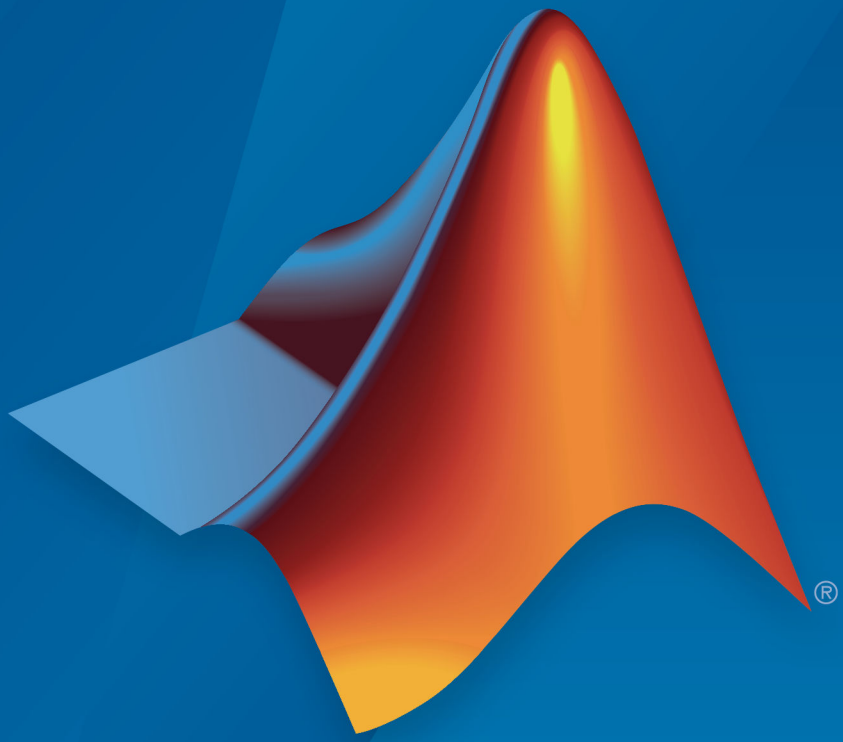
Text Analytics Toolbox™ Release Notes



MATLAB®

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Text Analytics Toolbox™ Release Notes

Contents

R2019b

R2019a

R2018b

R2018a

R2017b

R2019b

Version: 1.4

New Features

Bug Fixes

Compatibility Considerations

Korean Language Support: Perform text analytics on Korean language text including tokenization, lemmatization, part-of-speech tagging, and named entity recognition

Analyze Korean text using Text Analytics Toolbox. The following functions support Korean input:

- `tokenizedDocument`
- `normalizeWords`
- `removeStopWords`
- `addSentenceDetails`
- `addPartOfSpeechDetails`
- `addEntityDetails`
- `wordcloud`
- `wordCloudCounts`
- `splitSentences`
- `corpusLanguage`

For more information, see “Korean Language Support”.

Sentiment Analysis: Evaluate sentiment in text data using sentiment scoring algorithms including VADER

Analyze sentiment in text by evaluating the sentiment scores using the following scoring functions:

- `vaderSentimentScores` – Evaluate scores with the Valence Aware Dictionary and sEntiment Reasoner (VADER) algorithm.
- `ratioSentimentScores` – Evaluate scores with a ratio rule.

Specify custom lexicons which contain words and scores using the option.

When using `vaderSentimentScores`, you can further customize the VADER algorithm using the following options:

- Specify boosters (words like "very" and "really") using the `'Boosters'` option.

- Specify dampeners (words like "almost" and "somewhat") using the `'Dampeners'` option.
- Specify negations (words like "not" and "despite") using the `'Negations'` option.

Japanese and Korean Tokenization: Specify MeCab dictionary options

Customize Japanese and Korean tokenization by specifying options using the `mecabOptions` function. Specify the MeCab system and user models using the `Model` and `UserModel` options, respectively. Specify extractors for lemma, part-of-speech, and named entity details using the `LemmaExtractor`, `POSExtractor`, and `NERExtractor` options respectively.

To tokenize using the specified MeCab tokenization options, use the `'TokenizeMethod'` option of `tokenizedDocument`.

Deep Learning: Initialize word embedding layer with pretrained word embeddings

Initialize word embedding layer with pretrained word embeddings using the `Weights` option of `wordEmbeddingLayer` and create the corresponding word encoding directly from the word embedding vocabulary. For an example, see “Create Word Encoding from Word Embedding”.

Tokenization: Recompute sentence, part-of-speech, language, type, and named entity token details

The `addSentenceDetails`, `addPartOfSpeechDetails`, `addLanguageDetails`, `addTypeDetails`, and `addEntityDetails` functions, by default, do not recompute the details when contained in the documents. To recompute the corresponding token details, set the `'DiscardKnownValues'` option to `true`.

Deep Learning: Train Network For Language Translation using Attention

Train a deep learning network for language translation using a custom training loop. For an example showing how to train a network that translates decimals into roman numerals, see “Sequence-to-Sequence Translation Using Attention”.

Functionality Being Removed or Changed

tokenizedDocument detects Korean language
Behavior change

Starting in R2019b, `tokenizedDocument` detects the Korean language and sets the `'Language'` option to `'ko'`. This changes the default behavior of the `addSentenceDetails`, `addPartOfSpeechDetails`, `removeStopWords`, and `normalizeWords` functions for Korean document input. This change allows the software to use Korean-specific rules and word lists for analysis. If `tokenizedDocument` incorrectly detects text as Korean, then you can specify the language manually by setting the `'Language'` name-value pair of `tokenizedDocument`.

In previous versions, `tokenizedDocument` usually detects Korean text as English and sets the `'Language'` option to `'en'`. To reproduce this behavior, manually set the `'Language'` name-value pair of `tokenizedDocument` to `'en'`.

R2019a

German Language Support: Perform text analytics on German language text including tokenization, stop word removal, stemming, and part-of-speech tagging

Analyze German text using Text Analytics Toolbox. The functions `tokenizedDocument`, `addSentenceDetails`, `addPartOfSpeechDetails`, `addEntityDetails`, `removeStopWords`, and `normalizeWords` now support German input. For more information, see German Language Support.

For an example, see Analyze German Text Data.

Edit Distance: Find similarity between strings and documents using Levensthein distance and other distance measures

Compute the edit distance between strings and documents using the `editDistance` function. This function computes the number of grapheme (human perceived character) or word insertions, deletions, swaps, and substitutions to transform one string or document to another.

Create edit distance searchers to perform nearest neighborhood search in a list of known strings, using edit distance. To create an edit distance searcher, use the `editDistanceSearcher` function. To use the edit distance searcher to find the nearest neighbors, or neighbors within a specified range, use the `knnsearch` and `rangesearch` functions respectively.

For an example showing how to use edit distance searchers for spelling correction, see Correct Spelling Using Edit Distance Searchers.

Named Entity Recognition: Detect locations, organizations, people's names, and other named entities in text

Detect named entities in English, Japanese, and German text using the `addEntityDetails` function. To get the entity tags from the documents, use the `tokenDetails` function.

For an example, see Add Named Entity Tags to Documents.

Tokenization and Preprocessing: Specify and detect patterns of custom tokens and replace words or phrases in tokenized documents

Detect custom tokens and custom token types by specifying a list of tokens or regular expressions using the `'CustomTokens'` and `'RegularExpressions'` options of `tokenizedDocument` respectively.

For an example, see Specify Custom Tokens.

Deep Learning Examples: Explore deep learning workflows (requires Deep Learning Toolbox)

Lean how to classify text data using convolutional neural network (CNN) or out-of-memory text data using transformed datastores.

- Classify Text Data Using Convolutional Neural Network
- Classify Out-of-Memory Text Data Using Deep Learning

Tokenization: Replace words and n-grams in documents

Find and replace words and n-grams in documents using the `replaceWords` and `replaceNgrams` functions respectively.

For an example, see Replace Words in Documents.

Multiword Phrases: Search documents for n-gram occurrences

Search documents for n-gram occurrences and view them in context using the `context` function.

For an example, see Search Documents for N-Gram Occurrences.

Functionality Being Removed or Changed

`tokenizedDocument` detects German language
Behavior change

Starting in R2019a, `tokenizedDocument` detects the German language and sets the `'Language'` option to `'de'`. This changes the default behavior of the `addSentenceDetails`, `addPartOfSpeechDetails`, `removeStopWords`, and `normalizeWords` functions for German document input. This change allows the software to use German-specific rules and word lists for analysis. If `tokenizedDocument` incorrectly detects English language text as German, then you can specify the English language manually by setting the `'Language'` name-value pair of `tokenizedDocument` to `'en'`.

In previous versions, `tokenizedDocument` usually detects German text as English and sets the `'Language'` option to `'en'`. To reproduce this behavior, manually set the `'Language'` name-value pair of `tokenizedDocument` to `'en'`.

R2018b

Version: 1.2

New Features

Bug Fixes

Compatibility Considerations

Japanese Language Support: Perform text analytics on Japanese language text, including tokenization, stop word removal, lemmatization, and part-of-speech tagging

Analyze Japanese text using Text Analytics Toolbox. The functions `tokenizedDocument`, `removeStopWords`, `normalizeWords`, and `addPartOfSpeechDetails` now support Japanese input. For more information, see Japanese Language Support.

For an example, see Analyze Japanese Text Data.

Word Normalization: Convert words to their dictionary form using lemmatization with parts of speech and other information

Lemmatize English and Japanese language text using the `'Style'` option of the `normalizeWords` function. For an example showing how to preprocess your text data using lemmatization and other techniques, see Prepare Text Data for Analysis.

Part-of-Speech Tagging: Identify parts of speech, such as adjectives, adverbs, nouns, and verbs

Add English and Japanese language part-of-speech tags to documents using the `addPartOfSpeechDetails` function. To get the part-of-speech tags from the documents, use the `tokenDetails` function.

Deep Learning: Train deep learning networks using word embedding layers (requires Deep Learning Toolbox)

For data to train deep learning networks, convert documents to sequences using the `doc2sequence` function and `wordEncoding` objects. Train word embeddings inside a deep learning network using word embedding layers. To create a word embedding layer, use the `wordEmbeddingLayer` function. For an example showing how to train a deep learning network for text classification using a word embedding layer, see Classify Text Data Using Deep Learning.

HTML Parsing: Extract HTML from specific parts of a web page using HTML structure and CSS classes

Parse HTML code using `htmlTree` objects. To find particular HTML elements using CSS selectors, use the `findElement` function. To get the attributes from HTML elements, use the `getAttribute` function.

Tokenization: Detect emoticons and emoji characters

Analyze text containing emoticons and emoji characters using `tokenizedDocument`. This function, by default, automatically detects emoticons and emoji characters and assigns the token types `'emoticon'` and `'emoji'`. To view the token types in of the tokens in documents, use the `tokenDetails` function. To learn more, see Analyze Text Data Containing Emojis.

Sentiment Analysis Example: Learn how to analyze sentiment in text

For an example showing how to train a classifier for sentiment analysis, using an annotated list of positive and negative sentiment words and a pretrained word embedding, see Train a Sentiment Classifier.

Deep Learning Examples: Learn about generating text and working with out-of-memory text data (requires Deep Learning Toolbox)

Use examples to learn about different applications of text analytics with deep learning. New examples include:

- Pride and Prejudice and MATLAB
- Word-By-Word Text Generation Using Deep Learning
- Classify Out-of-Memory Text Data Using Custom Mini-Batch Datastore

Functionality Being Removed or Changed

erasePunctuation skips complex tokens
Behavior change

Starting in R2018b, for `tokenizedDocument` input, `erasePunctuation`, by default, erases punctuation and symbol characters from tokens with type `'punctuation'` or `'other'` only. This prevents the function from affecting complex tokens such as URLs and email addresses.

In previous versions, `erasePunctuation` erases punctuation characters from all tokens. To reproduce this behavior, use the `'TokenTypes'` name-value pair in `erasePunctuation`.

normalizeWords skips complex tokens
Behavior change

Starting in R2018b, for `tokenizedDocument` input, `normalizeWords` normalizes tokens with type `'letters'` or `'other'` only. This prevents the function from affecting complex tokens such as URLs and email addresses.

In previous versions, `normalizeWords` normalizes all tokens. To reproduce this behavior, use the command `newDocuments = docfun(@(str) normalizeWords(str),documents)`.

tokenizedDocument does not split at slash and colon characters between digits
Behavior change

Starting in R2018b, if slash, backslash, or colon characters appear between two digits, then `tokenizedDocument` does not split at these characters. This behavior produces better results when tokenizing text containing dates and times.

In previous versions, `tokenizedDocument` splits at these characters. To reproduce this behavior, tokenize the text manually, or insert whitespace characters around slash, backslash, and colon characters before using `tokenizedDocument`.

tokenizedDocument does not split emoticons
Behavior change

Starting in R2018b, `tokenizedDocument`, by default, detects emoticon tokens. This behavior makes it easier to analyze text containing emoticons.

In R2017b and R2018a, `tokenizedDocument` splits emoticon tokens into multiple tokens. To reproduce this behavior, in `tokenizedDocument`, specify the `'DetectPatterns'` option to be `{'email-address','web-address','hashtag','at-mention'}`.

tokenDetails returns token type emoji for emoji characters
Behavior change

Starting in R2018b, `tokenizedDocument` detects emoji characters and the `tokenDetails` function reports these tokens with type `"emoji"`. This makes it easier to analyze text containing emoji characters.

In R2018a, `tokenDetails` reports emoji characters with type `"other"`. To find the indices of the tokens with type `"emoji"` or `"other"`, use the indices `idx = tdetails.Type == "emoji" | tdetails.Type == "other"`, where `tdetails` is a table of token details.

fitlda sorts topics
Behavior change

Starting in R2018b, `fitlda`, by default, sorts the topics in descending order of the topic probabilities of the input document set. This behavior makes it easier to find the topics with the highest probabilities.

In previous versions, `fitlda` does not change the topic order. To reproduce the behavior, set the `'TopicOrder'` option to `'unordered'`.

ismember will be removed
Warns

To update your code, for `wordEmbedding` object input, change the function name from `ismember` to `isVocabularyWord`. You do not need to change the arguments. The syntaxes are equivalent.

R2018a

Multiword Phrases: Extract and count multiword phrases (n-grams) from tokenized text

You can extract and count multiword phrases (n-grams) from tokenized text using `bagOfNgrams` objects. For an example showing how to analyze text using n-grams, see Analyze Text Data Using Multiword Phrases.

HTML Text: Extract text content from HTML pages.

Extract text directly from HTML code in a string using `extractHTMLText`. To extract text content from HTML files, use `extractFileText`.

Deep Learning: Learn how to use deep learning LSTM networks for text classification (requires Neural Network Toolbox)

An LSTM network is a type of deep learning network that can learn long-term dependencies between time steps of sequence data. By treating text data as sequences of words, you can use deep learning techniques with your text data.

To learn how to use deep learning long short-term memory (LSTM) networks for text classification, see Classify Text Data Using Deep Learning.

Pattern Detection: Detect sentences, email addresses, and URLs in text

You can detect complex tokens such as email addresses, web addresses, hashtags, and at-mentions using the `'DetectPatterns'` option in `tokenizedDocument`. Use `splitSentences` to split text into sentences, and `addSentenceDetails` to add sentence information to tokenized documents. To get information about the tokens in a `tokenizedDocument` array, use `tokenDetails`.

Stochastic LDA Model Training: Fit LDA models to large datasets

Fit latent Dirichlet allocation (LDA) models to large datasets using stochastic approximate variational Bayes (SAVB) by specifying the `'Solver'` name-value pair to be `'savb'` in

`fitlda`. This solver is best suited for large datasets and can fit a good model in fewer passes through the data. For an example showing how to compare LDA solvers, see Compare LDA Solvers.

Pretrained Word Embedding: Download pretrained fastText word embedding

You can download a pretrained fastText word embedding using `fastTextWordEmbedding`. This function requires Text Analytics Toolbox Model *for FastText English 16 Billion Token Word Embedding* support package. If this support package is not installed, the function provides a download link.

Word Frequency Counting: Count words and n-grams in parallel (requires Parallel Computing Toolbox)

You can create multiple bag-of-words or bag-of-n-grams models in parallel and combine them using `join`. For an example showing how to create a bag-of-words model in parallel, see Create Bag-of-Words Model in Parallel.

Functionality Being Removed or Changed

Functionality	Result	Use Instead	Compatibility Considerations
`tokenizedDocument`	Still runs	Not applicable	In R2018a, `tokenizedDocument`, by default, detects complex tokens (email addresses, web addresses, hashtags, and at-mentions). In R2017b, `tokenizedDocument` splits complex tokens into multiple tokens. To reproduce this behavior, in `tokenizedDocument`, specify the `'DetectPatterns'` option to be `'none'`.

R2017b

Version: 1.0

New Features

Text Preprocessing: Prepare text for analysis by automatically extracting and preprocessing words from raw text

You can perform the following common character level preprocessing steps to prepare text data before splitting it into words:

- Erase HTML and XML tags using `eraseTags`.
- Erase URLs using `eraseURLs`.
- Erase punctuation using `erasePunctuation`.
- Convert HTML and XML entities into characters using `decodeHTMLEntities`.

After character level preprocessing, you can split text into words using `tokenizedDocument` which creates an array of `tokenizedDocument` objects. With a `tokenizedDocument` array, you can perform the following word level preprocessing steps:

- Remove specified words from an array of documents using `removeWords`.
- Remove a common list of stop words which are not useful for analysis (such as "a" and "the") using `removeWords` and `stopWords`.
- Remove long and short words using `removeLongWords` and `removeShortWords` respectively.
- Stem words using `normalizeWords`.

For an example showing how to preprocess text data and prepare for it for analysis, see Prepare Text Data for Analysis.

Machine Learning Algorithms: Discover topics and clusters of documents using Latent Dirichlet Allocation (LDA) and Latent Semantic Analysis (LSA)

You can analyze text data using the Latent Dirichlet Allocation topic model. Latent Dirichlet Allocation models a collection of documents as mixtures of topics.

Fit an `ldaModel` using `fitlda`. You can resume training using `resume`. Using `ldaModel` objects, you can perform the following tasks:

- Visualize topics and word importance of an LDA model using `wordcloud` and `topkwords`.

- Extract features, or reduce dimensionality using `transform`. This function transforms documents into the lower dimensional topic probability space.
- Predict top topics of documents using `predict`.
- Calculate document log probabilities and detect outliers using `logp`.

You can also use Latent Semantic Analysis to model your text data.

Fit an `lsaModel` using `fitlsa`. To use an LSA model as a feature extractor, or a dimension reducing tool, use `transform`. This function transforms documents into a lower dimensional semantic space.

For an example showing how to use LDA to analyze text data, see Analyze Text Data Using Topic Models. For more information on LSA models, see `lsaModel`.

Word Embeddings: Convert words to numeric vectors using word2vec, FastText, and GloVe word embedding models

Use word embeddings to discover relationships between words. Word embeddings model words as vectors in a fixed dimensional space. For example, a word embedding may learn the relationship "king" – "man" + "woman" = "queen".

Create a `WordEmbedding` object by using one of the following methods:

- Import word embedding files from word2vec, FastText, and GloVe using `readWordEmbedding`.
- Train your own word embeddings from text data using `trainWordEmbedding`.

With a `WordEmbedding` object, you can do the following:

- Map words to vectors and back using `word2vec` and `vec2word`.
- Write the word embedding to a file using `writeWordEmbedding`.

For an example showing how to explore word embeddings, see Visualize Word Embedding Using Text Scatter Plots.

Text Plots: Visualize text data using word clouds and text scatter plots

Text Analytics Toolbox extends the functionality of the `wordcloud` (MATLAB®) function. It adds support for the following tasks:

- Create word clouds directly from string. `wordcloud` automatically tokenizes, preprocesses, and counts word frequencies of string input.
- Create word clouds from bag-of-words models.
- Create word clouds from LDA topics.

You can also visualize text data using 2-D and 3-D text scatter plots. Use `textscatter` and `textscatter3` to plot words at specified coordinates of 2-D and 3-D scatter plots respectively.

For an example showing how to visualize collections of text data using word clouds, see Visualize Text Data Using Word Clouds.

Document Import: Read text from PDF and Microsoft Word files

You can extract text data directly from plain text, PDF, and Microsoft® Word files using `extractFileText`.

For an example showing how to extract text data from files and import it into MATLAB, see Extract Text Data From Files.

Text Statistics: Calculate word frequency and TF-IDF matrices from document collections

A bag-of-words model (also known as a term-frequency counter) records the number of times that words appear in each document of a collection.

Create a `bagOfWords` object using `bagOfWords`.

With a `bagOfWords` object, you can perform the following tasks:

- Encode documents as a matrix of word counts using `encode`.

- View the most frequent words using `topkwords`.
- Add and remove documents using `addDocument` and `removeDocument` respectively.
- Remove empty documents using `removeEmptyDocuments`.
- Remove infrequent words using `removeInfrequentWords`.

You can input `bagOfWords` objects directly into `fitlda`, `fitlsa`, and `wordcloud`.

You can create tf-idf matrices from a bag-of-words model using `tfidf`. A tf-idf matrix is a statistic that captures word importance in a collection of documents. It captures the number of times each word appear in a collection, and how many documents each word appears in.

For more information, see `bagOfWords`.

Word Normalization: Convert words to their word roots using the Porter stemming algorithm

To group different forms of English words by reducing them to a common stem, use `normalizeWords`. For example, use this function to reduce the words "walk", "walks", "walking" and "walk" all to their word root "walk". `normalizeWords` uses the Porter stemmer.

For more information, see `normalizeWords`.