VecText

VecText is an application that converts raw text to a structured format suitable for various data mining software (e.g., Weka, C5, CLUTO). The application is written in the interpreted programming language Perl which runs on more than 100 platforms. A part of the functionality is realized by external modules (e.g., Lingua::Stem::Snowball for stemming) 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 can be also obtained from the CPAN archive.

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 options need to be specified using the 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 application consists of the following parts:

- VecText.pm – a module with implementing the conversion process of texts to vectors,
- vectext.pl – a graphical user interface to the application functionality,
- vectext-cmdline.pl – a command line interface to the application functionality,
- Tk – a directory with some of the necessary modules.

The application might require the following modules available at CPAN:

- Tk::FileDialog, Tk::DirSelect, Tk::Help for GUI,
- Win32::LongPath for forking with long paths and Unicode in the Windows environment,
- List::Util to randomize input,
- Cwd to get the pathname of current working directory,
- Lingua::Stem::Snowball for stemming,
- URI::Find for URI removal capability.

The process of conversion

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 documents in these folders will be processed. The directories' names will be used as the first tokens in each document and be later used as, e.g., class labels.

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 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. 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 a desired number of documents from the entire collection might be randomly selected.

If the text of the documents is marked by a SGML based markup language only the content of selected elements (e.g., the <text> element used in the Reuters data set 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 stopwords 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 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. The user might also remove URIs from the texts.

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. Stemming procedure is based on Snowball, a language designed for creating stemming algorithms, and stemming rules implemented for any natural language might be used.

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).

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, Normalized Logarithm, Square root, Augmented logarithm, Augmented average TF, Changed-coefficient average TF, Alternate Logarithm, Squared TF, DFR-like normalization, Thresholded TF. For a detailed description of these weights see Table 1.

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 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.

An example of running the application in command line mode:

```
perl vectext-cmdline.pl --input=data.txt --output_dir=. --output_file=data
                        --local_weights="Term Frequency (TF)" --output_format=arff
                        --print_statistics --create_dictionary=”with frequencies”
                        --stopwords_file=stop.txt --stemming=English
```

The list of available parameters and their meaning follows. In the graphical mode all parameters are set using common graphical elements.

input, input_file (string, optional)
Specifies the name of a file with raw texts to be converted to vectors. There is one document on each line.

read_directory (string, optional)
When a directory to be read is supplied all directories in it are scanned and all filed in these folders will be processed (one file = 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.

encoding (string, optional)
Specifies the input and output encodings for the data files read and produced by the application. Available values include 'ascii', 'iso-8859-1', 'iso-8859-2', and 'utf8'.

subset, subset_size (integer, optional)
Specifies the size of the generated data set in terms of documents number. When not provided, all documents from the source file are processed.

skip_tokens (integer, optional)
When specified, the desired number of tokens at the beginning of every document is skipped and excluded from further processing. This is suitable, e.g., in situations when there the documents contain headers or other meta-data at the beginning. In this step, tokens are considered to be strings separated by one or more whitespaces, a comma, or a semicolon.

class_position (integer, optional)
This parameter specifies the position of a token that determines the class of a document. If the value of this parameter is less or equal to the number of skipped tokens, the token will be preserved. In this step, tokens are considered to be strings separated by one or more whitespaces, a comma, or a semicolon.

randomize (Boolean, optional)
When selected, the output instances will be randomized and not in the same order as they are on the input.

processed_classes (string, optional)
A list of comma or whitespace separated values might be provided. In that case, only documents with these supplied labels will be processed.

output_dir (string, optional)
The name of a directory where the output (term-document matrix and possibly some additional files) will be stored.

output_file (string, mandatory)
The name of output file. Several files might be generated, e.g., term-document matrix (in one or more files, according to the output format), tokens, some statistics etc. This name will be extended by pre-defined extensions, like arff, names, data, tokens, etc.

local_weights (string, mandatory)
Specifies what kind of local weighting will be used. Allowed values are listed in table 1.

global_weights (string, optional)
Specifies what kind of global weighting will be used. Allowed values are listed in table 2.

normalization (string, optional)
Specifies what kind of local weighting will be used. Allowed values are listed in table 3.

min_word_length (integer, optional)
Specifies the minimal length (in characters) of a word to appear in the output. Shorter words are filtered out.

max_word_length (integer, optional)
Specifies the maximal length (in characters) of a word to appear in the output. Longer words are filtered out.

min_global_frequency (integer, optional)
Specifies the minimal global frequency of a word to appear in the output. Words that appear less times (thus are very rare) are filtered out.

max_global_frequency (integer, optional)
Specifies the maximal global frequency of a word to appear in the output. Words that appear more times (thus are very common) are filtered out.

min_local_frequency (integer, optional)
Specifies the minimal local frequency of a word (i.e., in one document) to appear in the output. Words that appear less times (thus are very rare) are filtered out.

max_local_frequency (integer, optional)
Specifies the maximal local frequency of a word (i.e., in one document) to appear in the output. Words that appear more times (thus are very common) are filtered out.

min_document_frequency (integer, optional)
Specifies the minimal document frequency of a word (i.e., the number of documents where is the word contained) to appear in the output. Words that appear less times (thus are very rare) are filtered out.

max_document_frequency (integer, optional)
Specifies the maximal document frequency of a word (i.e., the number of documents where is the word contained) to appear in the output. Words that appear more times (thus are very common) are filtered out.

min_rel_document_frequency (real, optional)
Specifies the minimal relative document frequency of a word (i.e., the percentage of documents where is the word contained) to appear in the output. Words that appear less times (thus are rare) are filtered out. For example, if the value is set to 0.05, words that don't appear in at least 5% of documents will be removed from further processing.

max_rel_document_frequency (real, optional)
Specifies the maximal relative document frequency of a word (i.e., the percentage of documents where is the word contained) to appear in the output. Words that appear more times (thus are very common) are filtered out. For example, if the value is set to 0.9, words that appear in at least 90% of documents will be removed from further processing.

terms_to_keep (integer, optional)
When provided, only the desired number of terms with highest global frequency are kept in the dictionary.

df_percentage (real, optional)
This parameter specifies what percentage of terms with highest document frequency will be kept in the dictionary. If the value is set to, e.g., 0.05, only 5 percent of terms with highest document frequency will remain in the dictionary.

output_format (string, compulsory)
Specifies an output format of the term-document matrix. Allowed values are:

- ARFF (Attribute-Relation File Format) used by the Weka machine learning software,
- Sparse ARFF (Attribute-Relation File Format) used by the Weka machine learning software,
- XRFF (eXtensible attribute-Relation File Format) is an XML-based extension of the ARFF format,
- Sparse XRFF – a sparse representation of the XRFF format,
- C5 required by C5/See5 software package (a file with extension names describing attributes and classes and a file with extension data containing training data for the classifier),
- SPARSE – sparse matrix format, comma separated list of attribute number-attribute vale pairs, followed by a class label,
- CSV – comma separated list of attribute values followed by a class label, with attribute names on the first row,
- CLUTO – sparse matrix format for software packages Cluto, creating also the rlabel file (file with class labels) when applicable,
- SVMlight - sparse matrix format for software packages SVMlight,
- YALE - Yale sparse matrix format.

n_grams (string, optional)
When provided, sequences consisting of the specified numbers of subsequent words will be treated as attributes (for example, for n_grams=2, the attributes derived from this paragraph will include when_provided, provided_sequences, sequences_consisting, etc.). It is possible to specify more n-gram sizes by supplying a comma separated list, e.g. –n_grams=2,3.

create_dictionary (string, optional)
When desired, a dictionary is written to a file. The dictionary contains one token per line, optionally with followed by term or document frequencies of the token. Allowed values of this parameter are 'plain' (only the list of tokens is written), 'with frequencies' (a list of tokens and their global frequencies, ordered according to the frequency in descending order, is written to a file with extension freq.dict), 'with frequencies for classes' (a list of tokens and their global frequencies and frequencies in individual classes, ordered according to the global frequency in descending order, is

written to a file with extension freq.dict), 'with document frequencies' (a list of tokens and their document frequencies, ordered according to the document frequency in descending order, is written to a file with extension df.dict), and 'with document frequencies for classes' (a list of tokens and their document frequencies and document frequencies in individual classes, ordered according to the document frequency in descending order, is written to a file with extension df.dict).

print_statistics (Boolean, optional)
Some algorithm parameters and data set statistics are written to a file with extension stat.txt. The statistics include the total number of processed documents and unique attributes. For each processed class, the number of documents, unique attributes, and information about term numbers (minimal, maximal, average, variance) in the documents from the given class are provided.

dictionary_file (string, optional)
When a dictionary (a list of allowed words) is supplied the words that are not in the dictionary are eliminated. This is useful, e.g., when creating 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. The name of the file containing the allowed words is passed using this parameter. In the file, there is a single word on each line.

replacement_rules_file (string, optional)
When supplied, rules for replacing some text with another (e.g., "European Union" -> "EU") are applied. The name of the file containing the allowed words is passed using this parameter. In the file, there is a single rule on each line in the form “ORIGINAL => REPLACE\n”.

stopwords_file (string, optional)
When stopwords are supplied they are filtered out before further processing. The name of the file containing stopwords is passed using this parameter. In the file, there is a single word on each line.

allowed_symbols_file (string, optional)
Some expressions (symbols), like abbreviations, might include symbols that would be eliminated (e.g., string “U.S.A.” would be transformed to three terms “U”, “S”, and “A”). When a list of allowed symbols is supplied these symbols are preserved in the text. The name of the file containing allowed symbols is passed using this parameter. In the file, there is a single expression on each line.

logarithm_type (string, optional)
When the logarithm is used in calculations of term weights (e.g., global weight inverse document frequency, or local weight logarithm, alternate logarithm, etc.) a user might decide what will be the base of the logarithm. The two allowed options include natural (the base is e) and common (with base 10).

tags (string, optional)
If the text of the documents is marked by a SGML based markup language (e.g., HTML or XML) only the content of selected elements (e.g., the <text> element used in the Reuters data set) might be processed. The element names are passed as a comma or space separated list.

split_into_elements (Boolean, optional)
When the content of selected elements is extracted this parameter determines whether every such extracted content will become a separate document or not.

sentences (string, optional)
A user might also request splitting the documents into sentences. At this moment, sentence boundaries are simply determined by occurrences of given characters. These characters might be enumerated by the user, for example by passing :!?; as the value of this parameter.

print_original_texts (Boolean, optional)
Original documents satisfying the specified preprocessing rules (e.g., documents from given classes, containing only allowed words) are written to a file with extension original.txt.

print_tokens (Boolean, optional)
A file with the documents containing only the terms that satisfy the specified preprocessing rules, or the derived terms, like stemmed words or n-grams, are written to a file with extension tokens.txt.

print_skipped_tokens (Boolean, optional)
If some tokens should be skipped (see the "skip_tokens" option) the files with original texts (see the "print_original_texts" option) or tokens (see the "print_tokens" option) will contain also the skipped tokens.

preprocess_data (Boolean, optional)
When desired, only the preprocessing phase that finishes before term-document matrix creation is completed. All additional files, like the dictionary, statistics, a file with tokens, etc., are created when a user specifies so. This parameter is suitable when it is necessary to discover some characteristics of the data, like the list of unique terms and their frequencies after all filtering parameters (like minimal word length, minimal or maximal document frequency, stemming, etc.) are applied.

remove_multiple_characters (Boolean, optional)
When set to true, all appearances of the same character more than two times in a row are replaced by two appearances. For example, "goooooood" is replaced by "good".

remove_URIs (Boolean, optional)
When desired, every appearance of a URI (e.g., *http://www.example.com/page.html*) is removed from the text.

preserve_numbers (Booelan, optional)
Normally, all non-letters, including numbers, are removed from the texts. When desired, numbers will remain in the texts and are treated as words. Numbers have optional + or – sign, several digits, possibly followed by a decimal point (in that case no digits are necessary before the decimal point) and several decimal places, optionally followed by the semi-logarithmic notation (e or E followed by a positive or negative exponent containing at least one digit).

case (string, optional)
Specifies, what case folding transformation should be applied to the documents. Available options include 'no' (original case is preserved),'lower case', and 'upper case'.

preserve_symbols (string, optional)
Normally, all non-letters are removed from the texts. When desired, some of the symbols (characters) might be preserved. These characters are included in the string passed as this parameter. For example, when +:$ is supplied, all plus signs, colons, and dollar signs remain in the texts and are treated as words.

preserve_emoticons (Boolean, optional)
When desired, all emoticons (strings like :-), :-(, and others) in the texts will be replaced by their textual alternatives. The list of allowed emoticons is located in the file emoticons.txt located in the application directory. The content of the file might be modified so the user defined emoticons will be used. Every row of the file contains one emoticon and its textual representation, e.g., ":D laughing", or ":| straight face". All occurrences of ":D" will be then replaced by string "laughing", ":|" will be replaced by "straight_face", etc.

output_decimal_places (integer, optional)
When real numbers appear as the vectors' components, this parameter specifies the number of places after the decimal point. When not specified, 3 is used as the default value.

sort_attributes (string, optional)
This parameter defines how the attributes will be sort in all output files (term-document matrix, dictionary, etc.). Allowed values include 'none' (no explicit sorting is used, the attributes are sorted according to the internal representation in computer memory, depending on the data), 'alphabetically', 'alphabetically inversely', 'document frequency (descending)', and 'document frequency (ascending)'.

stemming (string, optional)
When desired, stemming is applied to the input data. Stemming algorithms are implemented through a Perl interface to the C version of the Snowball stemmers. Supported languages (used as parameter values) include 'danish','dutch','english','finnish','french','german', 'hungarian', 'italian','norwegian', 'portuguese', 'romanian', 'russian', 'spanish', 'swedish', and 'turkish'.

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

Table 2: 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	Calculation
No weight	$gw_i = 1$
Inverse Document Frequency (IDF)	$gw_i = \log \frac{N}{n_i}$
Squared IDF	$gw_i = \log^2 \frac{N}{n_i}$
Probabilistic IDF	$gw_i = log \frac{N-n_i}{n_i}$
Global frequency IDF	$gw_i = \frac{F_i}{n_i}$
Entropy	$gw_i = 1 + \sum_{j=1}^{N} \frac{\frac{f_{ij}}{F_i} \log \frac{f_{ij}}{F_i}}{\log N},$
Incremented global frequency IDF	$gw_i = \frac{F_i}{n_i} + 1$
Log-global frequency IDF	$gw_i = \log \left(\frac{F_i}{n_i} + 1 \right)$
Square root global frequency IDF	$gw_i = \sqrt{\frac{F_i}{n_i} - 0.9}$
Inverse total term frequency	$gw_i = \log \frac{\sum_{j=1}^{N} l_j}{F_i}$

Table 3: Normalization of weights in document j (g_{wi} 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	Calculation
None	$n_j = 1$
Cosine	$n_j = \sqrt{\sum_{i=1}^{m} (gw_i * lw_{ij})^2}$
Sum of weights	$n_j = \sum_{i=1}^{m} gw_i * lw_{ij}$
Max weight	$n_j = \max gw_i * lw_{ij}$
Max TF	$n_j = 0.5 + 0.5 * \frac{gw_i * lw_{ij}}{\max gw_i * lw_{ij}}$
Square root	$f_{ij} = \sqrt{f_{ij}}$ if $f_{ij} \geq 0$ $f_{ij} = -\sqrt{\text{abs}(f_{ij})}$ if $f_{ij} < 0$
Logarithm	$f_{ij} = \log f_{ij}$ if $f_{ij} \geq 0$ $f_{ij} = -\log \text{abs}(f_{ij})$ if $f_{ij} < 0$
Fourth normalization	$n_j = \sum_{i=1}^{m} (gw_i * lw_{ij})^4$