# 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-

Manuscript received October 24, 2018; revised January 7, 2019. This work was supported by the Czech Science Foundation, grant No. 17-25924S "Comparative Study of Crowdfunding Projects in EU: Access to Finance, Risks and Regulation".

F. Dařena is with the Department of Informatics, Faculty of Business and Economics, Mendel University Brno, Brno, 613 00, Czech Republic, e-mail: frantisek.darena@mendelu.cz.

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]:

$$w_{ij} = \frac{lw_{ij} * gw_i}{n_j}$$

The calculation formulae of some commonly used local and global weights and normalization factors can be found in tables I–III. 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), stopwords 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

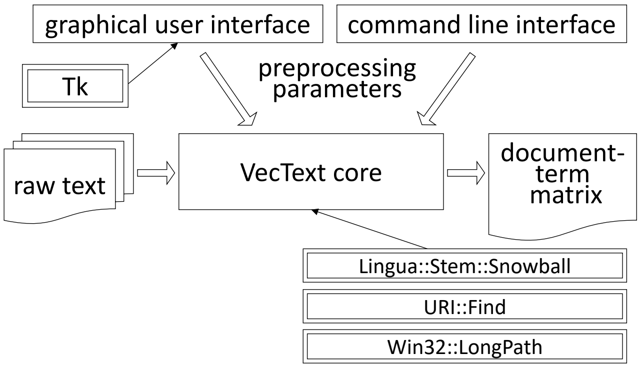


Fig. 1. The architecture of VecText.

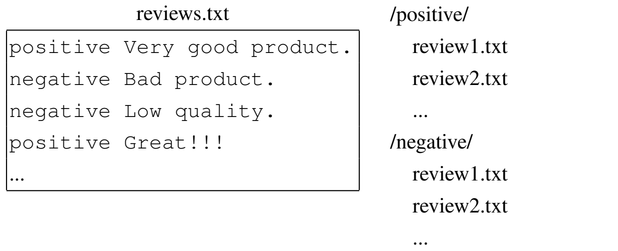


Fig. 2. Raw texts as the source data: left – in one file, right – in multiple files and directories.

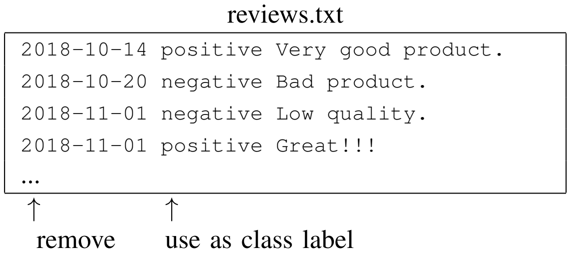


Fig. 3. Skipping initial two tokens, the second will be used as a class label.

phase and can be used to prepare data for a wide variety of machine learning tools and programming languages. It enables choosing from many techniques to be applied to data without the necessity of specifying their order. Numbers of options for these parameters enable finding a suitable structured representation for a given task. A graphical user interface can be used to even simplify the task for not much-experienced researchers.

### A. *Software Architecture*

The application is written in the interpreted programming language Perl [24] which runs on more than 100 platforms. To run VecText a user needs a *perl* interpreter installed.

A part of VecText's functionality is implemented by proven external modules (e.g., *Lingua::Stem::Snowball* for stemming or *Win32::LongPath* for working with long file names on Windows) freely available at the Comprehensive Perl Archive Network (CPAN)(*www.cpan.org*). The graphical user interface is implemented in *Perl/Tk*, a widely used graphical interface for Perl. This extension and related libraries can be also obtained from the CPAN archive.

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 emoticons. 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.

The price is toooo <**b**>high</**b**>!!! It exceeds $ 100 :-(

*SGML tags and entities removed*

The price is toooo high!!! It exceeds **$** 100 :-(

*replacement rules applied (e.g., $ → USD)*

The price is toooo high!!! It exceeds USD **:-(**

*preserving emoticons (e.g., :-( → sad)*

The price is toooo high**!!!** It exceeds USD sad

*special characters and numbers replaced*

The price is t**oooo** high It exceeds USD sad

*collapsing repeating characters*

The price **is** too high **It** exceeds USD sad

*replacing short words (minimal length = 3)*

**The** price **too** high exceeds USD sad

*removing stop words*

price high exceed**s** USD sad

*stemming*

price high exceed **USD** sad

*converting to lower case*

price high exceed usd sad

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 preprocessing 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.

TABLE I
CALCULATING THE LOCAL WEIGHT FOR TERM $i$ IN DOCUMENT $j$ ($f_{ij}$ IS THE FREQUENCY OF TERM $i$ IN DOCUMENT $j$, $a_j$ IS $average(f_j)$, $x_j$ IS $max(f_j)$, $k$ IS A USER SPECIFIED CONSTANT BETWEEN 0 AND 1, $l_{avg}$ IS THE AVERAGE DOCUMENT LENGTH, $l_j$ IS THE LENGTH OF DOCUMENT $j$).

| Weight name | Value |
|---|---|
| Binary (Term Presence) | 0 if $f_{ij} = 0$<br>1 if $f_{ij} > 0$ |
| Term Frequency (TF) | $f_{ij}$ |
| Squared TF | $f_{ij}^2$ |
| Thresholded TF | 0 if $f_{ij} = 0$<br>1 if $f_{ij} = 1$<br>2 if $f_{ij} >= 2$ |
| Logaritm | 0 if $f_{ij} = 0$<br>$\log(f_{ij} + 1)$ if $f_{ij} > 0$ |
| Alternate Logarithm | 0 if $f_{ij} = 0$<br>$1 + \log f_{ij}$ if $f_{ij} > 0$ |
| Normalized Logarithm | 0 if $f_{ij} = 0$<br>$\frac{1+\log f_{ij}}{1+\log a_j}$ if $f_{ij} > 0$ |
| Augmented Normalized TF | 0 if $f_{ij} = 0$<br>$k + (1-k)\left(\frac{f_{ij}}{x_j}\right)$ if $f_{ij} > 0$ |
| Changed-coefficient Average TF | 0 if $f_{ij} = 0$<br>$k + (1-k)\frac{f_{ij}}{x_j}$ if $f_{ij} > 0$ |
| Square Root | 0 if $f_{ij} = 0$<br>$\sqrt{f_{ij} - 0.5} + 1$ if $f_{ij} > 0$ |
| Augmented Logarithm | 0 if $f_{ij} = 0$<br>$k + (1-k)\log(f_{ij+1})$ if $f_{ij} > 0$ |
| Augmented Average TF | 0 if $f_{ij} = 0$<br>$k + (1-k)\frac{f_{ij}}{a_j}$ if $f_{ij} > 0$ |
| DFR-like Normalization | $f_{ij} * \frac{l_{avg}}{l_j}$ |
| Okapi's TF Factor | $\frac{f_{ij}}{2+f_{ij}}$ |

TABLE II
CALCULATING GLOBAL WEIGHTS FOR TERM $i$ ($N$ IS THE NUMBER OF DOCUMENTS IN THE COLLECTION, $n_i$ IS THE NUMBER OF DOCUMENTS CONTAINING TERM $i$ (DOCUMENT FREQUENCY), $f_{ij}$ IS THE FREQUENCY OF TERM $i$ IN DOCUMENT $j$, $F_i$ IS THE GLOBAL FREQUENCY OF TERM $i$, AND $l_j$ IS THE LENGTH OF DOCUMENT $j$).

| Weight name | Value |
|---|---|
| None | 1 |
| Inverse Document Frequency (IDF) | $\log \frac{N}{n_i}$ |
| Squared IDF | $\log^2 \frac{N}{n_i}$ |
| Probabilistic IDF | $log \frac{N-n_i}{n_i}$ |
| Global frequency IDF | $\frac{F_i}{n_i}$ |
| Entropy | $1 + \sum_{j=1}^{N} \frac{\frac{f_{ij}}{F_i} \log \frac{f_{ij}}{F_i}}{\log N}$, |
| Incremented global frequency IDF | $\frac{F_i}{n_i} + 1$ |
| Log-global frequency IDF | $\log\left(\frac{F_i}{n_i} + 1\right)$ |
| Square root global frequency IDF | $\sqrt{\frac{F_i}{n_i} - 0.9}$ |
| Inverse total term frequency | $\log \frac{\sum_{j=1}^{N} l_j}{F_i}$ |

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 (XRFF), 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 III
NORMALIZATION OF WEIGHTS IN DOCUMENT $j$ ($gw_i$ IS A VALUE OF THE GLOBAL WEIGHT OF TERM $i$, $lw_{ij}$ IS A VALUE OF THE GLOBAL WEIGHT OF TERM $i$ IN DOCUMENT $j$, $m$ IS THE NUMBER OF TERMS IN DOCUMENT $j$, AND $f_{ij}$ IS THE FREQUENCY OF TERM $i$ IN DOCUMENT $j$).

| Weight name | Value |
|---|---|
| None | 1 |
| Cosine | $\sqrt{\sum_{i=1}^{m}(gw_i * lw_{ij})^2}$ |
| Sum of weights | $\sum_{i=1}^{m} gw_i * lw_{ij}$ |
| Max weight | $\max gw_i * lw_{ij}$ |
| Max TF | $0.5 + 0.5 * \frac{gw_i * lw_{ij}}{\max gw_i * lw_{ij}}$ |
| Square root | $\sqrt{gw_i * lw_{ij}}$ |
| Logarithm | $\log gw_i * lw_{ij}$ |
| Fourth normalization | $\sum_{i=1}^{m}(gw_i * lw_{ij})^4$ |

the Yale sparse matrix format [35]. If desired, the order of vectors representing the processed documents on the output is randomized. 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 simply copied to, e.g., a batch file or script.

## V. ILLUSTRATIVE EXAMPLES

The application can be used to infer a structured representation from an arbitrary number of texts having various forms (long/short, formal/informal, with/without HTML tags, etc.) related to any domain. One example, see the screenshot in Fig. 6, shows the usage of VecText in a research focused on automatic extraction of product features from customer reviews published on *amazon.com* [36]. The collection consisting of several tens of thousands of reviews needed to be transformed into a structured representation to be further analyzed by data mining algorithms (clustering, feature selection and ranking). The reviews were placed in a text file where every row contained the number of stars assigned by a customer, a publication date, and the text of the review in the UTF-8 encoding. For the task, only the review text was relevant so the first two tokens were skipped.

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 researches, 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

Dictionary:

```
#WORD   TOTAL   neg     pos
the     9913    4901    5012
a       9681    4779    4902
and     9660    4741    4919
of      9480    4683    4797
to      9360    4671    4689
this    9063    4567    4496
is      8938    4373    4565
it      8906    4423    4483
in      8794    4295    4499
that    8094    4088    4006
...
movie   6136    3296    2840
...
film    5539    2681    2858
...
```

Statistics:

```
Preprocessing parameters
========================
input directory: C:/VecText/imdb
minimal word length: 0
minimal global word frequency: 4
...

Data set characteristics
========================
10000 documents
20938 unique attributes

class: neg
----------
4934 documents
19529 unique attributes
terms number: min 10, max 1507,
avg 227.286988244832, var 26831.2042206415

class: pos
----------
5066 documents
20044 unique attributes
terms number: min 12, max 1809,
avg 232.741610738255, var 31105.5848338839
```

Fig. 5. The files with the dictionary and statistical information about the texts for 10,000 randomly selected texts from the IMDB reviews database.

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 tools implementing only some basic functionality like, for example, doc2mat. VecText is, therefore, more applicable in prototyping a text mining solution and in applications that are not critical from the performance point of view.

The usability of VecText has been proven by applications in the research of the text mining group at the Mendel University during a few last years (see, e.g., [40], [6]), in research projects of many cooperating students, in student's theses, and in the educational process (text mining course) at the university where is the author active.

Besides the bag of words model, another popular format for representing texts has gained attention in the last few years. It is based on continuous word vectors, known as word embeddings, learned using a neural network model. The vectors, that have a significantly lower number of dimensions 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 universities where is the author active.

## Acknowledgment

The author would like to acknowledge the support of the Czech Science Foundation, grant No. 17-25924S "Comparative Study of Crowdfunding Projects in EU: Access to Finance, Risks and Regulation".

## References

[1] S. M. Weiss, N. Indurkhya, T. Zhang, and F. J. Damerau, *Text Mining: Predictive Methods for Analyzing Unstructured Information*. New York, NY: Springer New York, 2005.

[2] B. Liu, *Sentiment Analysis and Opinion Mining*. Morgan & Claypool Publishers, 2012.

[3] X. Rao. and Z. Ke, "Hierarchical rnn for information extraction from lawsuit documents," in *Proceedings of the International MultiConference of Engineers and Computer Scientists 2018 Vol I*, 2018.

[4] S. A. Yousif, V. W. Samawi, and I. Elkabani, "Arabic text classification: The effect of the awn relations weighting scheme," in *Proceedings of the World Congress on Engineering 2017 Vol II*, 2017.

[5] M. M. Al-Daeef, N. Basir, and M. M. Saudi, "Evaluation of phishing email classification features: Reliability ratio measure," in *Proceedings of the World Congress on Engineering 2017 Vol I*, 2017.

[6] J. Žižka and F. Dařena, "Automated mining of relevant n-grams in relation to predominant topics of text documents," in *Text, Speech, and Dialogue: 18th International Conference, TSD 2015, Pilsen, Czech Republic, September 14-17, 2015*, P. Král and V. Matoušek, Eds. Springer, 2015, pp. 461–469.

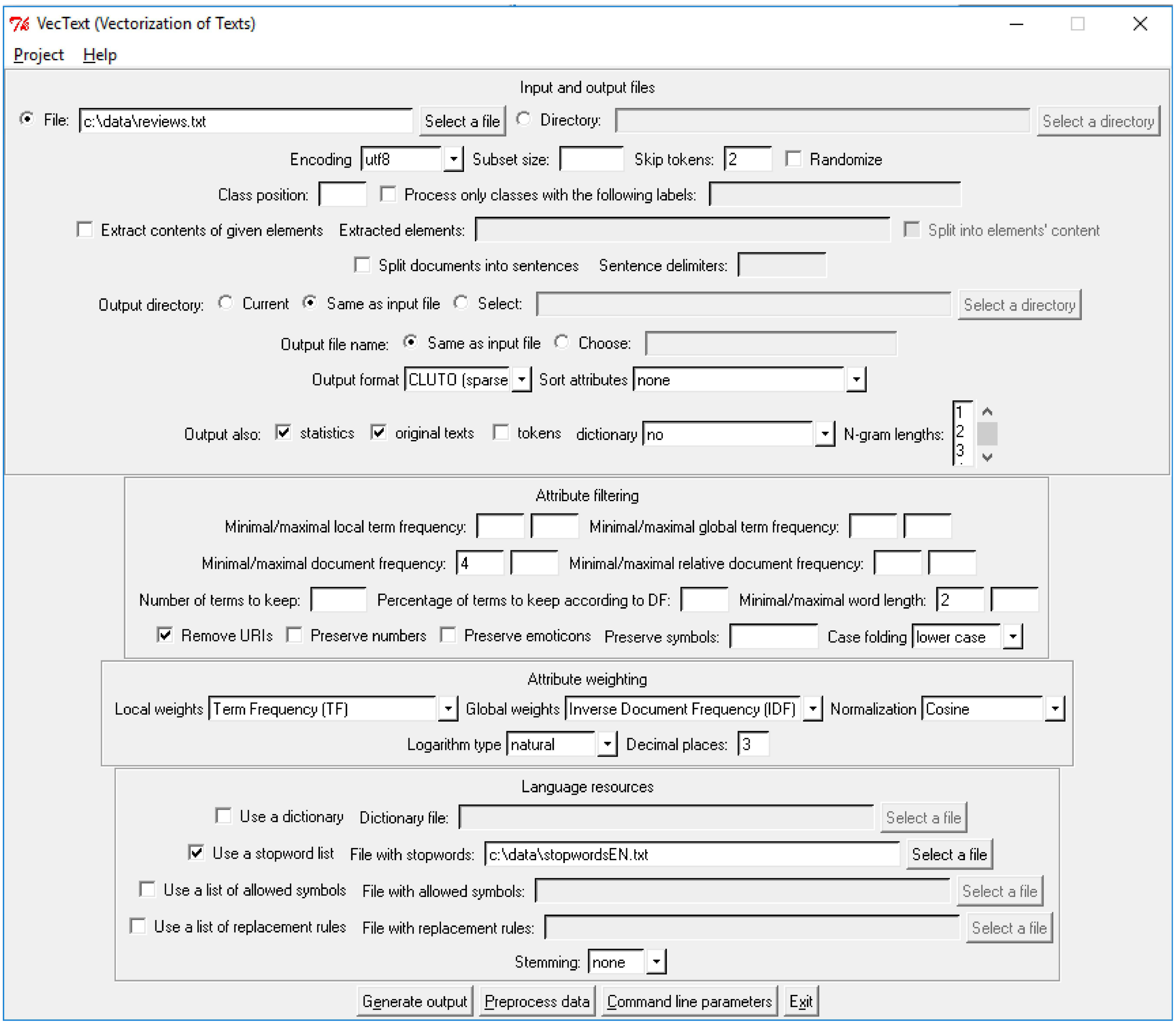


Fig. 6. An example of a VecText application in research.

[7] A. Y. Al-Obaidi and V. W. Samawi, "Opinion mining: Analysis of comments written in arabic colloquial," in *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering and Computer Science 2016*, 2016, pp. 470–475.

[8] T. L. Georgieva-Trifonova, M. E. Stefanova, and S. D. Kalchev, "Customer feedback text analysis for online stores reviews in bulgarian," *IAENG International Journal of Computer Science*, vol. 45, no. 4, pp. 560–568, 2018.

[9] A. S. Patil and B. V. Pawar, "A machine learning approach for classification of internet web sites," *IAENG Transactions on Electrical Engineering Volume 1: Special Issue of the International Multiconference of Engineers and Computer Scientists 2012*, pp. 137–147, 2012.

[10] R. Feldman and J. Sanger, *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, 2007.

[11] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York: McGraw Hill, 1983.

[12] T. Joachims, *Learning to classify text using support vector machines*. Norwell: Kluwer Academic Publishers, 2002.

[13] G. Carvalho, D. M. Matos, and V. Rocio, "Document retrieval for question answering: A quantitative evaluation of text preprocessing," in *PIKM'07*. ACM, 2007, pp. 125–130.

[14] E. Clark and K. Araki, "Text normalization in social media: progress, problems and applications for a pre-processing system of casual english," *Procedia – Social and Behavioral Sciences*, vol. 27, pp. 2–11, 2011.

[15] E. Haddi, X. Liu, and Y. Shi, "The role of text pre-processing in sentiment analysis," *Procedia – Computer Science*, vol. 17, pp. 26–32, 2013.

[16] J.-P. Isson and J. Harriott, *Win with Advanced Business Analytics: Creating Business Value from Your Data*. Hoboken: Wiley, 2012.

[17] RuleQuest. (2015) Data mining tools see5 and c5.0. [Online]. Available: http://www.rulequest.com/see5-info.html

[18] G. Karypis, "Cluto: A clustering toolkit," University of Minnesota, Department of Computer Science, Technical Report 02-017, 2003.

[19] I. Feinerer, K. Hornik, and D. Meyer, "Text mining infrastructure in r," *Journal of Statistical Software*, vol. 25, no. 5, pp. 1–54, 2008.

[20] D. Z. E. Gallopoulos, "Tmg: A matlab toolbox for generating term-document matrices from text collections," in *Grouping Multidimensional Data: Recent Advances in Clustering*. Heidelberg: Springer, 2005, pp. 187–210.

[21] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical machine learning tools and techniques*. San Francisco: Morgan Kaufmann, 2011.

[22] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*. Association for Computational Linguistics, 2002, pp. 62–69.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[24] T. Christiansen, b. d foy, L. Wall, and J. Orwant, *Programming Perl: Unmatched power for text processing and scripting*, 4th ed. Sebastopol: O'Reilly, 2012.

[25] ISO, "Information processing – text and office systems – standard generalized markup language (sgml)," International Organization for Standardization, Geneva, Switzerland, ISO 8879:1986, 1986.

[26] T. Rose, M. Stevenson, and M. Whitehead, "The reuters corpus volume 1 - from yesterday's news to tomorrow's language resources,"

TABLE IV
COMPARISON OF THE PROPERTIES OF EXISTING TOOLS FOR CONVERTING RAW TEXTS TO A STRUCTURED REPRESENTATION.

| Property | VecText | tm | TMG | StringToWordVector | doc2mat | NLTK, scikit-learn |
|---|---|---|---|---|---|---|
| No. of local weights | 17 | 5 | 5 | 1 | 1 | 3 |
| No. of global weights | 9 | 2 | 5 | 2 | 0 | 2 |
| No. of normalization factors | 7 | 3 | 1 | 2 | 0 | 2 |
| No. of input formats | 2 (file, directory) | 3 (filehash database, URI, directory) | 2 (file, directories) | 2 (file, directory) | 1 (file) | 2 (memory, file) |
| No. of output formats | 11 | N/A | N/A | N/A | 1 | N/A |
| Reguires programing skills | no | yes | yes | no | no | yes |
| GUI | yes | no | yes | yes | no | no |
| Custom transformations | no | yes | no | no | no | yes |
| Programming language | perl | R | matlab, perl | java | perl | python |

in *Proceedings of the Third International Conference on Language Resources and Evaluation*, 2002, pp. 29–31.
[27] M. F. Porter, “Snowball: A language for stemming algorithms,” 2006. [Online]. Available: http://www.snowball.tartarus.org/texts/stemmersoverview.html
[28] A. K. Singhal, “Term weighting revisited,” Ph.D. dissertation, Faculty of the Graduate School of Cornell University, 1997.
[29] E. Chisholm and T. G. Kolda, “New term weighting formulas for the vector space method in information retrieval,” Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, Tech. Rep. ORNL/TM-13756, 1999.
[30] N. Polettini, “The vector space model in information retrieval – term weighting problem,” 2004.
[31] P. Tirilly, V. Claveau, and P. Gros, “A review of weighting schemes for bag of visual words image retrieval,” Research Report PI 1927, 2009.
[32] S. Robertson, “Understanding inverse document frequency: On theoretical arguments for idf,” *Journal of Documentation*, vol. 60, no. 5, pp. 503–520, 2004.
[33] Y. Shafranovich, “Common format and mime type for comma-separated values (csv) files,” RFC 4180, October 2005.
[34] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
[35] R. E. Banky and C. C. Douglas, “Sparse matrix multiplication package (smmp),” *Advances in Computational Mathematics*, vol. 1, pp. 127–137, 1993.
[36] K. Barák, F. Dařena, and J. Žižka, “Automated extraction of typical expressions describing product features from customer reviews,” *European Journal of Business Science and Technology*, vol. 1, pp. 83–92, 2015.
[37] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, “Learning word vectors for sentiment analysis,” in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150.
[38] J. Žižka and F. Dařena, “Mining significant words from customer opinions written in different natural languages,” in *Text, Speech and Dialogue*, I. Habernal and V. Matoušek, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 211–218.
[39] F. Dařena, J. Žižka, and J. Přichystal, “Clients freely written assessment as the source of automatically mined opinions,” *Procedia Economics and Finance*, vol. 12, pp. 103–110, 2014, 17th International Conference Enterprise and Competitive Environment 2014.
[40] J. Žižka and F. Dařena, “Automatic sentiment analysis using the textual pattern content similarity in natural language,” in *Text, Speech, and Dialogue, 2010 September 6-10; Brno, Czech Republic*, I. Kopeček and K. Pala, Eds. Springer, 2010, pp. 224–231.
[41] Y. Li, L. Xu, F. Tian, L. Jiang, X. Zhong, and E. Chen, “Word embedding revisited: A new representation learning and explicit matrix factorization perspective,” in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*. AAAI Press, 2015, pp. 3650–3656.
[42] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *CoRR*, vol. abs/1301.3781, 2013.
[43] X. Rong, “word2vec parameter learning explained,” *CoRR*, vol. abs/1411.2738, 2014.
[44] Q. Li, S. Shah, X. Liu, and A. Nourbakhsh, “Data sets: Word embeddings learned from tweets and general data,” in *Proceedings of the Eleventh International AAAI Conference on Web and Social Media (ICWSM 2017)*, 2017, pp. 428–436.
[45] A. Leeuwenberg, M. Vela, J. Dehdari, and J. van Genabith, “A minimally supervised approach for synonym extraction with word embeddings,” *The Prague Bulletin of Mathematical Linguistics*, no. 105, pp. 111–142, 2016.

**František Dařena (M’15)** František Dařena 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.