a text transformation with a few pre-processing techniques applied can be found in Fig. 4.

The user can choose from a wide variety of local and global weights and normalizations. Local weighting include Binary (Term Presence), Term Frequency, Logarithmic, Augmented Normalized Term Frequency (with parameter K specification), Okapi’s TF factor [28], Normalized Logarithm, Square root, Augmented logarithm, Augmented average TF, Changed-coefficient average TF [29], Alternate Logarithm [30], Squared TF, DFR-like normalization [31], and Thresholded TF. For a detailed description of these weights see table I.

Global weighting possibilities include Inverse Document Frequency (IDF) [32], Probabilistic Inverse Document Frequency, Global frequency IDF, Entropy, Log-global frequency IDF, Incremented global frequency IDF, Square root global frequency IDF [29], Inverse total term frequency [32], and Squared IDF [31], see table II.

Normalization schemes, described in table III, include Cosine [29], Max Weight, Sum of Weights, Fourth Normalization [30], Max TF, Square root, and Logarithm [18].

When a logarithm is needed to calculate a weight, a user decides whether to use the common or natural one. Considering the number of term occurrences, a user might specify a minimal and maximal local frequency (in a document), global frequency (in the entire collection), document frequency for the terms, the number of most frequent words to keep, the percentage of the words with highest document frequency to keep in the dictionary, or relative document frequency.

The output is a document-term matrix in the desired format created according to the user-specified rules. The supported output formats include Attribute-Relation File Format (ARFF), eXtensible Attribute-Relation File Format (XRRF), both also in sparse alternations [21], Comma-separated values (CSV) [33], sparse and dense matrix formats for software packages Cluto [18], c4.5 or c5 [34], SVMlight [12], and

---

**TABLE I**

<table>
<thead>
<tr>
<th>Weight name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary (Term Presence)</td>
<td>0 if ( f_{ij} = 0 ) 0 if ( f_{ij} &gt; 0 )</td>
</tr>
<tr>
<td>Term Frequency (TF)</td>
<td>( f_{ij} )</td>
</tr>
<tr>
<td>Thresholded TF</td>
<td>0 if ( f_{ij} = 0 ) 1 if ( f_{ij} = 1 ) 2 if ( f_{ij} &gt;= 2 )</td>
</tr>
<tr>
<td>Logarithm</td>
<td>0 if ( f_{ij} = 0 ) ( \log (f_{ij} + 1) ) if ( f_{ij} &gt; 0 )</td>
</tr>
<tr>
<td>Alternate Logarithm</td>
<td>0 if ( f_{ij} = 0 ) 1 + ( \log f_{ij} ) if ( f_{ij} &gt; 0 )</td>
</tr>
<tr>
<td>Normalized Logarithm</td>
<td>0 if ( f_{ij} = 0 ) ( \log (f_{ij} + 1) ) if ( f_{ij} &gt; 0 )</td>
</tr>
<tr>
<td>Augmented Normalized TF</td>
<td>( k + (1 - k) \frac{f_{ij}}{N} ) if ( f_{ij} &gt; 0 )</td>
</tr>
<tr>
<td>Square Root</td>
<td>0 if ( f_{ij} = 0 ) ( \sqrt{f_{ij} - 0.5 + 1} ) if ( f_{ij} &gt; 0 )</td>
</tr>
<tr>
<td>Augmented Logarithm</td>
<td>0 if ( f_{ij} = 0 ) ( k + (1 - k) \frac{\log f_{ij} + 1}{\log N} ) if ( f_{ij} &gt; 0 )</td>
</tr>
<tr>
<td>Augmented Average TF</td>
<td>0 if ( f_{ij} = 0 ) ( k ) if ( f_{ij} &gt; 0 )</td>
</tr>
<tr>
<td>DFR-like Normalization</td>
<td>( f_{ij} * \frac{N}{\sum_{j=1}^{N} f_{ij}} )</td>
</tr>
<tr>
<td>Okapi’s TF Factor</td>
<td>( \frac{f_{ij}}{P_{ij}} )</td>
</tr>
</tbody>
</table>

**TABLE II**

<table>
<thead>
<tr>
<th>Weight name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>Inverse Document Frequency (IDF)</td>
<td>( \log \frac{N}{n_{i}} )</td>
</tr>
<tr>
<td>Squared IDF</td>
<td>( \log^{2} \frac{n_{i}}{N} )</td>
</tr>
<tr>
<td>Probabilistic IDF</td>
<td>( \log N \frac{n_{i}}{n_{i}} )</td>
</tr>
<tr>
<td>Global frequency IDF</td>
<td>( \frac{N}{n_{i}} )</td>
</tr>
<tr>
<td>Entropy</td>
<td>( 1 + \sum_{j=1}^{N} \frac{f_{ij}}{P_{ij}} \log \frac{f_{ij}}{P_{ij}} \log N )</td>
</tr>
<tr>
<td>Incremented global frequency IDF</td>
<td>( \frac{N}{n_{i}} + 1 )</td>
</tr>
<tr>
<td>Log-global frequency IDF</td>
<td>( \log \left( \frac{N}{n_{i}} + 1 \right) )</td>
</tr>
<tr>
<td>Square root global frequency IDF</td>
<td>( \sqrt{\frac{N}{n_{i}} - 0.9} )</td>
</tr>
<tr>
<td>Inverse total term frequency</td>
<td>( \log \sum_{j=1}^{N} f_{ij} )</td>
</tr>
</tbody>
</table>
The document-term matrix was written to a file with the same filestem as the input file had. The file was placed in the same directory where was the input file located. To cluster the reviews, the CLUTO application was used so the output format was set to “CLUTO (sparse)” (a space-saving variant storing only non zero values as the vectors representing the texts are generally very sparse). To obtain some information about the nature of the documents, statistical information was printed too. To be able to later process the content of the reviews with knowledge of the clusters they belonged to, the original content of the documents fulfilling the specified criteria needed to be stored as well.

Because very rare terms usually play no or very little role during the analysis, terms that appeared in less than four documents were filtered out together with terms having just one character. Stopwords contained in the supplied list were removed as well. To assign a weight to document features, the popular tf-idf weighting scheme with cosine normalization was used. The calculated numbers were printed with three decimal places.

Another example is an analysis of statistical properties of texts of movie reviews from the IMDB database [37]. This information can be found in the file with the dictionary and statistics, see Fig. V, generated by VecText upon request. No vectors representing the documents needed to be created so only the preprocessing phase was carried out. In the set of 10,000 randomly selected documents, the most frequent words expectedly contain “the”, “a”, “and”, “of” and so like. On the 19th position, with the global frequency equal to about two-thirds of the frequency of the most frequent word, appeared the word “movie”. Seven positions further with slightly smaller frequency was another not typical stop word – “film”.

VI. DISCUSSION

Converting raw texts to a format suitable for further analysis is a procedure having a significant impact on the process and results of knowledge discovery. The procedure can be very simple or can consist of many carefully selected preprocessing steps arranged in a specific order. It is not possible to determine what techniques should be used in advance. Everything strongly depends on the analyzed data. For example, in two almost identical tasks but with data from different domains (hotel accommodation and medical service), different preprocessing techniques needed to be applied in order to obtain meaningful results [38], [39]. A possibility to create different structured representations of documents and experimenting with them is therefore often crucial.

Text mining has become a very topical discipline with applications in many domains. Besides the classical data mining software applications, specialized tools for processing unstructured text are required by researchers, practitioners, or even students. Existing tools are generally bound to a specific programming language (e.g., Python, R, Matlab) and thus require knowledge of the language so a user can write the necessary code transforming raw texts to a format suitable for further analysis. Besides knowing, what library, function, or parameters to use, one needs to understand the principles (e.g., object-orientation design) and syntax (i.e., control flow statements, data types, variables) of the language.

<table>
<thead>
<tr>
<th>Weight name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>Cosine</td>
<td>$\sqrt{\sum_{i=1}^{m} (gw_i \cdot lw_{ij})^2}$</td>
</tr>
<tr>
<td>Sum of weights</td>
<td>$\sum_{i=1}^{m} gw_i \cdot lw_{ij}$</td>
</tr>
<tr>
<td>Max weight</td>
<td>$\max gw_i \cdot lw_{ij}$</td>
</tr>
<tr>
<td>Max TF</td>
<td>$0.5 + 0.5 \cdot \frac{gw_i \cdot lw_{ij}}{\max gw_i \cdot lw_{ij}}$</td>
</tr>
<tr>
<td>Square root</td>
<td>$\sqrt{gw_i \cdot lw_{ij}}$</td>
</tr>
<tr>
<td>Logarithim</td>
<td>$\log gw_i \cdot lw_{ij}$</td>
</tr>
<tr>
<td>Fourth normalization</td>
<td>$\sum_{i=1}^{m} (gw_i \cdot lw_{ij})^4$</td>
</tr>
</tbody>
</table>

The attributes in the document-term matrix are sorted alphabetically or according to the document frequency.

Besides the vectors representing the processed documents, a user decides on generating a dictionary file (optionally with global term frequencies or global term frequencies for individual classes), simple statistics (containing the numbers of documents in individual classes, the number of unique terms, minimal, maximal, and average document lengths, and the variance), original documents satisfying the specified preprocessing rules (e.g., documents from given classes, containing only allowed words), and a file with the generated terms (tokens). It is also possible to execute just the preprocessing phase and not to produce the document-term matrix in order to only generate a dictionary, filter the documents according to some rules, clean the texts and prepare them for further processing.

To help the users define all necessary and desired parameters for the command line mode, the application with the graphical interface enables generating the string with command line parameters based on the current values of all form elements in the application window. These parameter settings are returned in the form of a text string and might be useful in the application window. These parameter settings are returned in the form of a text string and might be useful in the application window.
that might be discouraging for certain group of people. Other tools, like Weka or doc2mat, provide a graphical or command line interface to specify all necessary parameters for the conversion of texts to vectors. Their possibilities are, however, quite limited and the user-friendliness is often moderate. A table comparing the properties of a few tools can be found in Tab. IV.

VecText eliminates many disadvantages of the existing tools for converting texts to a structured format. On the other hand, it is fair to say that the solutions that are a part of a programming language are more flexible because a programmer can change whatever operation in the entire process and has complete control over it. Because VecText provides so many options, it also runs significantly slower than in a standard bag of words model, try to maximize the corpus likelihood [41]. Popular models proposed by Mikolov [42] include the CBOW (continuous bag of words) and skipgram models. CBOW tries to predict current word based on its context while the skipgram model predicts words in the context. The inputs and outputs of both neural models are one-hot encoded vectors (vector where only one out of its units is 1 and all others are 0) [43]. To prepare the data for embeddings training, some preprocessing can be applied too [44], [45] so VecText is relevant to this domain as well.

VII. CONCLUSIONS

The paper introduces a software application VecText that is used to convert raw text data into a structured vector format according to the user supplied rules and requirements. It supports most of the operations needed for common text data preprocessing tasks as well as not very usual functions, both adjustable by user-defined parameters. Working in two modes, graphical and command line, it enables uncomplicated use in the interactive or batch modes.

The application is based on open source technologies and might be easily extended or modified by researchers with programming skills. It is available for many operating systems thanks to the implementation in the interpreted programming language Perl.

The usability has been proven by an application in the research of the author during the last years, by many cooperating students, and in the educational process at the university where the author active.

ACKNOWLEDGMENT

The author would like to acknowledge the support of the Czech Science Foundation, grant No. 17-259245 “Comparative Study of Crowdfunding Projects in EU: Access to Finance, Risks and Regulation”.

REFERENCES

Fig. 6. An example of a VecText application in research.


TABLE IV

<table>
<thead>
<tr>
<th>Property</th>
<th>VecText</th>
<th>tm</th>
<th>TMG</th>
<th>StringToWordVector</th>
<th>doc2mat</th>
<th>NLTk, scikit-learn</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of local weights</td>
<td>17</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>No. of global weights</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>No. of normalization factors</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>No. of input formats</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(file, directory)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(filehash database, URL, directory)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of output formats</td>
<td>11</td>
<td>N/A</td>
<td>N/A</td>
<td>1</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>GUI</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Repeated programming skills</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Custom transformations</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Programming language</td>
<td>perl</td>
<td>R</td>
<td>matlab, perl</td>
<td>java</td>
<td>perl</td>
<td>python</td>
</tr>
</tbody>
</table>


František Daˇrena (M’15) František Daˇrena works as an associate professor and head of the Text Mining and Natural Language Processing group at the Department of Informatics, Faculty of Business and Economics, Mendel University Brno, he is a member of SoNet research center, author of several publications in international scientific journals, conference proceedings, and monographs, member of editorial boards of international journals, and editor-in-chief of International Journal in Foundations of Computer Science & Technology (IJFCST). His research areas include text/data mining, intelligent data processing, and machine learning.