Orange3 Text Mining Documentation

Release

Biolab

Jun 24, 2017

Contents

CHAPTER 1

Widgets

1.1 Corpus

Load a corpus of text documents, (optionally) tagged with categories.

1.1.1 Signals

Inputs:

- (None)

Outputs:

- **Corpus**

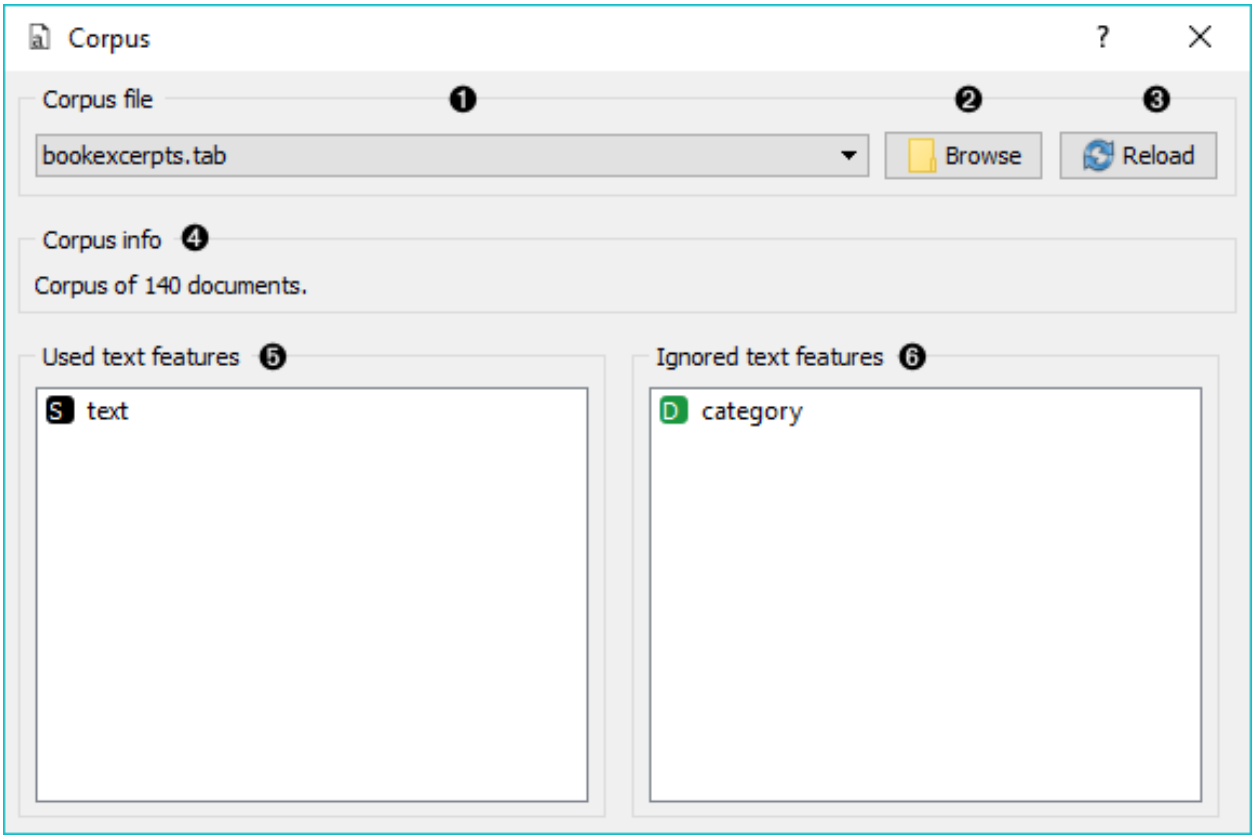
A *Corpus* instance.

1.1.2 Description

Corpus widget reads text corpora from files and sends a corpus instance to its output channel. History of the most recently opened files is maintained in the widget. The widget also includes a directory with sample corpora that come pre-installed with the add-on.

The widget reads data from Excel (**.xlsx**), comma-separated (**.csv**) and native tab-delimited (**.tab**) files.

1. Browse through previously opened data files, or load any of the sample ones.
2. Browse for a data file.



3. Reloads currently selected data file.
4. Information on the loaded data set.
5. Features that will be used in text analysis.
6. Features that won't be used in text analysis and serve as labels or class.

You can drag and drop features between the two boxes and also change the order in which they appear.

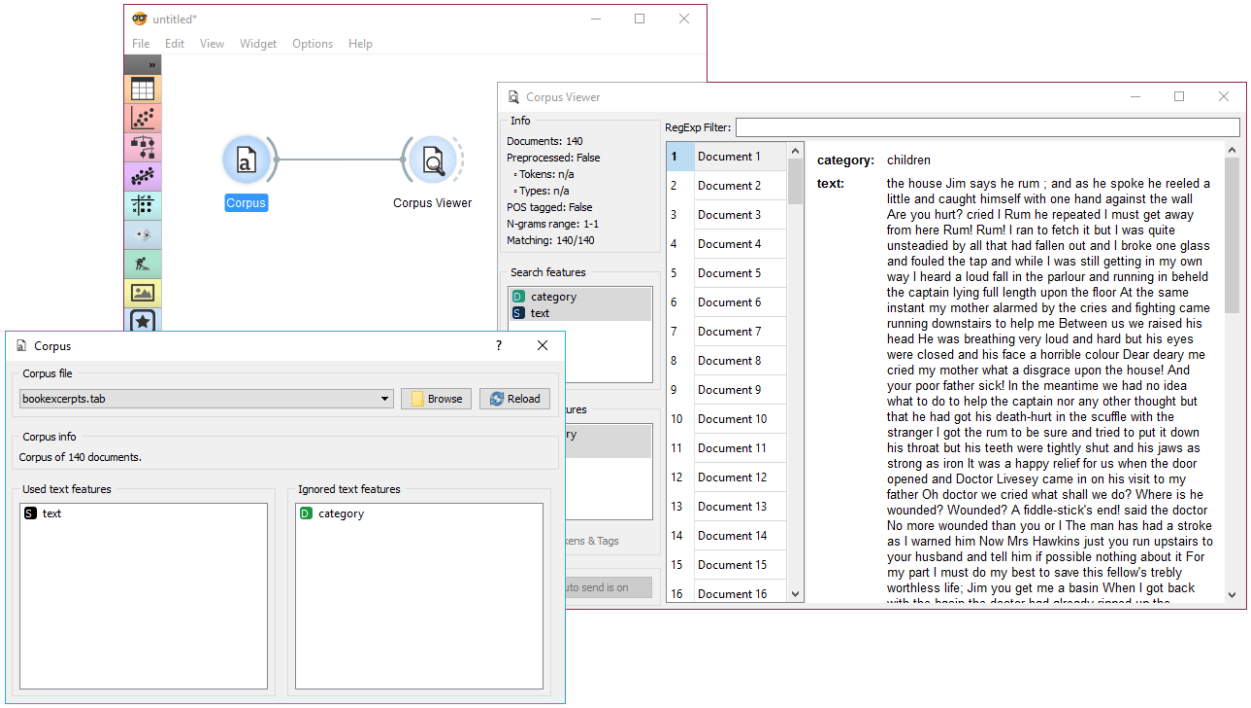
1.1.3 Example

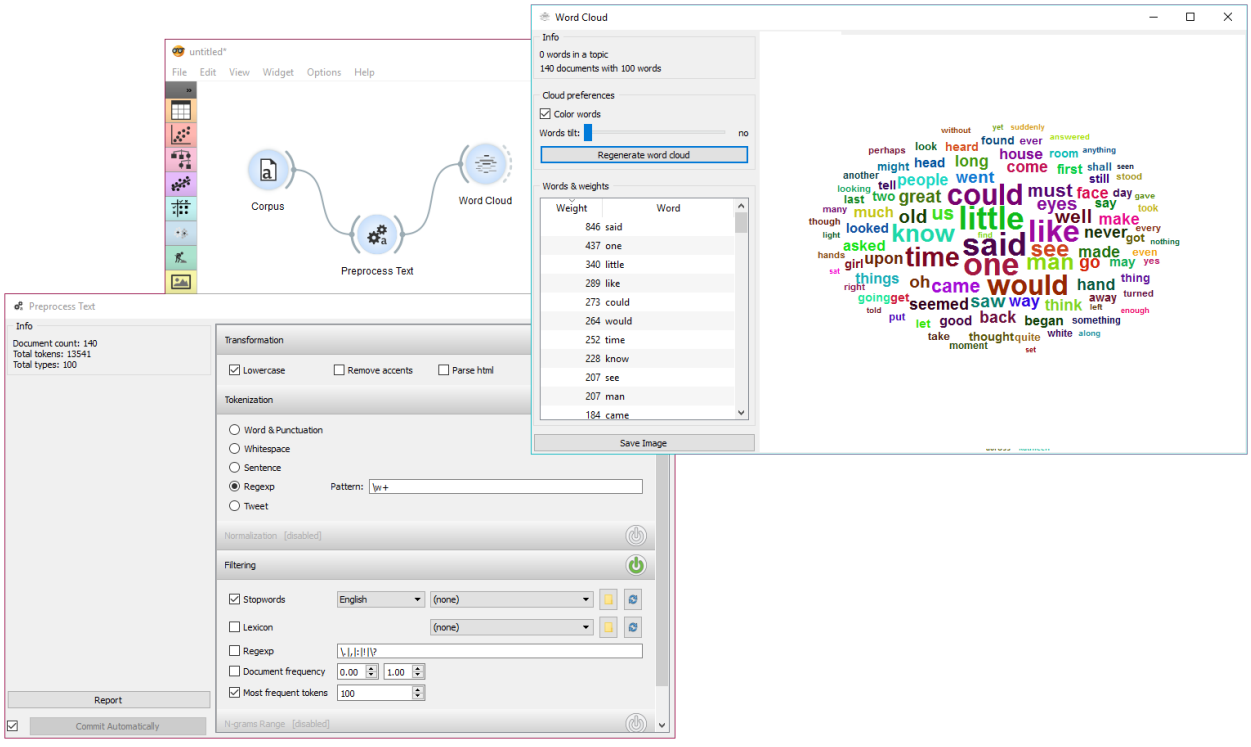
The first example shows a very simple use of **Corpus** widget. Place **Corpus** onto canvas and connect it to *Corpus Viewer*. We've used *booxexcerpts.tab* data set, which comes with the add-on, and inspected it in **Corpus Viewer**.

The second example demonstrates how to quickly visualize your corpus with *Word Cloud*. We could connect **Word Cloud** directly to **Corpus**, but instead we decided to apply some preprocessing with *Preprocess Text*. We are again working with *bookexcerpts.tab*. We've put all text to lowercase, tokenized (split) the text to words only, filtered out English stopwords and selected a 100 most frequent tokens.

1.2 NY Times

Loads data from the New York Times' Article Search API.


untitled*
File Edit View Widget Options Help
Corpus
Corpus Viewer
Corpus Viewer
Info
Documents: 140
Preprocessed: False
Tokens: n/a
Types: n/a
POS tagged: False
N-grams range: 1-1
Matching: 140/140
Search features
category
text
RegExp Filter:
Document 1
Document 2
Document 3
Document 4
Document 5
Document 6
Document 7
Document 8
Document 9
Document 10
Document 11
Document 12
Document 13
Document 14
Document 15
Document 16
category: children
text: the house Jim says he rum ; and as he spoke he reeled a little and caught himself with one hand against the wall Are you hurt? cried I Rum he repeated I must get away from here Rum! Rum! I ran to fetch it but I was quite unsteaded by all that had fallen out and I broke one glass and fouled the tap and while I was still getting in my own way I heard a loud fall in the parlour and running in beheld the captain lying full length upon the floor At the same instant my mother alarmed by the cries and fighting came running downstairs to help me Between us we raised his head He was breathing very loud and hard but his eyes were closed and his face a horrible colour Dear deary me cried my mother what a disgrace upon the house! And your poor father sick! In the meantime we had no idea what to do to help the captain nor any other thought but that he had got his death-hurt in the scuffle with the stranger I got the rum to be sure and tried to put it down his throat but his teeth were tightly shut and his jaws as strong as iron It was a happy relief for us when the door opened and Doctor Livesey came in on his visit to my father Oh doctor we cried what shall we do? Where is he wounded? Wounded? A fiddle-stick's end! said the doctor No more wounded than you or I The man has had a stroke as I warned him Now Mrs Hawkins just you run upstairs to your husband and tell him if possible nothing about it For my part I must do my best to save this fellow's trebly worthless life; Jim you get me a basin When I got back
Corpus
Corpus file
bookexcerpts.tab
Browse
Reload
Corpus info
Corpus of 140 documents.
Used text features
text
Ignored text features
category


untitled*
File Edit View Widget Options Help
Corpus
Word Cloud
Preprocess Text
Word Cloud
Info
0 words in a topic
140 documents with 100 words
Cloud preferences
Color words
Words tilt:
Regenerate word cloud
Words & weights
Weight Word
846 said
437 one
340 little
289 like
273 could
264 would
252 time
228 know
207 see
207 man
184 came
Save Image
Preprocess Text
Info
Document count: 140
Total tokens: 13541
Total types: 100
Transformation
Lowercase
Remove accents
Parse html
Tokenization
Word & Punctuation
Whitespace
Sentence
Regexp
Pattern: \w+
Tweet
Normalization [disabled]
Filtering
Stopwords
English
(none)
Lexicon
Regexp
Document frequency
Most frequent tokens
Report
Commit Automatically
N-grams Range [disabled]

1.2.1 Signals

Inputs:

- (None)

Outputs:

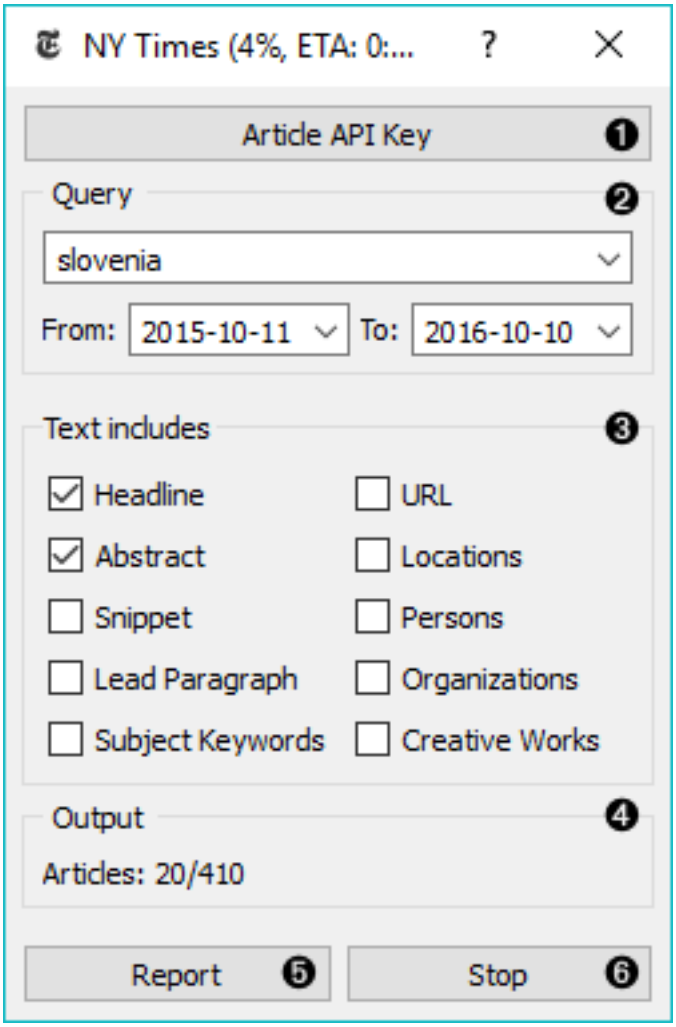
- **Corpus**

  A *Corpus* instance.

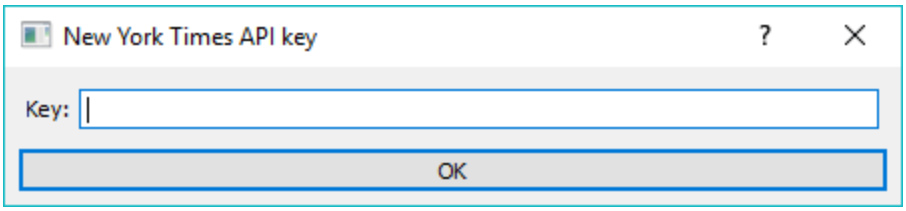
1.2.2 Description

NYTimes widget loads data from New York Times' Article Search API. You can query NYTimes articles from September 18, 1851 to today, but the API limit is set to allow retrieving only a 1000 documents per query. Define which features to use for text mining, *Headline* and *Abstract* being selected by default.

To use the widget, you must enter your own API key.



1. To begin your query, insert NY Times' Article Search API key. The key is securely saved in your system keyring service (like Credential Vault, Keychain, KWallet, etc.) and won't be deleted when clearing widget settings.



2. **Set query parameters:**
    - *Query*
    - Query time frame. The widget allows querying articles from September 18, 1851 onwards. Default is set to 1 year back from the current date.
3. Define which features to include as text features.
4. Information on the output.
5. Produce report.
6. Run or stop the query.

1.2.3 Example

NYTimes is a data retrieving widget, similar to *Twitter* and *Wikipedia*. As it can retrieve geolocations, that is geographical locations the article mentions, it is great in combination with *GeoMap* widget.

First, let's query **NYTimes** for all articles on Slovenia. We can retrieve the articles found and view the results in *Corpus Viewer*. The widget displays all the retrieved features, but includes on selected features as text mining features.

Now, let's inspect the distribution of geolocations from the articles mentioning Slovenia. We can do this with *GeoMap*. Unsuprisingly, Croatia and Hungary appear the most often in articles on Slovenia (discounting Slovenia itself), with the rest of Europe being mentioned very often as well.

1.3 The Guardian

Fetching data from The Guardian Open Platform.

1.3.1 Signals

Inputs:

- (None)

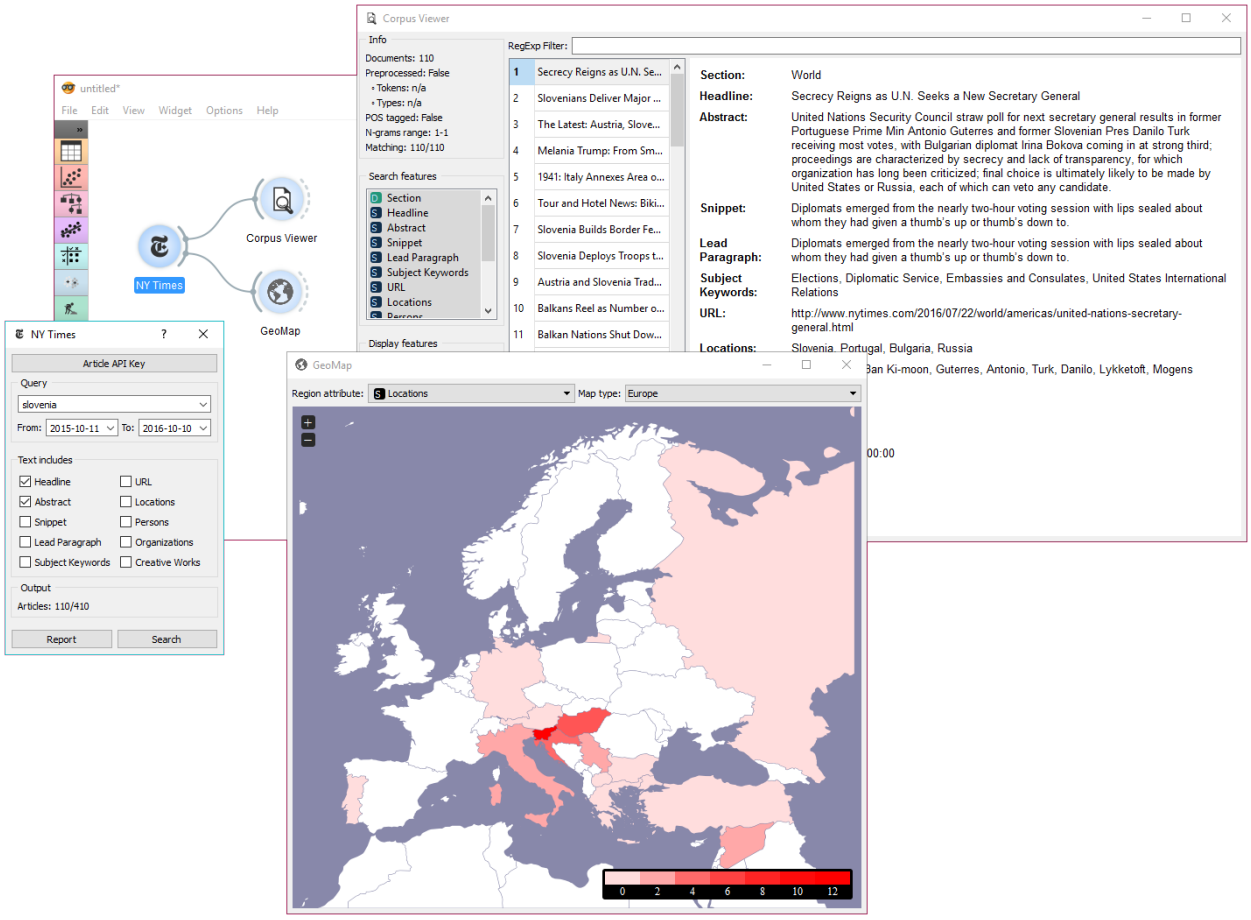
Outputs:

- **Corpus**

  A *Corpus* instance.

1.3.2 Description

No description for this widget yet.


Corpus Viewer
Info
Documents: 110
Preprocessed: False
Tokens: n/a
Types: n/a
POS tagged: False
N-grams range: 1-1
Matching: 110/110
RegExp Filter:
Search features
Section
Headline
Abstract
Snippet
Lead Paragraph
Subject Keywords
URL
Locations
Display features
Secrecy Reigns as U.N. Se...
Slovenians Deliver Major ...
The Latest: Austria, Slove...
Melania Trump: From Sm...
1941: Italy Annexes Area o...
Tour and Hotel News: Biki...
Slovenia Builds Border Fe...
Slovenia Deploys Troops t...
Austria and Slovenia Trad...
Balkans Reel as Number o...
Balkan Nations Shut Dow...
Section: World
Headline: Secrecy Reigns as U.N. Seeks a New Secretary General
Abstract: United Nations Security Council straw poll for next secretary general results in former Portuguese Prime Min Antonio Guterres and former Slovenian Pres Danilo Turk receiving most votes, with Bulgarian diplomat Irina Bokova coming in at strong third; proceedings are characterized by secrecy and lack of transparency, for which organization has long been criticized; final choice is ultimately likely to be made by United States or Russia, each of which can veto any candidate.
Snippet: Diplomats emerged from the nearly two-hour voting session with lips sealed about whom they had given a thumb's up or thumb's down to.
Lead Paragraph: Diplomats emerged from the nearly two-hour voting session with lips sealed about whom they had given a thumb's up or thumb's down to.
Subject Keywords: Elections, Diplomatic Service, Embassies and Consulates, United States International Relations
URL: http://www.nytimes.com/2016/07/22/world/americas/united-nations-secretary-general.html
Locations: Slovenia, Portugal, Bulgaria, Russia
untitled*
File Edit View Widget Options Help
Corpus Viewer
NY Times
GeoMap
NY Times
Article API Key
Query
slovenia
From: 2015-10-11 To: 2016-10-10
Text includes
Headline
Abstract
Snippet
Lead Paragraph
Subject Keywords
URL
Locations
Persons
Organizations
Creative Works
Output
Articles: 110/410
Report
Search
GeoMap
Region attribute: Locations
Map type: Europe
0 2 4 6 8 10 12

1.3.3 Examples

No examples for this widget yet.

1.4 Twitter

Fetching data from The Twitter Search API.

1.4.1 Signals

Inputs:

- (None)

Outputs:

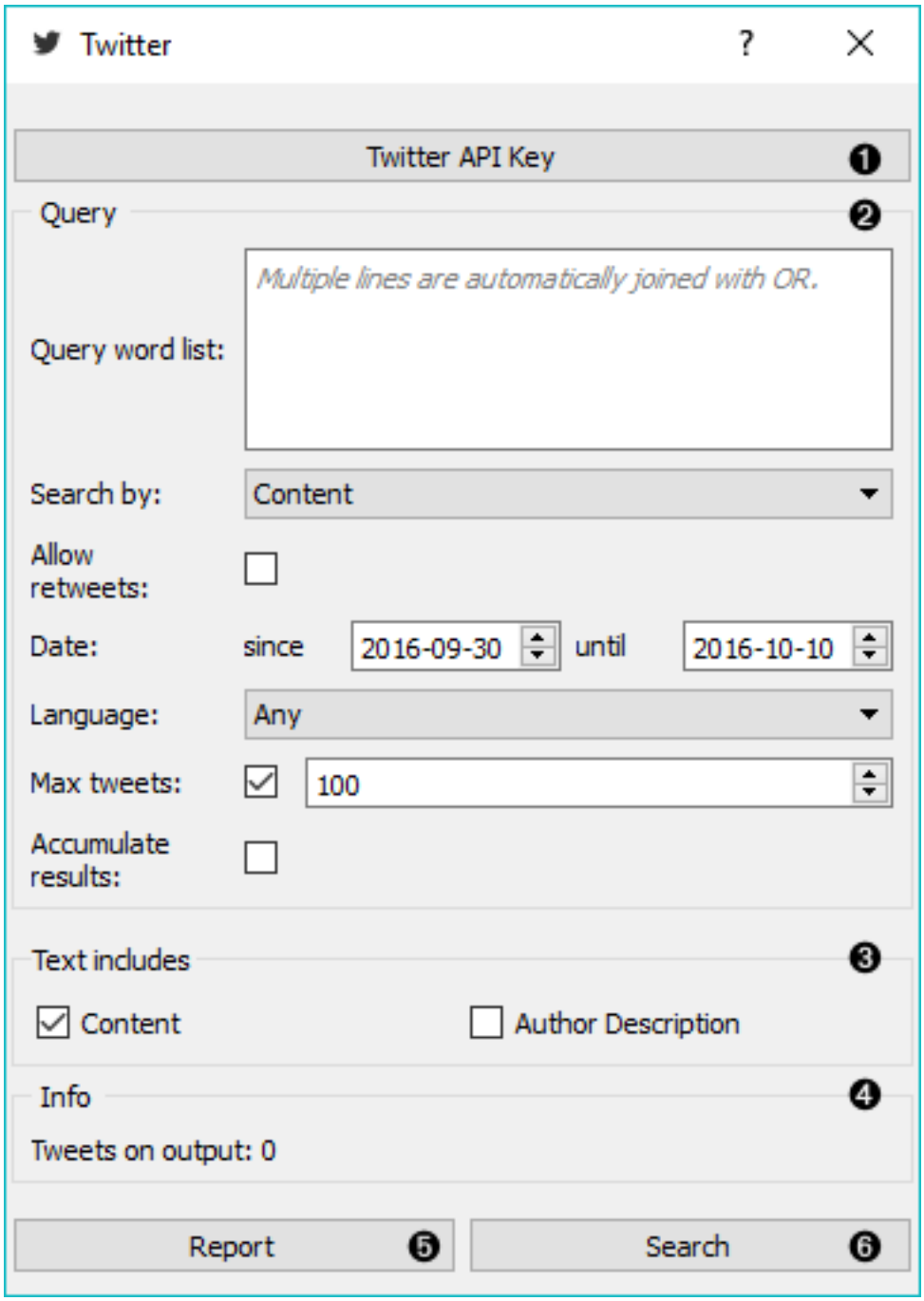
- **Corpus**

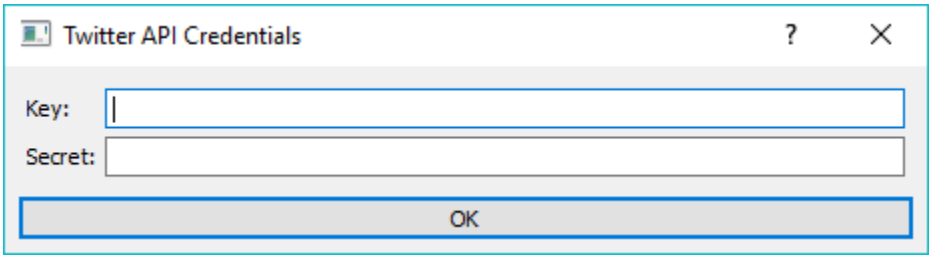
  A *Corpus* instance.

1.4.2 Description

Twitter widget enables querying tweets through Twitter API. You can query by content, author or both and accumulate results should you wish to create a larger data set. The widget only supports REST API and allows queries for up to two weeks back.

1. To begin your queries, insert Twitter key and secret. They are securely saved in your system keyring service (like Credential Vault, Keychain, KWallet, etc.) and won't be deleted when clearing widget settings. You must first create a Twitter app to get API keys.
2. **Set query parameters:**
   - *Query word list*: list desired queries, one per line. Queries are automatically joined by OR.
   - *Search by*: specify whether you want to search by content, author or both. If searching by author, you must enter proper Twitter handle (without @) in the query list.
   - *Allow retweets*: if 'Allow retweets' is checked, retweeted tweets will also appear on the output. This might duplicate some results.
   - *Date*: set the query time frame. Twitter only allows retrieving tweets from up to two weeks back.
   - *Language*: set the language of retrieved tweets. Any will retrieve tweets in any language.
   - *Max tweets*: set the top limit of retrieved tweets. If box is not ticked, no upper bound will be set - widget will retrieve all available tweets.
   - *Accumulate results*: if 'Accumulate results' is ticked, widget will append new queries to the previous ones. Enter new queries, run *Search* and new results will be appended to the previous ones.
3. Define which features to include as text features.
4. Information on the number of tweets on the output.

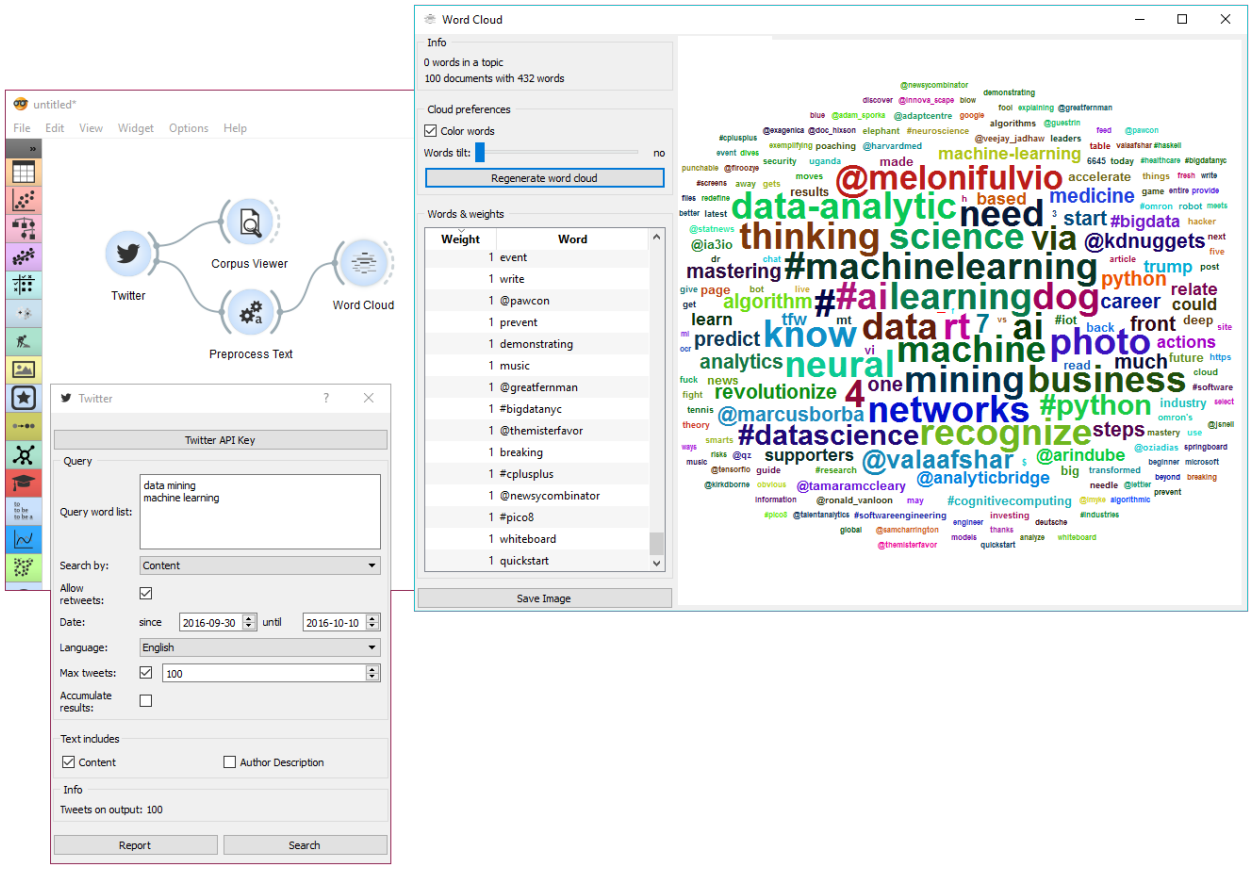

Twitter
Twitter API Key
1
Query
2
Multiple lines are automatically joined with OR.
Query word list:
Search by:
Content
Allow retweets:
Date:
since
2016-09-30
until
2016-10-10
Language:
Any
Max tweets:
100
Accumulate results:
Text includes
3
Content
Author Description
Info
4
Tweets on output: 0
Report
5
Search
6


Twitter API Credentials
Key:
Secret:
OK

5. Produce report.
6. Run query.

1.4.3 Examples

First, let's try a simple query. We will search for tweets containing either 'data mining' or 'machine learning' in the content and allow retweets. We will further limit our search to only a 100 tweets in English.



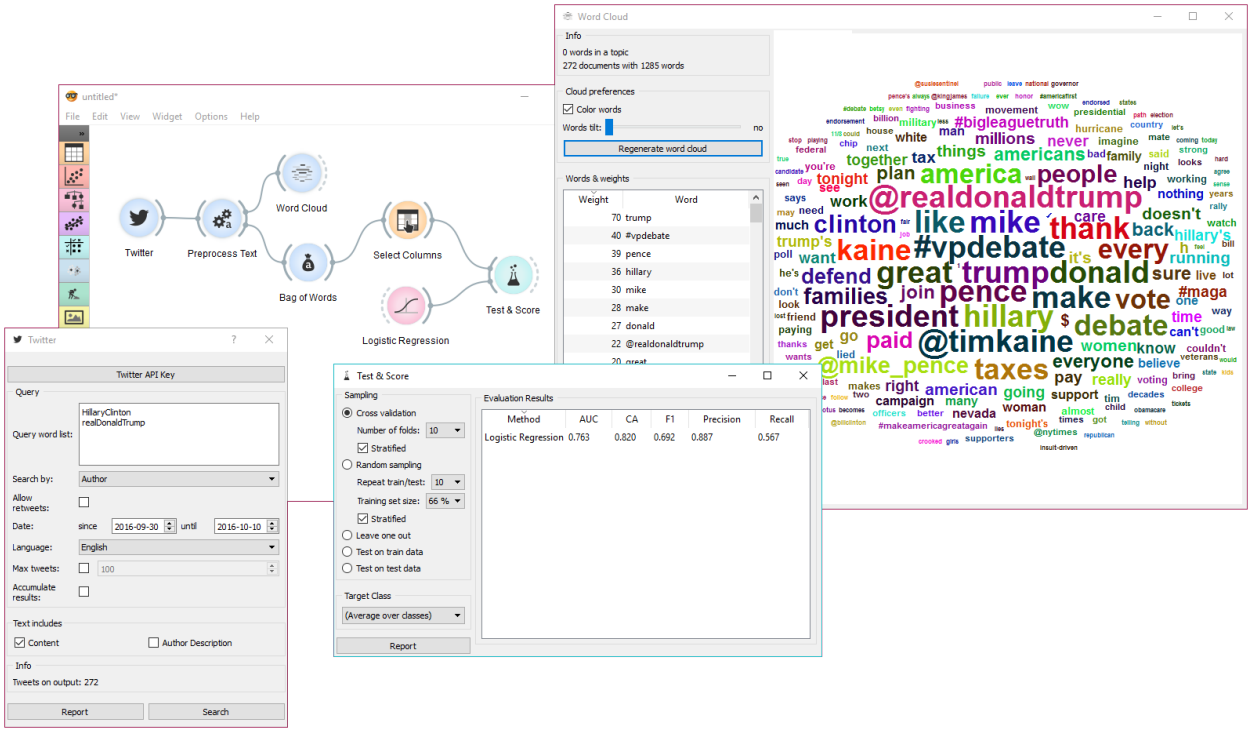
First, we're checking the output in *Corpus Viewer* to get the initial idea about our results. Then we're preprocessing the tweets with lowercase, url removal, tweet tokenizer and removal of stopword and punctuation. The best way to see the results is with *Word Cloud*. This will display the most popular words in field of data mining and machine learning in the past two weeks.

Our next example is a bit more complex. We're querying tweets from Hillary Clinton and Donald Trump from the presidential campaign 2016.

Then we've used *Preprocess Text* to get suitable tokens on our output. We've connected **Preprocess Text** to *Bag of Words* in order to create a table with words as features and their counts as values. A quick check in **Word Cloud** gives us an idea about the results.

Now we would like to predict the author of the tweet. With **Select Columns** we're setting 'Author' as our target variable. Then we connect **Select Columns** to **Test & Score**. We'll be using **Logistic Regression** as our learner, which we also connect to **Test & Score**.

We can observe the results of our author predictions directly in the widget. AUC score is quite ok. Seems like we can to some extent predict who is the author of the tweet based on the tweet content.



1.5 Wikipedia

Fetching data from MediaWiki RESTful web service API.

1.5.1 Signals

Inputs:

- (None)

Outputs:

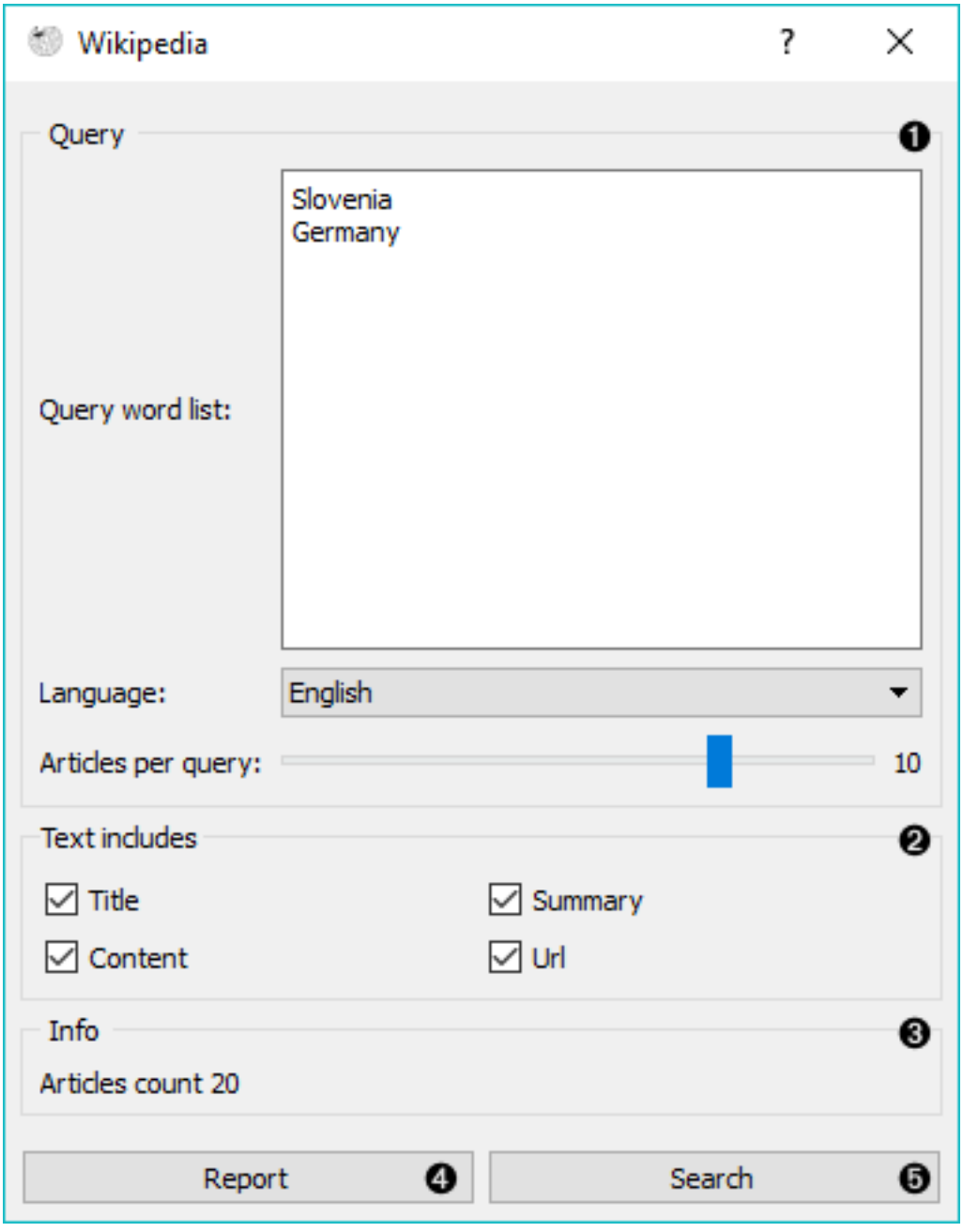
- **Corpus**

  A *Corpus* instance.

1.5.2 Description

Wikipedia widget is used to retrieve texts from Wikipedia API and it is useful mostly for teaching and demonstration.

1. **Query parameters:**
   - Query word list, where each query is listed in a new line.
   - Language of the query. English is set by default.

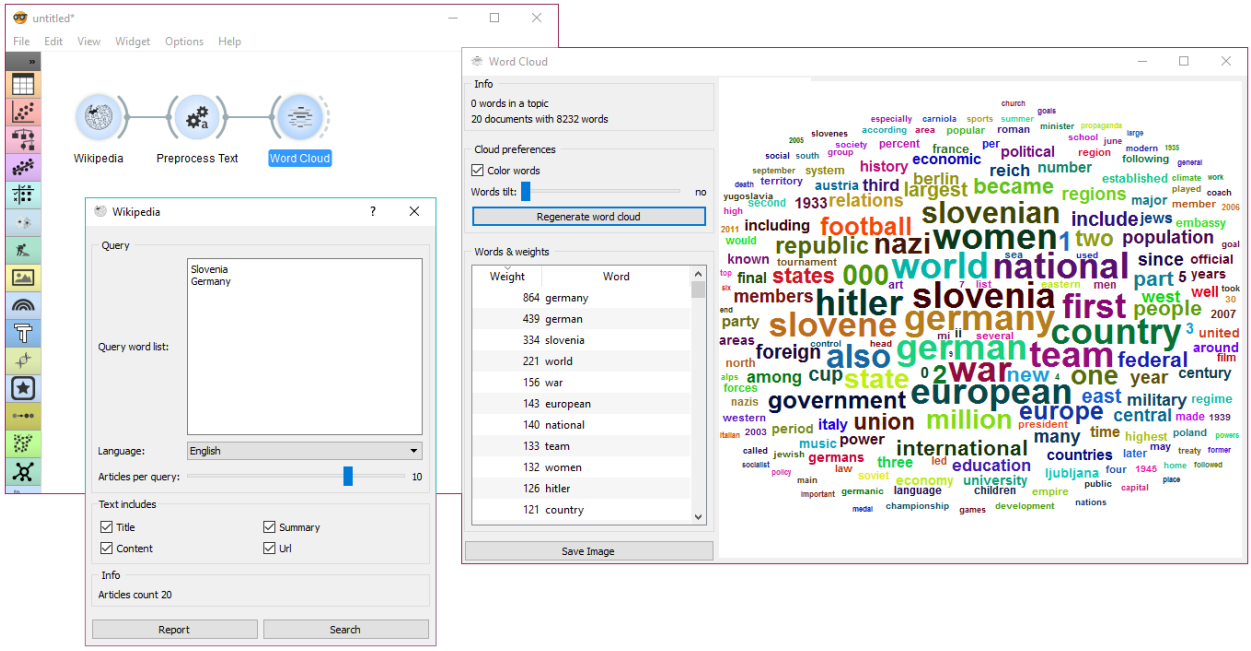

Wikipedia
?
Query
1
Slovenia
Germany
Query word list:
Language:
English
Articles per query:
10
Text includes
2
Title
Summary
Content
Url
Info
3
Articles count 20
Report
4
Search
5

- Number of articles to retrieve per query (range 1-25). Please note that querying is done recursively and that disambiguations are also retrieved, sometimes resulting in a larger number of queries than set on the slider.

2. Select which features to include as text features.
3. Information on the output.
4. Produce a report.
5. Run query.

1.5.3 Example

This is a simple example, where we use **Wikipedia** and retrieve the articles on 'Slovenia' and 'Germany'. Then we simply apply default preprocessing with *Preprocess Text* and observe the most frequent words in those articles with *Word Cloud*.



Wikipedia works just like any other corpus widget (*NY Times*, *Twitter*) and can be used accordingly.

1.6 Pubmed

Fetch data from PubMed journals.

1.6.1 Signals

Inputs:

- (None)

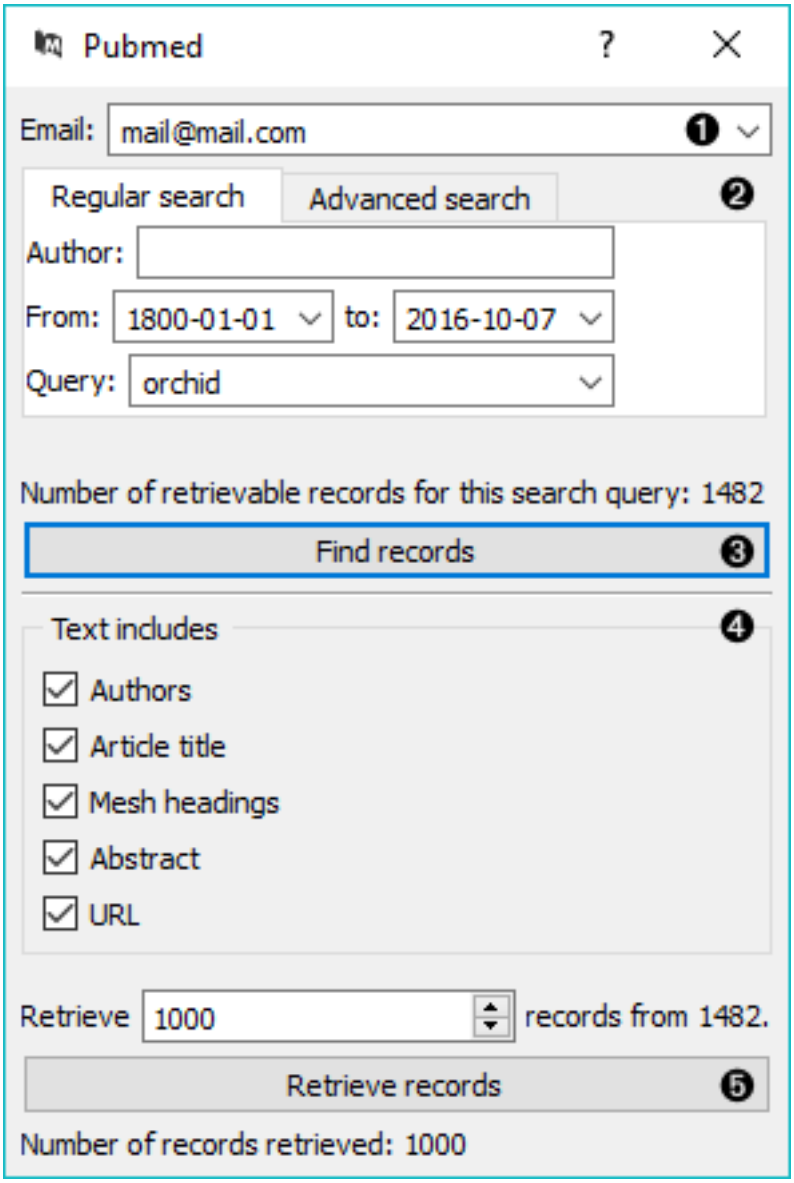
Outputs:

- **Corpus**

  A *Corpus* instance.

1.6.2 Description

PubMed comprises more than 26 million citations for biomedical literature from MEDLINE, life science journals, and online books. The widget allows you to query and retrieve these entries. You can use regular search or construct advanced queries.



1. Enter a valid e-mail to retrieve queries.
2. ***Regular search*:**

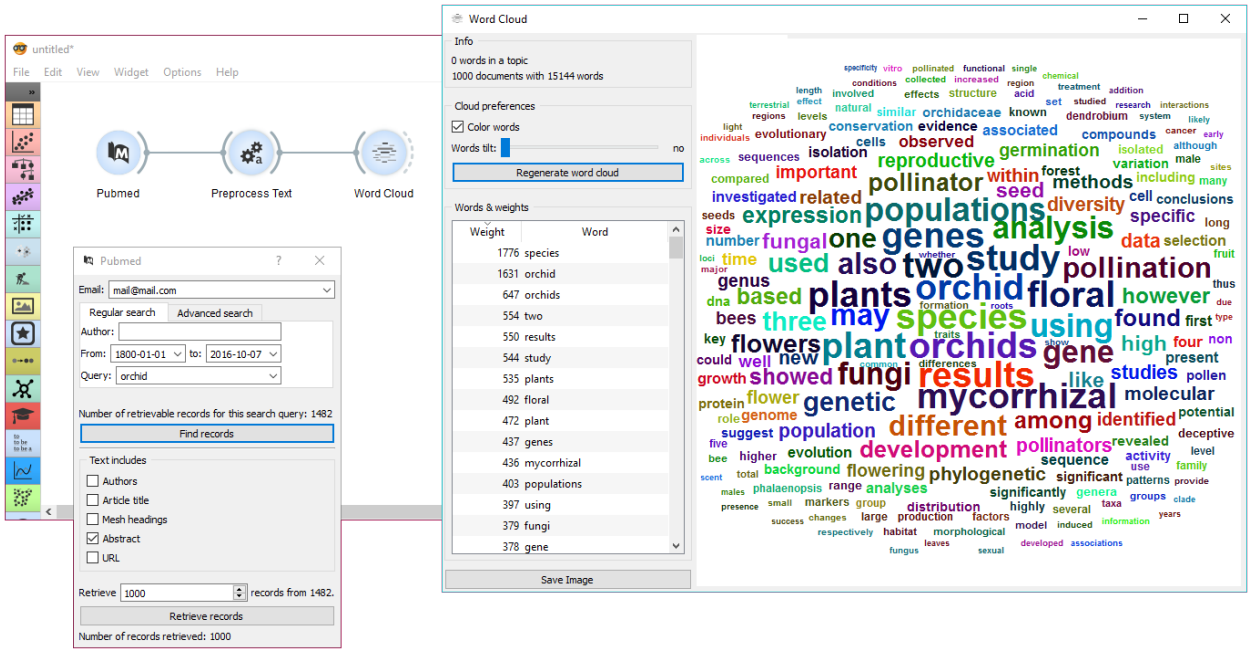
- *Author*: queries entries from a specific author. Leave empty to query by all authors.
- *From*: define the time frame of publication.
- *Query*: enter the query.

Advanced search: enables you to construct complex queries. See PubMed's website to learn how to construct such queries. You can also copy-paste constructed queries from the website.

3. *Find records* finds available data from PubMed matching the query. Number of records found will be displayed above the button.
4. Define the output. All checked features will be on the output of the widget.
5. Set the number of record you wish to retrieve. Press *Retrieve records* to get results of your query on the output. Below the button is an information on the number of records on the output.

1.6.3 Example

PubMed can be used just like any other data widget. In this example we've queried the database for records on orchids. We retrieved 1000 records and kept only 'abstract' in our meta features to limit the construction of tokens only to this feature.



We used *Preprocess Text* to remove stopword and words shorter than 3 characters (regexp `\b\w{1,2}\b`). This will perhaps get rid of some important words denoting chemicals, so we need to be careful with what we filter out. For the sake of quick inspection we only retained longer words, which are displayed by frequency in *Word Cloud*.

1.7 Corpus Viewer

Displays corpus content.

1.7.1 Signals

Inputs:

- **Data**

  Data instance.

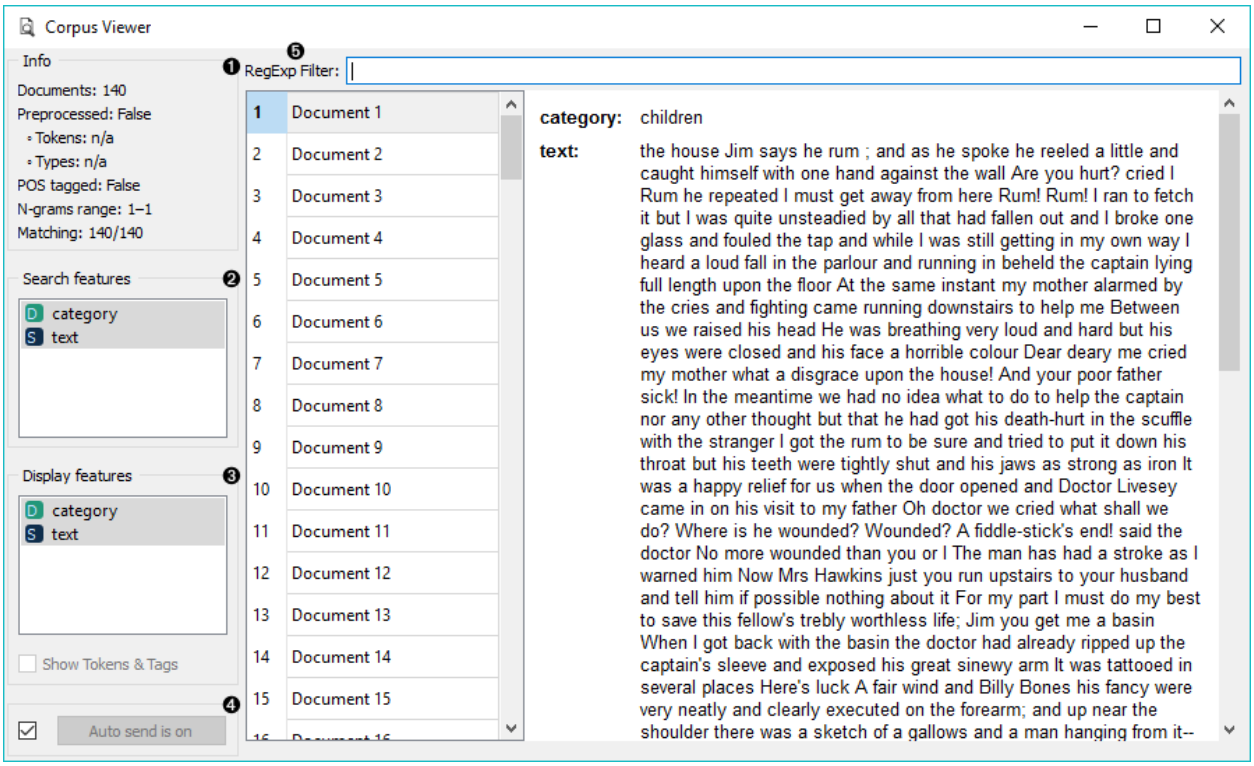
Outputs:

- **Corpus**

  A *Corpus* instance.

1.7.2 Description

Corpus Viewer is primarily meant for viewing text files (instances of *Corpus*), but it can also display other data files from **File** widget. **Corpus Viewer** will always output an instance of corpus. If *RegExp* filtering is used, the widget will output only matching documents.



1. ***Information*:**

   - *Documents*: number of documents on the input
   - *Preprocessed*: if preprocessor is used, the result is True, else False. Reports also on the number of tokens and types (unique tokens).

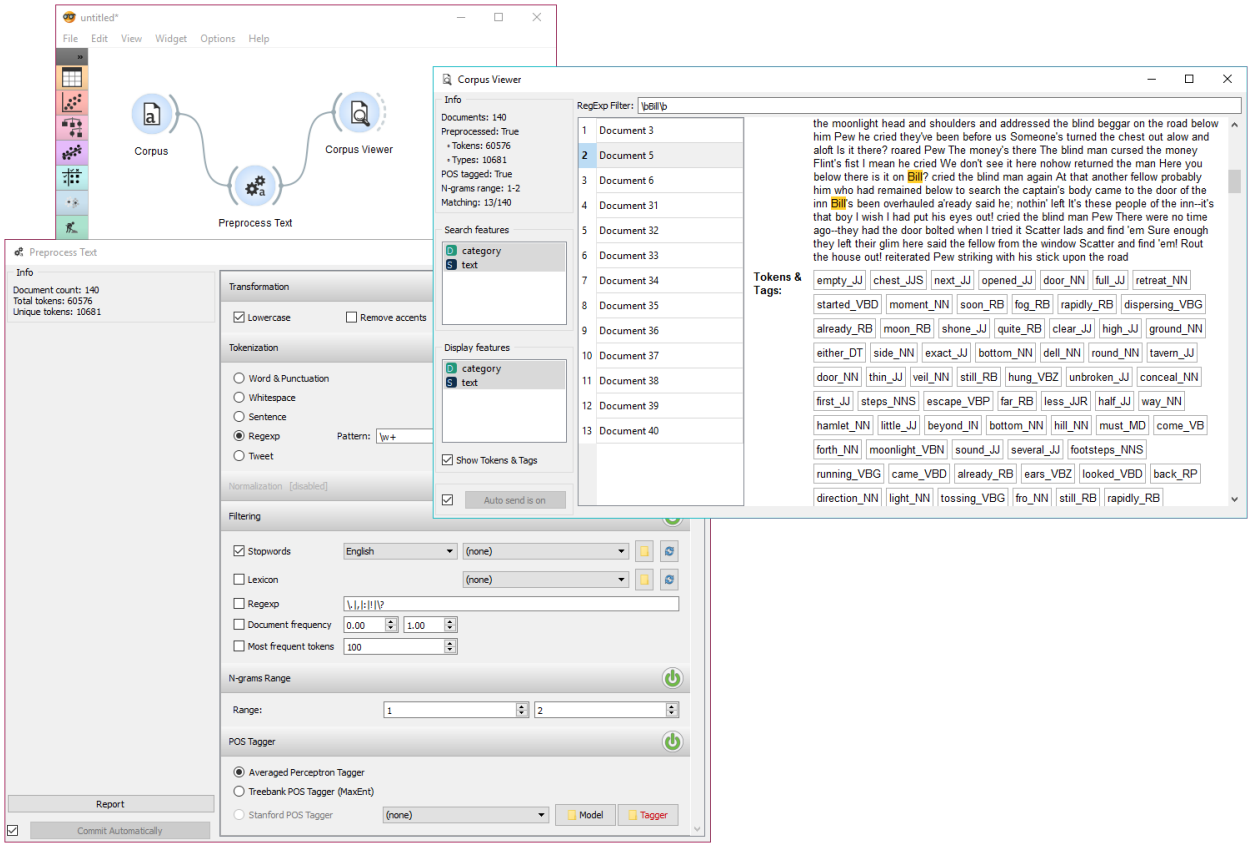
- *POS tagged*: if POS tags are on the input, the result is True, else False.
- *N-grams range*: if N-grams are set in *Preprocess Text*, results are reported, default is 1-1 (one-grams).
- *Matching*: number of documents matching the *RegExp Filter*. All documents are output by default.

2. *RegExp Filter*: Python regular expression for filtering documents. By default no documents are filtered (entire corpus is on the output).
3. *Search Features*: features by which the RegExp Filter is filtering. Use Ctrl (Cmd) to select multiple features.
4. *Display Features*: features that are displayed in the viewer. Use Ctrl (Cmd) to select multiple features.
5. *Show Tokens & Tags*: if tokens and POS tag are present on the input, you can check this box to display them.
6. If *Auto commit is on*, changes are communicated automatically. Alternatively press *Commit*.

1.7.3 Example

Corpus Viewer can be used for displaying all or some documents in corpus. In this example, we will first load *bookexcerpts.tab*, that already comes with the add-on, into *Corpus* widget. Then we will preprocess the text into words, filter out the stopwords, create bi-grams and add POS tags (more on preprocessing in *Preprocess Text*). Now we want to see the results of preprocessing. In *Corpus Viewer* we can see, how many unique tokens we got and what they are (tick *Show Tokens & Tags*). Since we used also POS tagger to show part-of-speech labels, they will be displayed alongside tokens underneath the text.

Now we will filter out just the documents talking about a character Bill. We use regular expression *\bBill\b* to find the documents containing only the word Bill. You can output matching or non-matching documents, view them in another *Corpus Viewer* or further analyse them.



1.8 Preprocess Text

Preprocesses corpus with selected methods.

1.8.1 Signals

Inputs:

- **Corpus**

  Corpus instance.

Outputs:

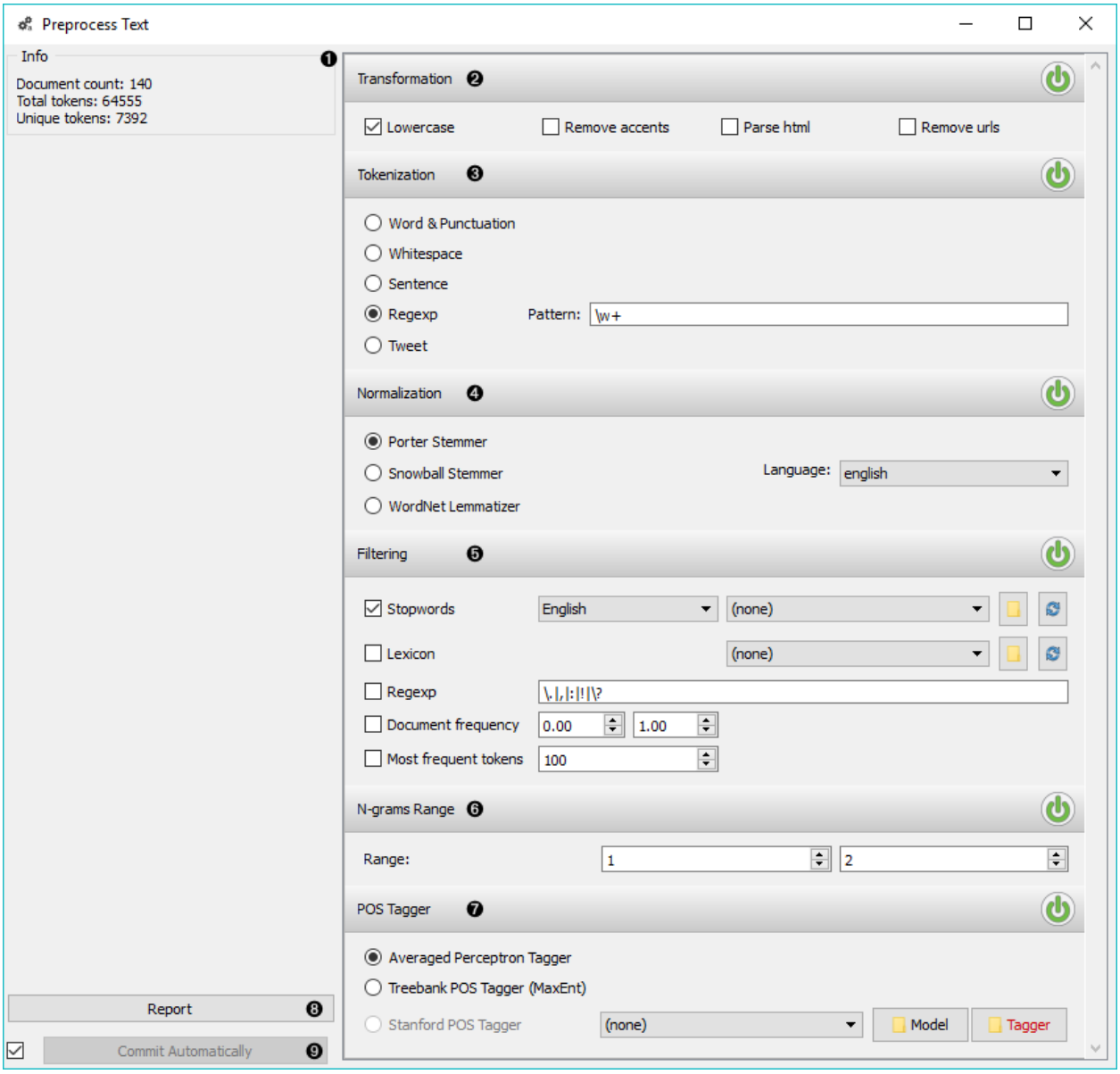
- **Corpus**

  Preprocessed corpus.

1.8.2 Description

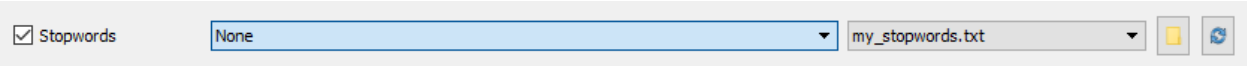
Preprocess Text splits your text into smaller units (tokens), filters them, runs normalization (stemming, lemmatization), creates n-grams and tags tokens with part-of-speech labels. Steps in the analysis are applied sequentially and can be turned on or off.

1. **Information on preprocessed data**. *Document count* reports on the number of documents on the input. *Total tokens* counts all the tokens in corpus. *Unique tokens* excludes duplicate tokens and reports only on unique tokens in the corpus.
2. **Transformation transforms input data. It applies lowercase transformation by default.**
   - *Lowercase* will turn all text to lowercase.
   - ***Remove accents* will remove all diacritics/accents in text.** naïve → naive
   - ***Parse html* will detect html tags and parse out text only.** <a href...>Some text → Some text
   - ***Remove urls* will remove urls from text.** This is a http://orange.biolab.si/ url. → This is a url.
3. **Tokenization is the method of breaking the text into smaller components (words, sentences, bigrams).**
   - ***Word & Punctuation* will split the text by words and keep punctuation symbols.** This example. → (This), (example), (.)
   - ***Whitespace* will split the text by whitespace only.** This example. → (This), (example.)
   - ***Sentence* will split the text by fullstop, retaining only full sentences.** This example. Another example. → (This example.), (Another example.)
   - Regexp will split the text by provided regex. It splits by words only by default (omits punctuation).
   - ***Tweet* will split the text by pre-trained Twitter model, which keeps hashtags, emoticons and other special symbol** This example. :-) #simple → (This), (example), (.), (:-)), (#simple)
4. **Normalization applies stemming and lemmatization to words. (I've always loved cats. → I have alway love cat.) For langu**


Preprocess Text
Info
Document count: 140
Total tokens: 64555
Unique tokens: 7392
Transformation
Lowercase
Remove accents
Parse html
Remove urls
Tokenization
Word & Punctuation
Whitespace
Sentence
Regexp
Pattern: \w+
Tweet
Normalization
Porter Stemmer
Snowball Stemmer
WordNet Lemmatizer
Language: english
Filtering
Stopwords
English
(none)
Lexicon
(none)
Regexp
\.|,|:|!|\?
Document frequency
0.00
1.00
Most frequent tokens
100
N-grams Range
Range:
1
2
POS Tagger
Averaged Perceptron Tagger
Treebank POS Tagger (MaxEnt)
Stanford POS Tagger
(none)
Model
Tagger
Report
Commit Automatically

- Porter Stemmer applies the original Porter stemmer.
- Snowball Stemmer applies an improved version of Porter stemmer (Porter2). Set the language for normalization, default is English.
- WordNet Lemmatizer applies a networks of cognitive synonyms to tokens based on a large lexical database of English.

5. **Filtering removes or keeps a selection of words.**
   - *Stopwords* removes stopwords from text (e.g. removes 'and', 'or', 'in'...). Select the language to filter by, English is set as default. You can also load your own list of stopwords provided in a simple *.txt file with one stopword per line.



   Click 'browse' icon to select the file containing stopwords. If the file was properly loaded, its name will be displayed next to pre-loaded stopwords. Change 'English' to 'None' if you wish to filter out only the provided stopwords. Click 'reload' icon to reload the list of stopwords.
   - *Lexicon* keeps only words provided in the file. Load a *.txt file with one word per line to use as lexicon. Click 'reload' icon to reload the lexicon.
   - *Regexp* removes words that match the regular expression. Default is set to remove punctuation.
   - *Document frequency* keeps tokens that appear in not less than and not more than the specified number / percentage of documents. If you provide integers as parameters, it keeps only tokens that appear in the specified number of documents. E.g. DF = (3, 5) keeps only tokens that appear in 3 or more and 5 or less documents. If you provide floats as parameters, it keeps only tokens that appear in the specified percentage of documents. E.g. DF = (0.3, 0.5) keeps only tokens that appear in 30% to 50% of documents. Default returns all tokens.
   - *Most frequent tokens* keeps only the specified number of most frequent tokens. Default is a 100 most frequent tokens.
6. **N-grams Range** creates n-grams from tokens. Numbers specify the range of n-grams. Default returns one-grams and two-grams.
7. **POS Tagger runs part-of-speech tagging on tokens.**
   - Averaged Perceptron Tagger runs POS tagging with Matthew Honnibal's averaged perceptron tagger.
   - Treebank POS Tagger (MaxEnt) runs POS tagging with a trained Penn Treebank model.
   - Stanford POS Tagger runs a log-linear part-of-speech tagger designed by Toutanova et al. Please download it from the provided website and load it in Orange.
8. Produce a report.
9. If *Commit Automatically* is on, changes are communicated automatically. Alternatively press *Commit*.

Note: **Preprocess Text** applies preprocessing steps in the order they are listed. This means it will first transform the text, then apply tokenization, POS tags, normalization, filtering and finally constructs n-grams based on given tokens. This is especially important for WordNet Lemmatizer since it requires POS tags for proper normalization.

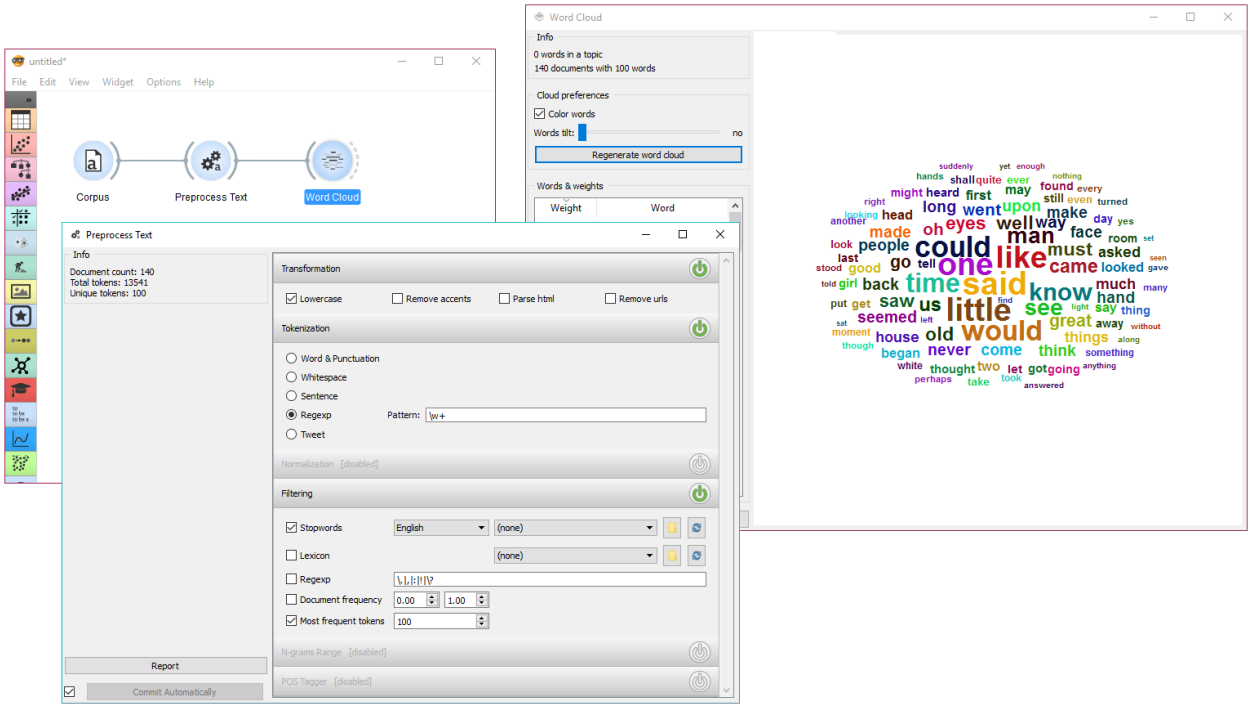
1.8.3 Useful Regular Expressions

Here are some useful regular expressions for quick filtering:

`\bword\b`	matches exact word
`\w+`	matches only words, no punctuation
`\b(B|b)\w+\b`	matches words beginning with the letter b
`\w{4,}`	matches words that are longer than 4 characters
`\b\w+(Y|y)\b`	matches words ending with the letter y

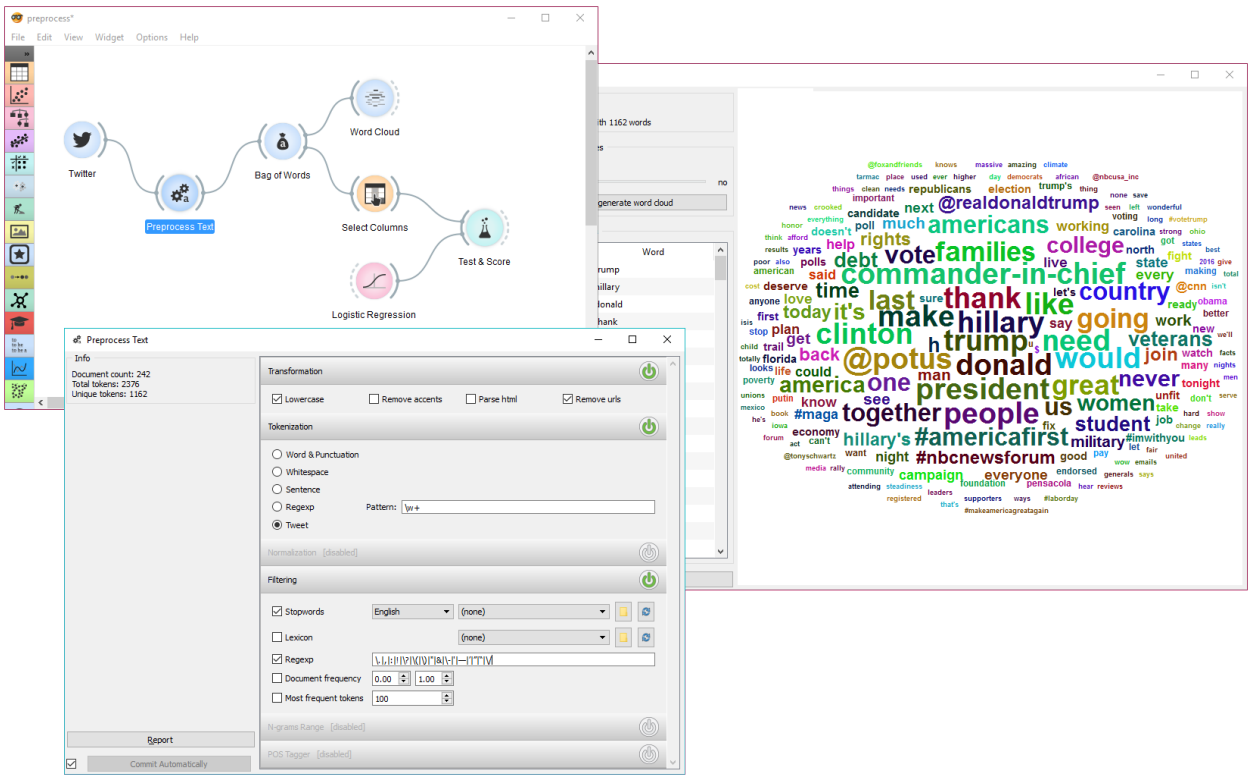
1.8.4 Examples

In the first example we will observe the effects of preprocessing on our text. We are working with *bookexcerpts.tab* that we've loaded with *Corpus* widget. We have connected **Preprocess Text** to **Corpus** and retained default preprocessing methods (lowercase, per-word tokenization and stopword removal). The only additional parameter we've added as outputting only the first 100 most frequent tokens. Then we connected **Preprocess Text** with *Word Cloud* to observe words that are the most frequent in our text. Play around with different parameters, to see how they transform the output.



The second example is slightly more complex. We first acquired our data with *Twitter* widget. We quired the internet for tweets from users @HillaryClinton and @realDonaldTrump and got their tweets from the past two weeks, 242 in total.

In **Preprocess Text** there's *Tweet* tokenization available, which retains hashtags, emojis, mentions and so on. However, this tokenizer doesn't get rid of punctuation, thus we expanded the Regexp filtering with symbols that we wanted to get rid of. We ended up with word-only tokens, which we displayed in *Word Cloud*. Then we created a schema for predicting author based on tweet content, which is explained in more details in the documentation for *Twitter* widget.



1.9 Bag of Words

Generates a bag of words from the input corpus.

1.9.1 Signals

Inputs:

- **Corpus**

  Corpus instance.

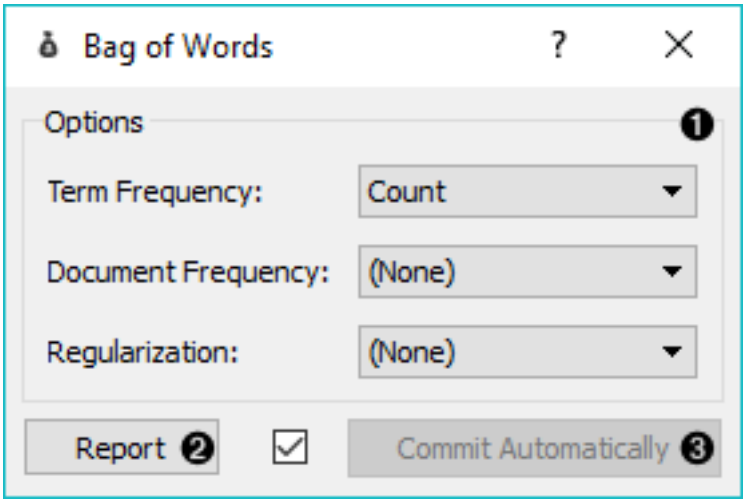
Outputs:

- **Corpus**

  Corpus with bag of words.

1.9.2 Description

Bag of Words model creates a corpus with word counts for each data instance (document). The count can be either absolute, binary (contains or does not contain) or sublinear (logarithm of the term frequency). Bag of words model is required in combination with *Word Enrichment* and could be used for predictive modelling.



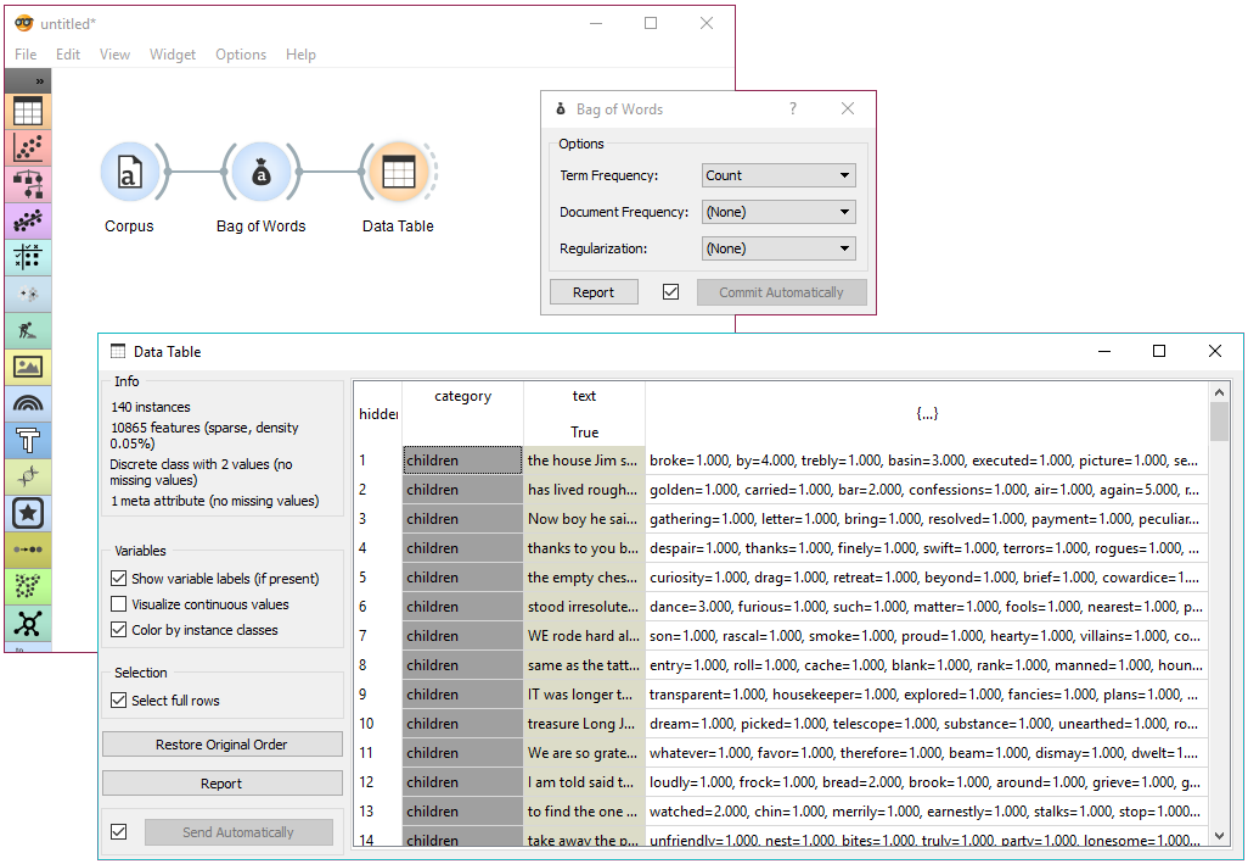
1. **Parameters for bag of words model:**
    - **Term Frequency:**
        - Count: number of occurences of a word in a document
        - Binary: word appears or does not appear in the document
        - Sublinear: logarithm of term frequency (count)
    - **Document Frequency:**
        - (None)
        - IDF: inverse document frequency
        - Smooth IDF: adds one to document frequencies to prevent zero division.
    - **Regulariation:**
        - (None)
        - L1 (Sum of elements): normalizes vector length to sum of elements
        - L2 (Euclidean): normalizes vector length to sum of squares
2. Produce a report.
3. If *Commit Automatically* is on, changes are communicated automatically. Alternatively press *Commit*.

1.9.3 Example

In the first example we will simply check how the bag of words model looks like. Load *bookexcerpts.tab* with *Corpus* widget and connect it to **Bag of Words**. Here we kept the defaults - a simple count of term frequencies. Check what the **Bag of Words** outputs with **Data Table**. The final column in white represents term frequencies for each document.

In the second example we will try to predict document category. We are still using the *bookexcerpts.tab* data set, which we sent through *Preprocess Text* with default parameters. Then we connected **Preprocess Text** to **Bag of Words** to obtain term frequencies by which we will compute the model.

Connect **Bag of Words** to **Test & Score** for predictive modelling. Connect **SVM** or any other classifier to **Test & Score** as well (both on the left side). **Test & Score** will now compute performance scores for each learner on the input. Here we got quite impressive results with SVM. Now we can check, where the model made a mistake.



Add **Confusion Matrix** to **Test & Score**. Confusion matrix displays correctly and incorrectly classified documents. *Select Misclassified* will output misclassified documents, which we can further inspect with *Corpus Viewer*.

1.10 Topic Modelling

Topic modelling with Latent Diriclet Allocation, Latent Semantic Indexing or Hierarchical Dirichlet Process.

1.10.1 Signals

Inputs:

- **Corpus**

Corpus instance.

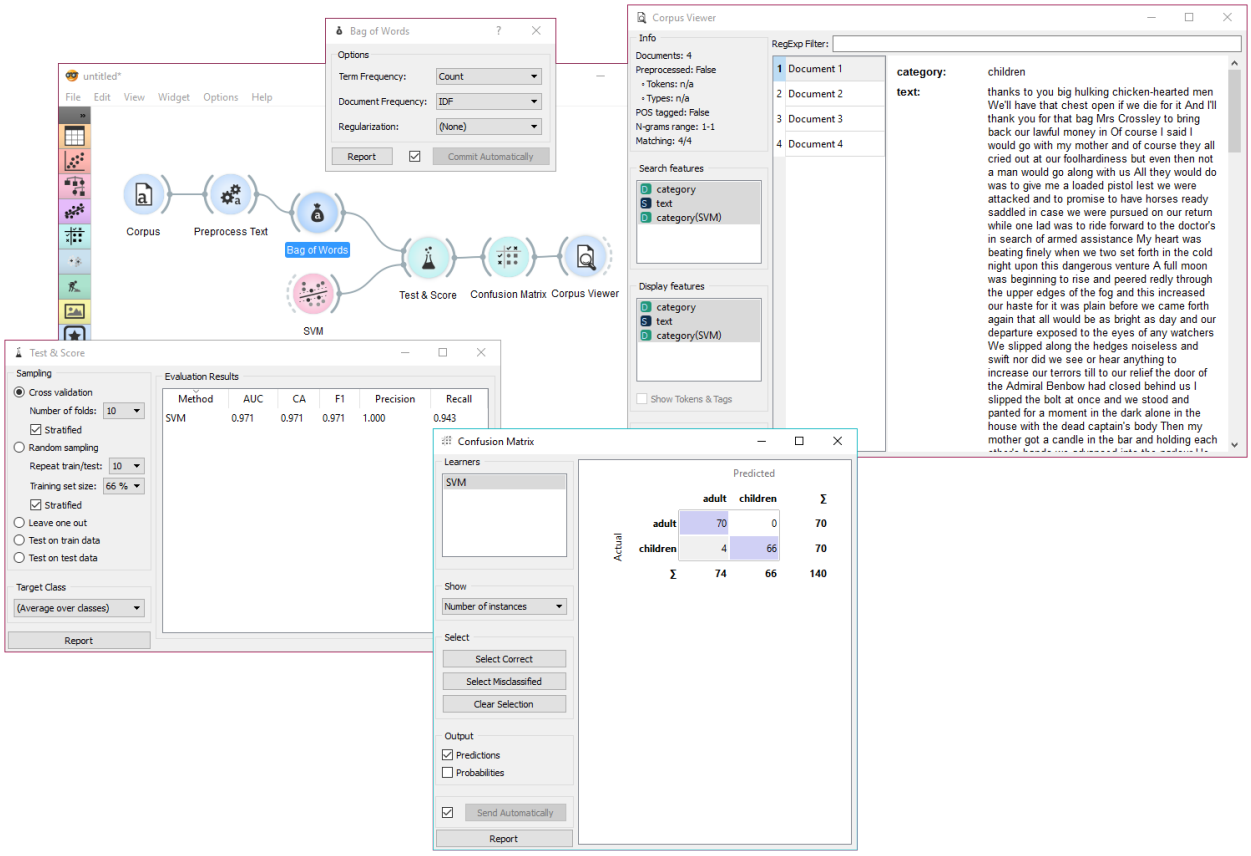
Outputs:

- **Data**

Data with topic weights appended.

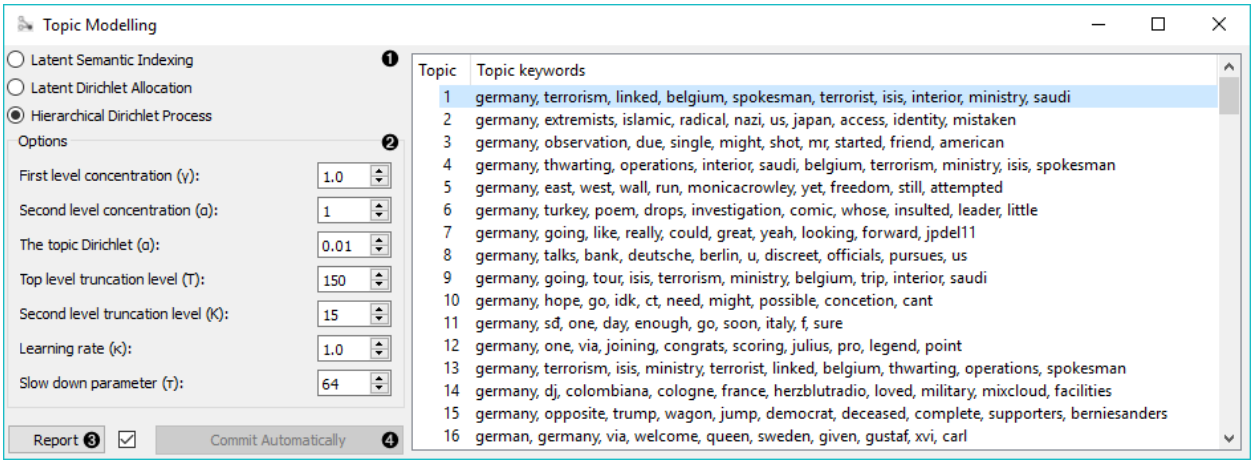
- **Topics**

Selected topics with word weights.


Bag of Words
Options
Term Frequency: Count
Document Frequency: IDF
Regularization: (None)
Report
Commit Automatically
untitled*
File Edit View Widget Options Help
Corpus
Preprocess Text
Bag of Words
SVM
Test & Score
Confusion Matrix
Corpus Viewer
Corpus Viewer
Info
Documents: 4
Preprocessed: False
Tokens: n/a
Types: n/a
POS tagged: False
N-grams range: 1-1
Matching: 4/4
RegExp Filter:
1 Document 1
2 Document 2
3 Document 3
4 Document 4
category: children
text: thanks to you big hulking chicken-hearted men We'll have that chest open if we die for it And I'll thank you for that bag Mrs Crossley to bring back our lawful money in Of course I said I would go with my mother and of course they all cried out at our foolhardiness but even then not a man would go along with us All they would do was to give me a loaded pistol lest we were attacked and to promise to have horses ready saddled in case we were pursued on our return while one lad was to ride forward to the doctor's in search of armed assistance My heart was beating finely when we two set forth in the cold night upon this dangerous venture A full moon was beginning to rise and peered redly through the upper edges of the fog and this increased our haste for it was plain before we came forth again that all would be as bright as day and our departure exposed to the eyes of any watchers We slipped along the hedges noiseless and swift nor did we see or hear anything to increase our terrors till to our relief the door of the Admiral Benbow had closed behind us I slipped the bolt at once and we stood and panted for a moment in the dark alone in the house with the dead captain's body Then my mother got a candle in the bar and holding each
Search features
category
text
category(SVM)
Display features
category
text
category(SVM)
Show Tokens & Tags
Test & Score
Sampling
Cross validation
Number of folds: 10
Stratified
Random sampling
Repeat train/test: 10
Training set size: 66 %
Stratified
Leave one out
Test on train data
Test on test data
Target Class
(Average over classes)
Report
Evaluation Results
Method AUC CA F1 Precision Recall
SVM 0.971 0.971 0.971 1.000 0.943
Confusion Matrix
Learners
SVM
Show
Number of instances
Select
Select Correct
Select Misclassified
Clear Selection
Output
Predictions
Probabilities
Send Automatically
Report
Predicted
adult children Σ
adult 70 0 70
children 4 66 70
Σ 74 66 140
Actual

1.10.2 Description

Topic Modelling discovers abstract topics in a corpus based on clusters of words found in each document and their respective frequency. A document typically contains multiple topics in different proportions, thus the widget also reports on the topic weight per document.

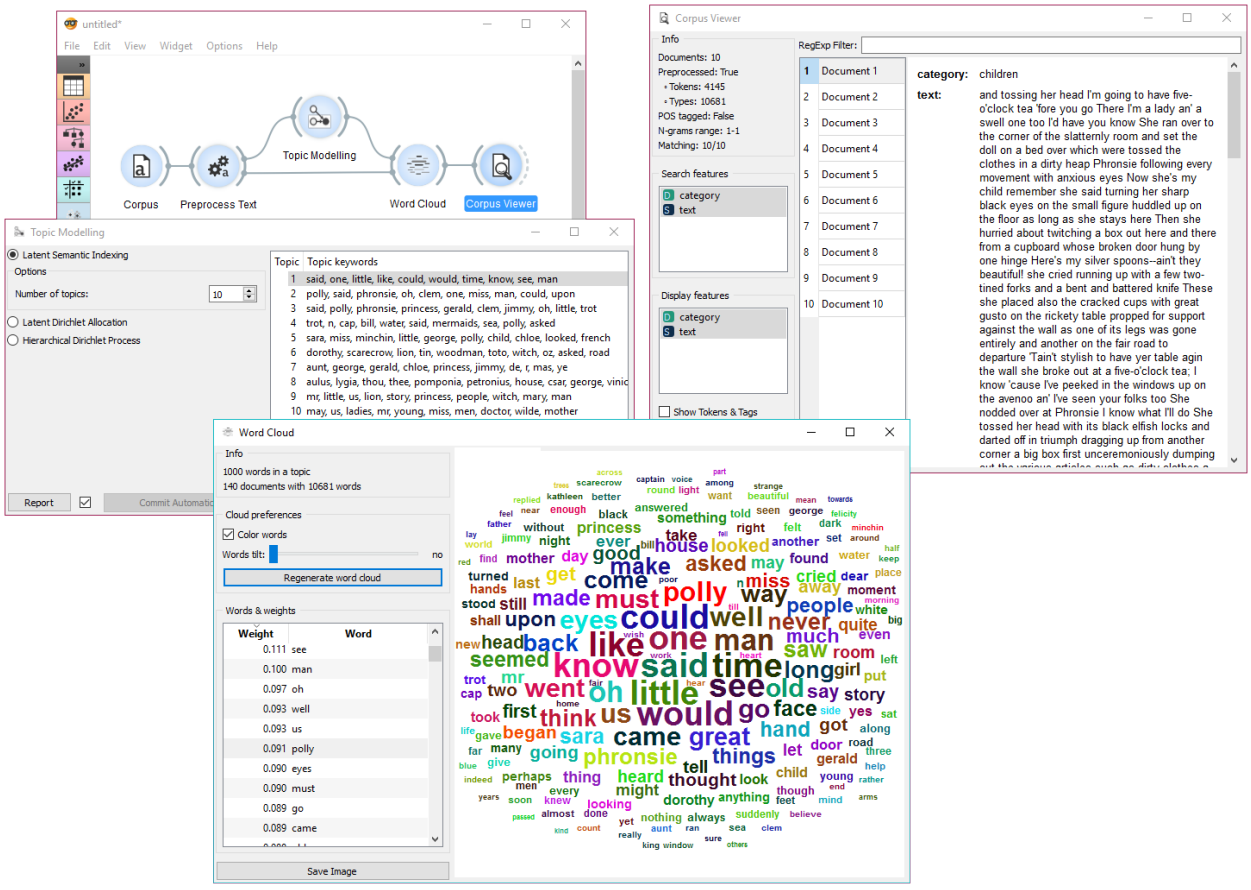


1. **Topic modelling algorithm:**
    - Latent Semantic Indexing
    - Latent Dirichlet Allocation
    - Hierarchical Dirichlet Process
2. **Parameters for the algorithm. LSI and LDA accept only the number of topics modelled, with the default set to 10. HDP, h**
    - First level concentration (γ): distribution at the first (corpus) level of Dirichlet Process
    - Second level concentration (α): distribution at the second (document) level of Dirichlet Process
    - The topic Dirichlet (α): concentration parameter used for the topic draws
    - Top level truncation (T): corpus-level truncation (no of topics)
    - Second level truncation (K): document-level truncation (no of topics)
    - Learning rate (κ): step size
    - Slow down parameter (τ)
3. Produce a report.
4. If *Commit Automatically* is on, changes are communicated automatically. Alternatively press *Commit*.

1.10.3 Example

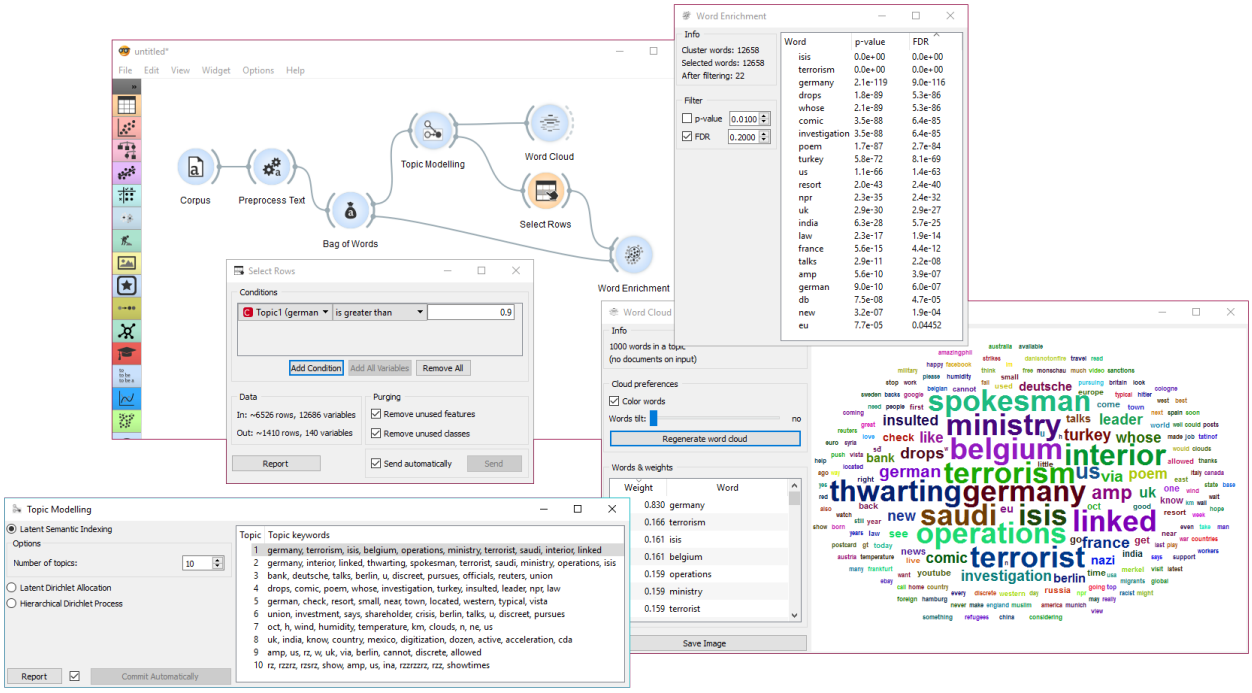
In the first example, we present a simple use of the **Topic Modelling** widget. First we load *bookexcerpts.tab* data set and use *Preprocess Text* to tokenize by words only. Then we connect **Preprocess Text** to **Topic Modelling**, where we use a simple *Latent Semantic Indexing* to find 10 topics in the text.

We then select the first topic and display the most frequent words in the topic in *Word Cloud*. We also connected **Preprocess Text** to **Word Cloud** in order to be able to output selected documents. Now we can select a specific word in the word cloud, say *polly*. It will be colored red and also highlighted in the word list on the left.


untitled*
File Edit View Widget Options Help
Corpus
Preprocess Text
Topic Modelling
Word Cloud
Corpus Viewer
Topic Modelling
Latent Semantic Indexing
Options
Number of topics: 10
Latent Dirichlet Allocation
Hierarchical Dirichlet Process
Report
Topic Topic keywords
1 said, one, little, like, could, would, time, know, see, man
2 polly, said, phronsie, oh, clem, one, miss, man, could, upon
3 said, polly, phronsie, princess, gerald, clem, jimmy, oh, little, trot
4 trot, n, cap, bill, water, said, mermaids, sea, polly, asked
5 sara, miss, minchin, little, george, polly, child, chloe, looked, french
6 dorothy, scarecrow, lion, tin, woodman, toto, witch, oz, asked, road
7 aunt, george, gerald, chloe, princess, jimmy, de, r, mas, ye
8 aulus, lygia, thou, thee, pomponia, petronius, house, csar, george, vinic
9 mr, little, us, lion, story, princess, people, witch, mary, man
10 may, us, ladies, mr, young, miss, men, doctor, wilde, mother
Corpus Viewer
Info
Documents: 10
Preprocessed: True
• Tokens: 4145
• Types: 10681
POS tagged: False
N-grams range: 1-1
Matching: 10/10
Search features
category
text
Display features
category
text
Show Tokens & Tags
RegExp Filter:
1 Document 1
2 Document 2
3 Document 3
4 Document 4
5 Document 5
6 Document 6
7 Document 7
8 Document 8
9 Document 9
10 Document 10
category: children
text: and tossing her head I'm going to have five-o'clock tea 'fore you go There I'm a lady an' a swell one too I'd have you know She ran over to the corner of the slatternly room and set the doll on a bed over which were tossed the clothes in a dirty heap Phronsie following every movement with anxious eyes Now she's my child remember she said turning her sharp black eyes on the small figure huddled up on the floor as long as she stays here Then she hurried about twitching a box out here and there from a cupboard whose broken door hung by one hinge Here's my silver spoons--ain't they beautiful! she cried running up with a few two-tined forks and a bent and battered knife These she placed also the cracked cups with great gusto on the rickety table propped for support against the wall as one of its legs was gone entirely and another on the fair road to departure 'Tain't stylish to have yer table agin the wall she broke out at a five-o'clock tea; I know 'cause I've peeked in the windows up on the avenoo an' I've seen your folks too She nodded over at Phronsie I know what I'll do She tossed her head with its black elfish locks and darted off in triumph dragging up from another corner a big box first unceremoniously dumping
Word Cloud
Info
1000 words in a topic
140 documents with 10681 words
Cloud preferences
Color words
Words tilt:
Regenerate word cloud
Words & weights
Weight Word
0.111 see
0.100 man
0.097 oh
0.093 well
0.093 us
0.091 polly
0.090 eyes
0.090 must
0.089 go
0.089 came
Save Image

Now we can observe all the documents containing the word *polly* in *Corpus Viewer*.

The second example will show how to use a more complex schema to find highly relevant words in a topic. We loaded a data set with recent tweets containing words 'Slovenia' and 'Germany'. We've done that with *Twitter* widget and saved it with **Save Data**. Since the data set was very big, we gathered the tweets and saved it to .tab format. Later we can always reload the saved data with *Corpus*.



Then we used *Preprocess Text* to tokenize by words and filter out numbers. Then we have to pass the data through *Bag of Words* in order to be able to use the corpus on *Word Enrichment*.

We pass the output of **Bag of Words** to **Topic Modelling**, where we select the first topic for inspection. We can already inspect word frequency of Topic 1 in **Word Cloud**.

Finally, we can use **Select Rows** to retrieve only those documents that have a weight of Topic 1 higher than 0.9 (meaning Topic 1 is represented in more than 9/10 of the document). Finally we connect **Select Rows** and **Bag of Words** to **Word Enrichment**. In **Word Enrichment** we can observe the most significant words in Topic 1.

1.11 Word Enrichment

Word enrichment analysis for selected documents.

1.11.1 Signals

Inputs:

- **Data**

  Corpus instance.

- **Selected Data**

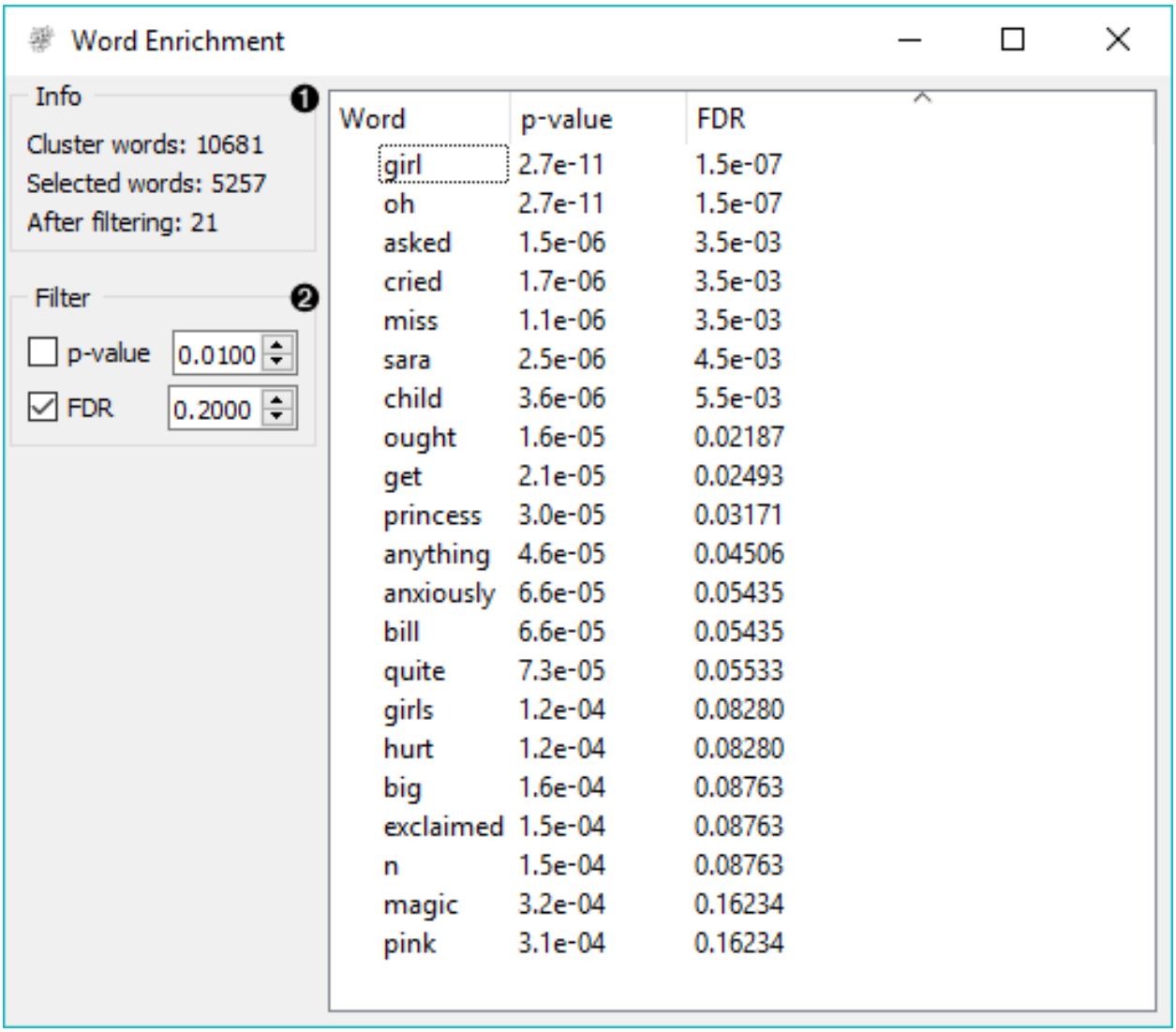
  Selected instances from corpus.

Outputs:

- (None)

1.11.2 Description

Word Enrichment displays a list of words with lower p-values (higher significance) for a selected subset compared to the entire corpus. Lower p-value indicates a higher likelihood that the word is significant for the selected subset (not randomly occurring in a text). FDR (False Discovery Rate) is linked to p-value and reports on the expected percent of false predictions in the set of predictions, meaning it account for false positives in list of low p-values.



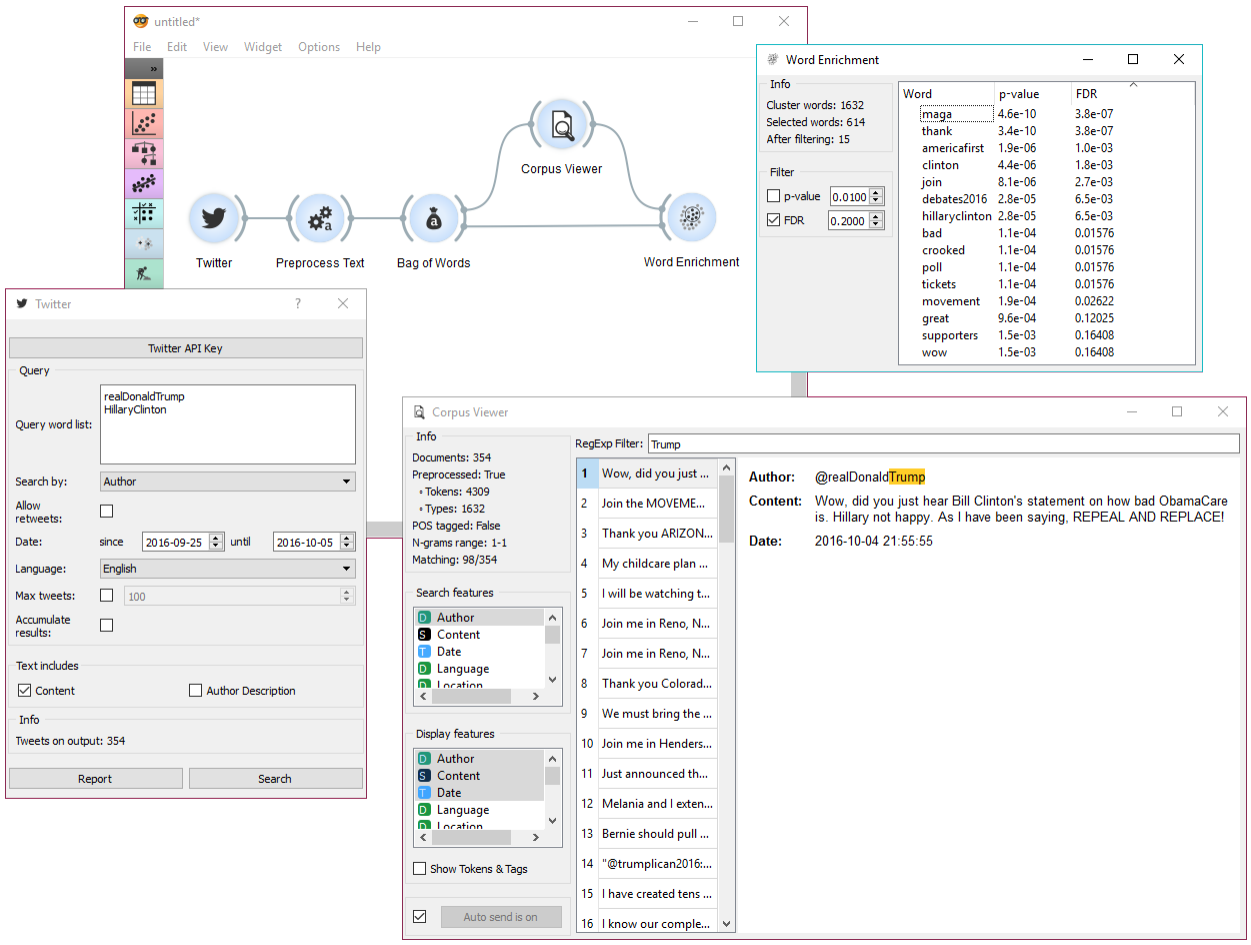
1. **Information on the input.**

- Cluster words are all the tokens from the corpus.
- Selected words are all the tokens from the selected subset.
- After filtering reports on the enriched words found in the subset.

2. **Filter enables you to filter by:**
   - p-value
   - false discovery rate (FDR)

1.11.3 Example

In the example below, we're retrieved recent tweets from the 2016 presidential candidates, Donald Trump and Hillary Clinton. Then we've preprocessed the tweets to get only words as tokens and to remove the stopwords. We've connected the preprocessed corpus to *Bag of Words* to get a table with word counts for our corpus.



Then we've connected *Corpus Viewer* to **Bag of Words** and selected only those tweets that were published by Donald Trump. See how we marked only the *Author* as our *Search feature* to retrieve those tweets.

Word Enrichment accepts two inputs - the entire corpus to serve as a reference and a selected subset from the corpus to do the enrichment on. First connect **Corpus Viewer** to **Word Enrichment** (input Matching Docs → Selected Data) and then connect **Bag of Words** to it (input Corpus → Data). In the **Word Enrichment** widget we can see the list of words that are more significant for Donald Trump than they are for Hillary Clinton.

1.12 Word Cloud

Generates a word cloud from corpus.

1.12.1 Signals

Inputs:

- **Topic**

  Selected topic.

- **Corpus**

  A *Corpus* instance.

Outputs:

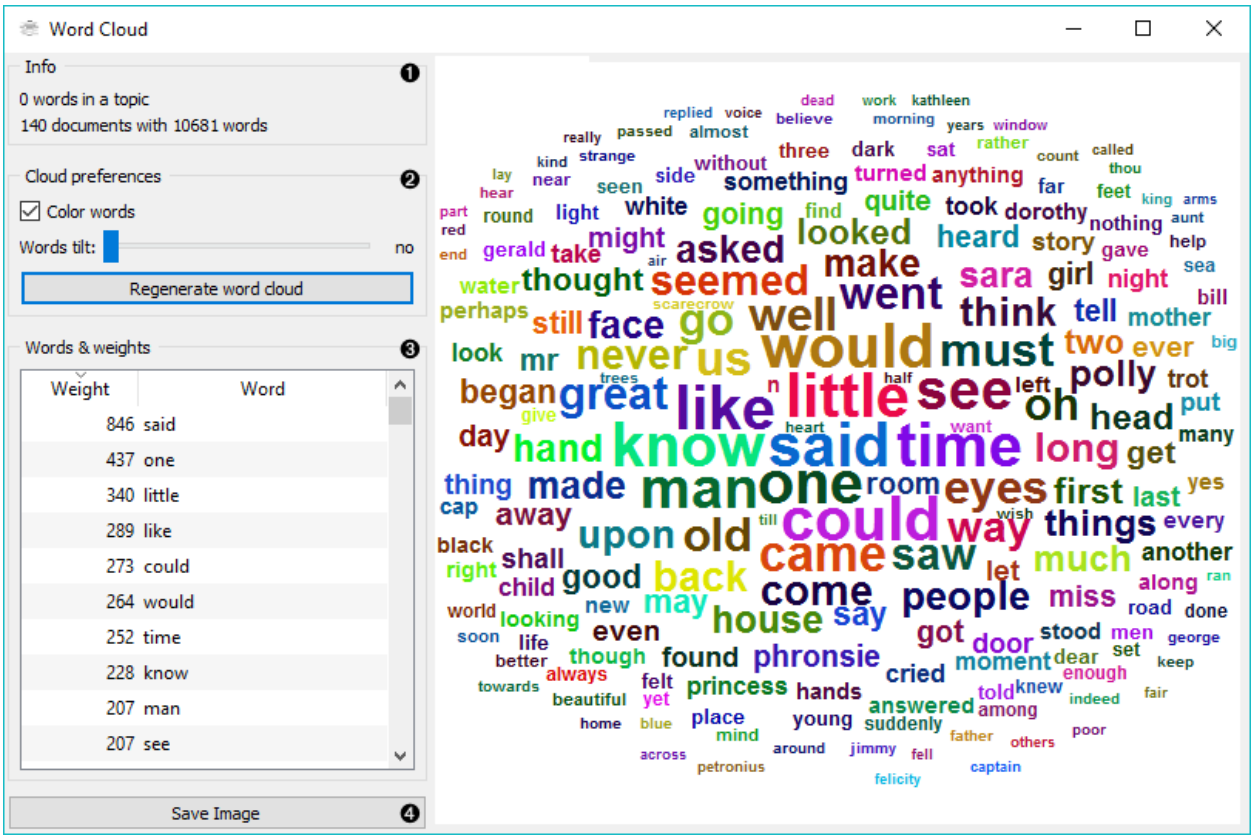
- **Corpus**

  Documents that match the selection.

1.12.2 Description

Word Cloud displays tokens in the corpus, their size denoting the frequency of the word in corpus. Words are listed by their frequency (weight) in the widget. The widget outputs documents, containing selected tokens from the word cloud.

1. **Information on the input.**
   - number of words (tokens) in a topic
   - number of documents and tokens in the corpus
2. **Adjust the plot.**
   - If *Color words* is ticked, words will be assigned a random color. If unchecked, the words will be black.
   - *Word tilt* adjust the tilt of words. The current state of tilt is displayed next to the slider ('no' is the default).
   - *Regenerate word cloud* plot the cloud anew.
3. Words & weights displays a sorted list of words (tokens) by their frequency in the corpus or topic. Clicking on a word will select that same word in the cloud and output matching documents. Use *Ctrl* to select more than one word. Documents matching ANY of the selected words will be on the output (logical OR).
4. *Save Image* saves the image to your computer in a .svg or .png format.



1.12.3 Example

Word Cloud is an excellent widget for displaying the current state of the corpus and for monitoring the effects of preprocessing.

Use *Corpus* to load the data. Connect *Preprocess Text* to it and set your parameters. We've used defaults here, just to see the difference between the default preprocessing in the **Word Cloud** widget and the **Preprocess Text** widget.

We can see from the two widgets, that **Preprocess Text** displays only words, while default preprocessing in the **Word Cloud** tokenizes by word and punctuation.

1.13 GeoMap

Displays geographic distribution of data.

1.13.1 Signals

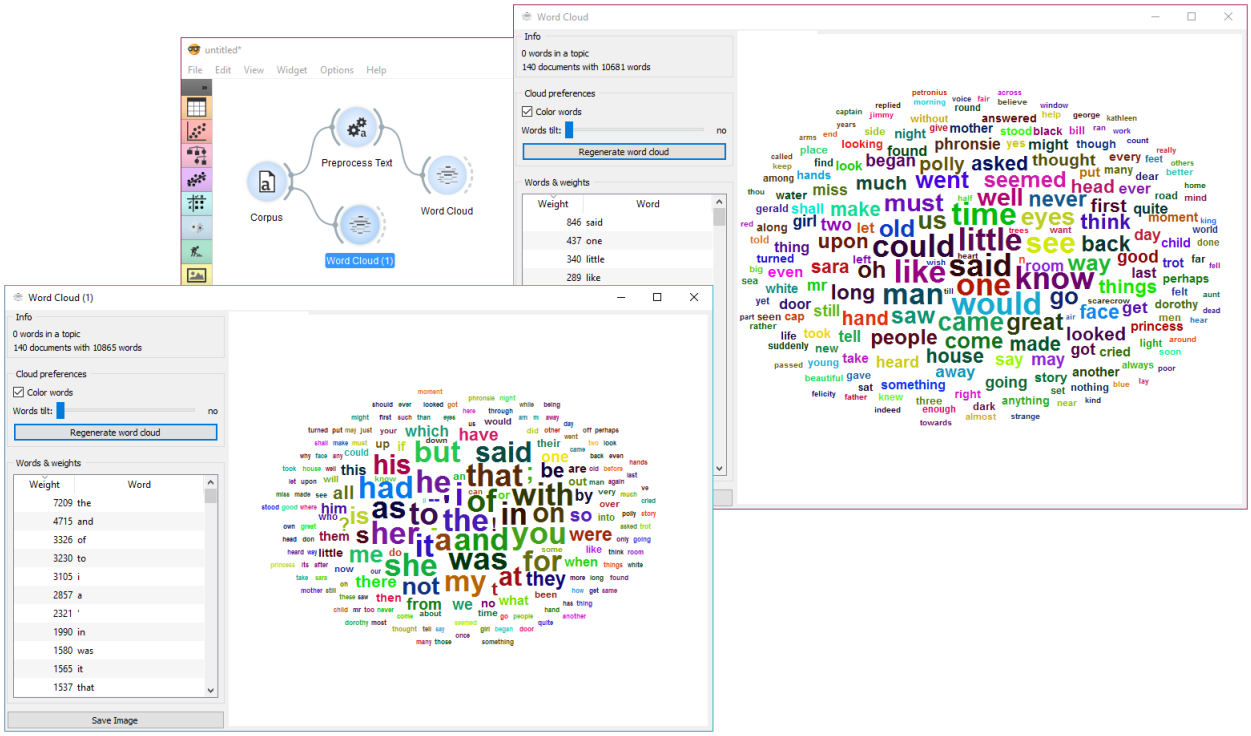
Inputs:

- **Data**

  Data set.

Outputs:

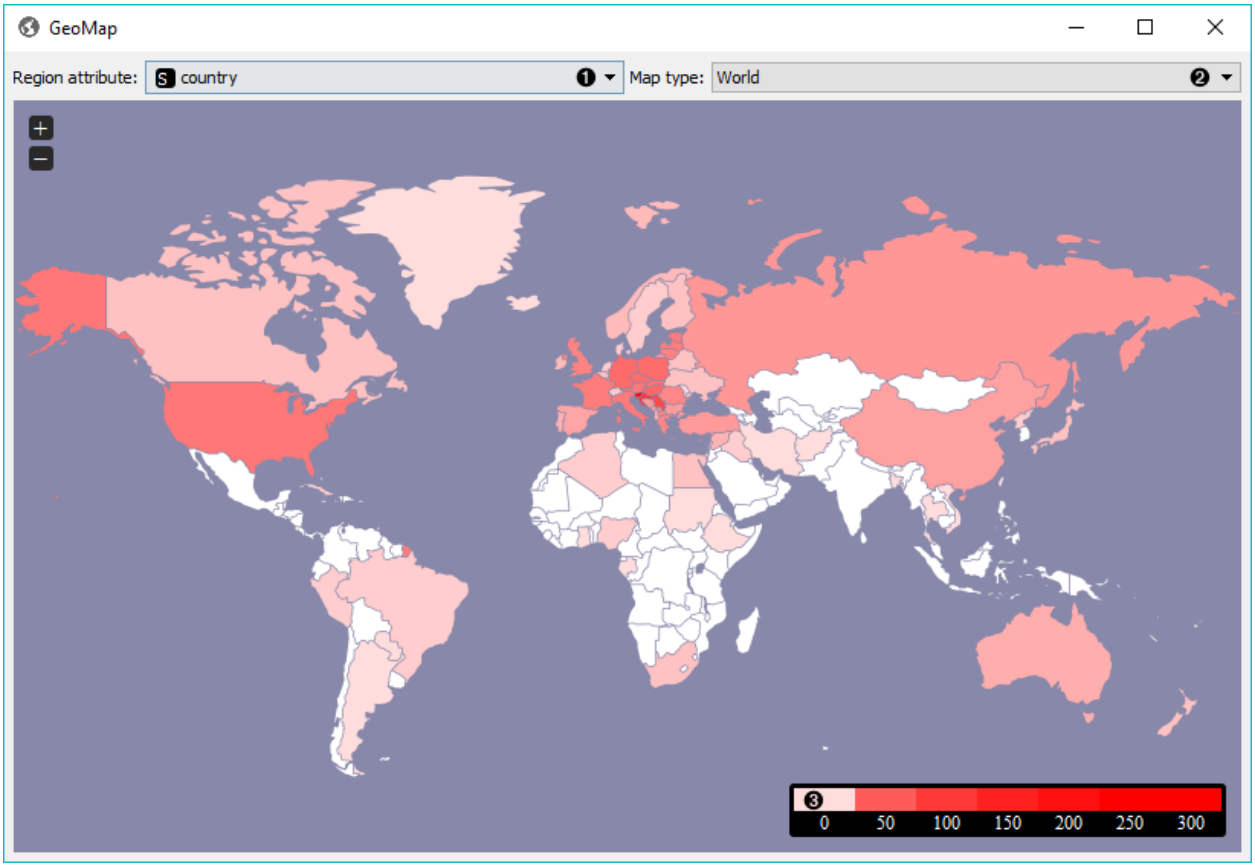
- **Corpus**


untitled*
File Edit View Widget Options Help
Corpus
Preprocess Text
Word Cloud
Word Cloud (1)
Word Cloud
Info
0 words in a topic
140 documents with 10681 words
Cloud preferences
Color words
Words tilt:
no
Regenerate word cloud
Words & weights
Weight
Word
846 said
437 one
340 little
289 like
Word Cloud (1)
Info
0 words in a topic
140 documents with 10865 words
Cloud preferences
Color words
Words tilt:
no
Regenerate word cloud
Words & weights
Weight
Word
7209 the
4715 and
3326 of
3230 to
3105 i
2857 a
2321 '
1990 in
1580 was
1565 it
1537 that
Save Image

A *Corpus* instance.

1.13.2 Description

GeoMap widget shows geolocations from textual (string) data. It finds mentions of geographic names (countries and capitals) and displays distributions (frequency of mentiones) of these names on a map. It works with any Orange widget that outputs a data table and that contains at least one string attribute. The widget outputs selected data instances, that is all documents containing mentions of a selected country (or countries).

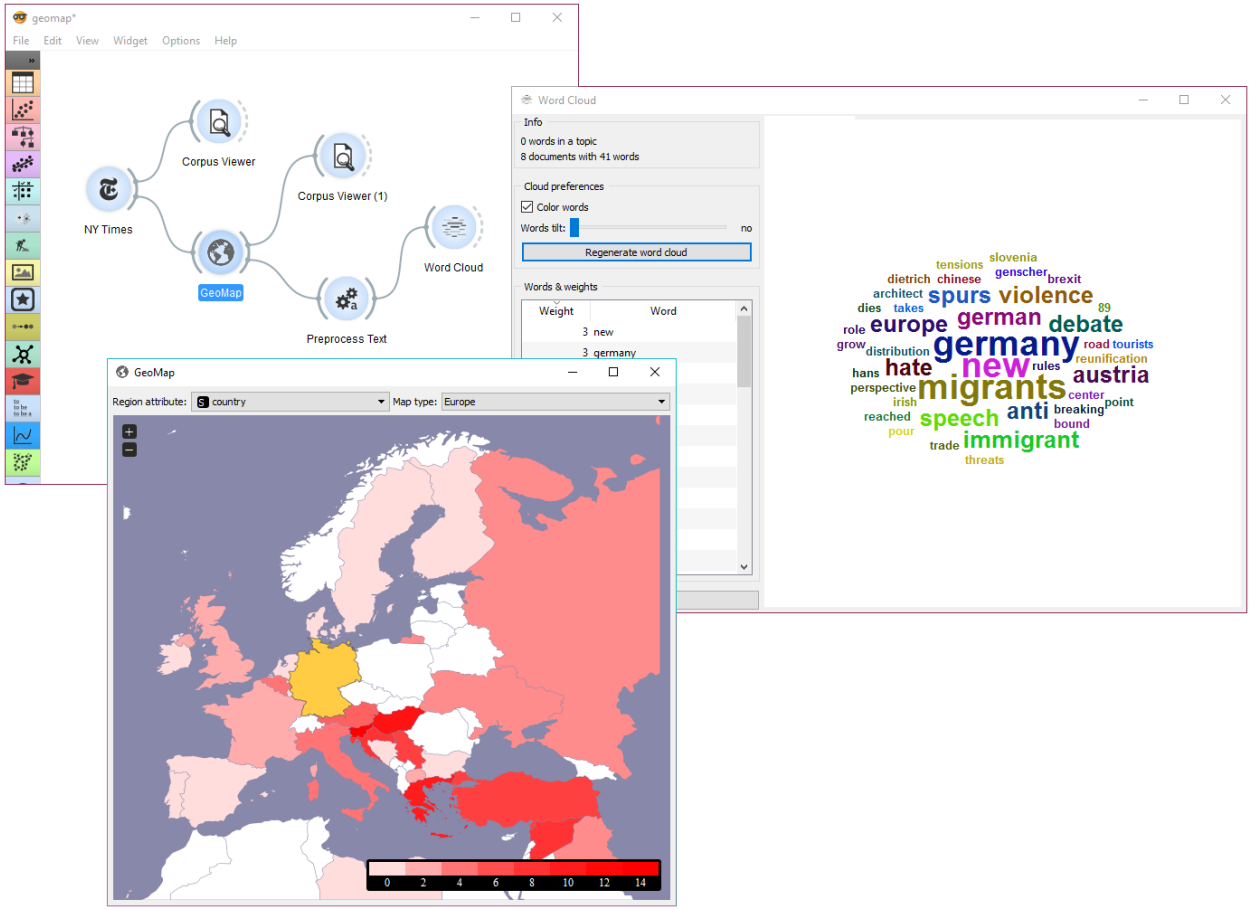


1. Select the meta attribute you want to search geolocations by. The widget will find all mentions of geolocations in a text and display distributions on a map.
2. Select the type of map you wish to display. The options are *World*, *Europe* and *USA*. You can zoom in and out of the map by pressing + and - buttons on a map or by mouse scroll.
3. The legend for the geographic distribution of data. Countries with the boldest color are most often mentioned in the selected region attribute (highest frequency).

To select documents mentioning a specific country, click on a country and the widget will output matching documents. To select more than one country hold Ctrl/Cmd upon selection.

1.13.3 Example

GeoMap widget can be used for simply visualizing distributions of geolocations or for a more complex interactive data analysis. Here, we've queried *NY Times* for articles on Slovenia for the time period of the last year (2015-2016). First we checked the results with *Corpus Viewer*.


geomap*
File Edit View Widget Options Help
Corpus Viewer
Corpus Viewer (1)
NY Times
GeoMap
Word Cloud
Preprocess Text
Word Cloud
Info
0 words in a topic
8 documents with 41 words
Cloud preferences
Color words
Words tilt:
no
Regenerate word cloud
Words & weights
Weight
Word
3 new
3 germany
tensions slovenia
dietrich chinese genscher brexit
architect spurs violence
dies takes 89
role europe german debate
grow distribution germany road tourists
reunification
hans hate new rules austria
perspective migrants center
irish point
reached speech anti breaking
bound
pour
trade immigrant
threats
GeoMap
Region attribute: country
Map type: Europe
0 2 4 6 8 10 12 14

Then we sent the data to **GeoMap** to see distributiosn of geolocations by *country* attribute. The attribute already contains country tags for each article, which is why **NY Times** is great in combinations with **GeoMap**. We selected Germany, which sends all the documents tagged with Germany to the output. Remember, we queried **NY Times** for articles on Slovenia.

We can again inspect the output with **Corpus Viewer**. But there's a more interesting way of visualizing the data. We've sent selected documents to *Preprocess Text*, where we've tokenized text to words and removed stopwords.

Finally, we can inspect the top words appearing in last year's documents on Slovenia and mentioning also Germany with *Word Cloud*.

1.14 Concordance

Display the context of the word.

1.14.1 Signals

Inputs:

- **Corpus**

  A *Corpus* instance.

Outputs:

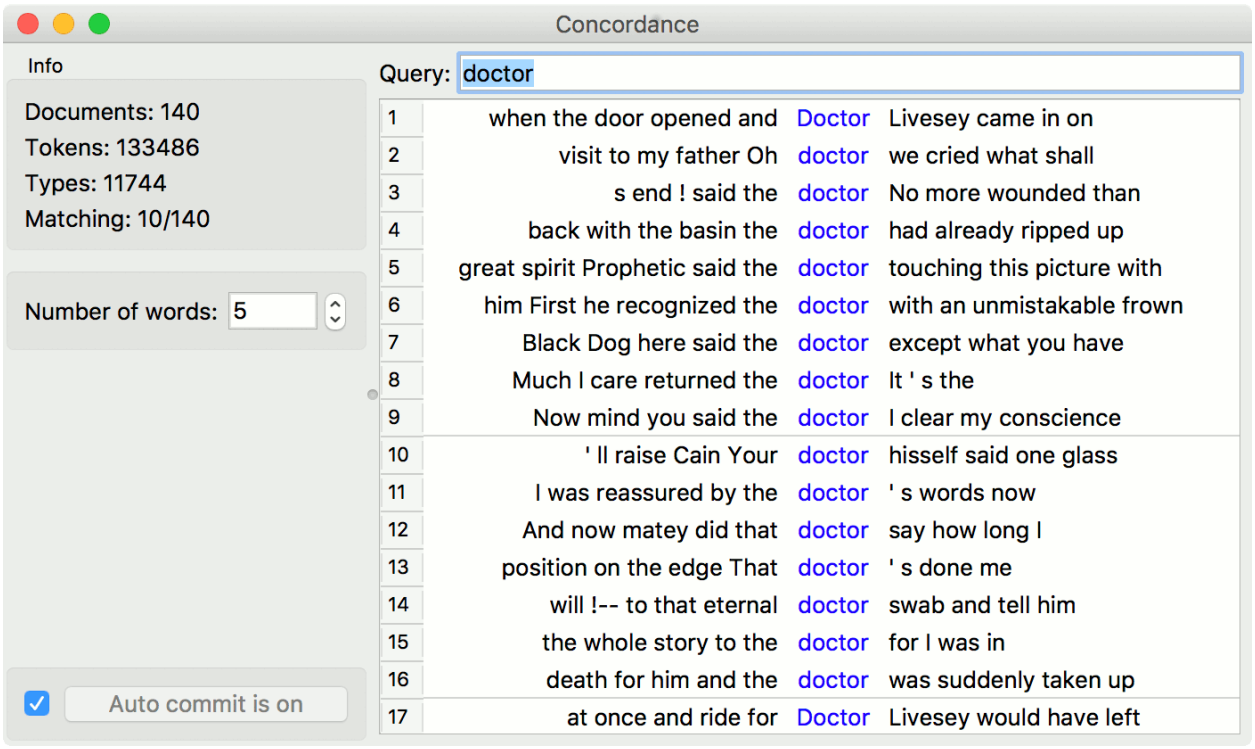
- **Selected Documents**

  A *Corpus* instance.

1.14.2 Description

Concordance finds the queried word in a text and displays the context in which this word is used. It can output selected documents for further analysis.

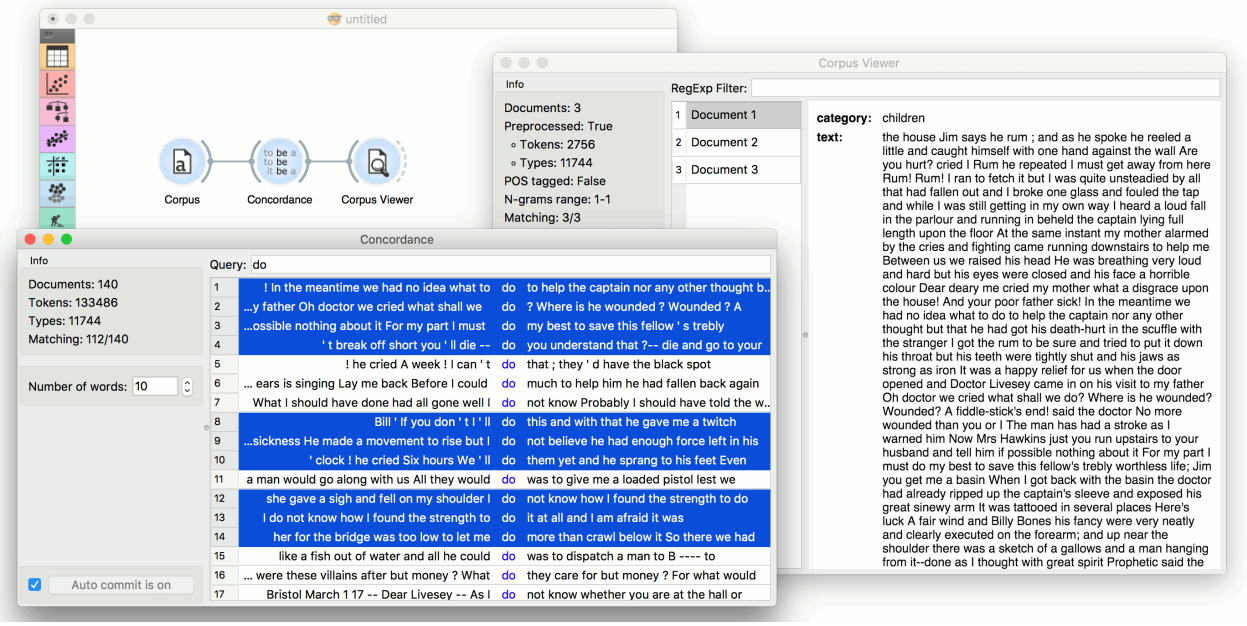
1. ***Information*:**
   - *Documents*: number of documents on the input.
   - *Tokens*: number of tokens on the input.
   - *Types*: number of unique tokens on the input.
   - *Matching*: number of documents containing the queried word.
2. *Number of words*: the number of words displayed on each side of the queried word.
3. Queried word.
4. If *Auto commit is on*, selected documents are communicated automatically. Alternatively press *Commit*.



1.14.3 Example

Concordance can be used for displaying word contexts in a corpus. First, we load *bookexcerpts.tab* in *Corpus*. Then we connect **Corpus** to **Concordances** and search for concordances of a word "doctor". The widget displays all documents containing the word "doctor" together with their surrounding (contextual) words. Note that the widget finds only exact matches of a word.

Now we can select those documents that contain interesting contexts and output them to *Corpus Viewer* to inspect them further.


untitled
Corpus
Concordance
Corpus Viewer
Corpus Viewer
Info
Documents: 3
Preprocessed: True
Tokens: 2756
Types: 11744
POS tagged: False
N-grams range: 1-1
Matching: 3/3
RegExp Filter:
1 Document 1
2 Document 2
3 Document 3
category: children
text: the house Jim says he rum ; and as he spoke he reeled a little and caught himself with one hand against the wall Are you hurt? cried I Rum he repeated I must get away from here Rum! Rum! I ran to fetch it but I was quite unsteadied by all that had fallen out and I broke one glass and fouled the tap and while I was still getting in my own way I heard a loud fall in the parlour and running in beheld the captain lying full length upon the floor At the same instant my mother alarmed by the cries and fighting came running downstairs to help me Between us we raised his head He was breathing very loud and hard but his eyes were closed and his face a horrible colour Dear deary me cried my mother what a disgrace upon the house! And your poor father sick! In the meantime we had no idea what to do to help the captain nor any other thought but that he had got his death-hurt in the scuffle with the stranger I got the rum to be sure and tried to put it down his throat but his teeth were tightly shut and his jaws as strong as iron It was a happy relief for us when the door opened and Doctor Livesey came in on his visit to my father Oh doctor we cried what shall we do? Where is he wounded? Wounded? A fiddle-stick's end! said the doctor No more wounded than you or I The man has had a stroke as I warned him Now Mrs Hawkins just you run upstairs to your husband and tell him if possible nothing about it For my part I must do my best to save this fellow's trebly worthless life; Jim you get me a basin When I got back with the basin the doctor had already ripped up the captain's sleeve and exposed his great sinewy arm It was tattooed in several places Here's luck A fair wind and Billy Bones his fancy were very neatly and clearly executed on the forearm; and up near the shoulder there was a sketch of a gallows and a man hanging from it--done as I thought with great spirit Prophetic said the
Concordance
Info
Documents: 140
Tokens: 133486
Types: 11744
Matching: 112/140
Query: do
Number of words: 10
Auto commit is on
1 ! In the meantime we had no idea what to do to help the captain nor any other thought b..
2 ...y father Oh doctor we cried what shall we do ? Where is he wounded ? Wounded ? A
3 ...ossible nothing about it For my part I must do my best to save this fellow ' s trebly
4 ' t break off short you ' ll die -- do you understand that ?-- die and go to your
5 ! he cried A week ! I can ' t do that ; they ' d have the black spot
6 ... ears is singing Lay me back Before I could do much to help him he had fallen back again
7 What I should have done had all gone well I do not know Probably I should have told the w..
8 Bill ' If you don ' t I ' ll do this and with that he gave me a twitch
9 ...sickness He made a movement to rise but I do not believe he had enough force left in his
10 ' clock ! he cried Six hours We ' ll do them yet and he sprang to his feet Even
11 a man would go along with us All they would do was to give me a loaded pistol lest we
12 she gave a sigh and fell on my shoulder I do not know how I found the strength to do
13 I do not know how I found the strength to do it at all and I am afraid it was
14 her for the bridge was too low to let me do more than crawl below it So there we had
15 like a fish out of water and all he could do was to dispatch a man to B ---- to
16 ... were these villains after but money ? What do they care for but money ? For what would
17 Bristol March 1 17 -- Dear Livesey -- As I do not know whether you are at the hall or

CHAPTER 2

Scripting

2.1 Corpus

class orangecontrib.text.corpus.**Corpus**(*domain=None*, *X=None*, *Y=None*, *metas=None*, *W=None*, *text_features=None*, *ids=None*)

Internal class for storing a corpus.

__init__(*domain=None*, *X=None*, *Y=None*, *metas=None*, *W=None*, *text_features=None*, *ids=None*)

Parameters

- **domain** (*Orange.data.Domain*) – the domain for this Corpus
- **X** (*numpy.ndarray*) – attributes
- **Y** (*numpy.ndarray*) – class variables
- **metas** (*numpy.ndarray*) – meta attributes; e.g. text
- **W** (*numpy.ndarray*) – instance weights
- **text_features** (*list*) – meta attributes that are used for text mining. Infer them if None.
- **ids** (*numpy.ndarray*) – Indices

copy()

Return a copy of the table.

dictionary

corpora.Dictionary – A token to id mapper.

documents

Returns – a list of strings representing documents — created by joining selected text features.

documents_from_features(*feats*)

Parameters **feats** (*list*) – A list fo features to join.

Returns: a list of strings constructed by joining feats.

extend_attributes(*X*, *feature_names*, *feature_values=None*, *compute_values=None*, *var_attrs=None*)
Append features to corpus. If *feature_values* argument is present, features will be Discrete else Continuous.

Parameters

- **X** (*numpy.ndarray or scipy.sparse.csr_matrix*) – Features values to append
- **feature_names** (*list*) – List of string containing feature names
- **feature_values** (*list*) – A list of possible values for Discrete features.
- **compute_values** (*list*) – Compute values for corresponding features.
- **var_attrs** (*dict*) – Additional attributes appended to variable.attributes.

extend_corpus(*metadata*, *Y*)
Append documents to corpus.

Parameters

- **metadata** (*numpy.ndarray*) – Meta data
- **Y** (*numpy.ndarray*) – Class variables

static **from_documents**(*documents*, *name*, *attributes=None*, *class_vars=None*, *metas=None*, *title_indices=None*)
Create corpus from documents.

Parameters

- **documents** (*list*) – List of documents.
- **name** (*str*) – Name of the corpus
- **attributes** (*list*) – List of tuples (Variable, getter) for attributes.
- **class_vars** (*list*) – List of tuples (Variable, getter) for class vars.
- **metas** (*list*) – List of tuples (Variable, getter) for metas.
- **title_indices** (*list*) – List of indices into domain corresponding to features which will be used as titles.

Returns Corpus.

has_tokens()
Return whether corpus is preprocessed or not.

ngrams
generator – Ngram representations of documents.

static **retain_preprocessing**(*orig*, *new*, *key=Ellipsis*)
Set preprocessing of 'new' object to match the 'orig' object.

set_text_features(*feats*)
Select which meta-attributes to include when mining text.

Parameters **feats** (*list or None*) – List of text features to include. If None infer them.

store_tokens(*tokens*, *dictionary=None*)

Parameters **tokens** (*list*) – List of lists containing tokens.

titles
Returns a list of titles.

tokens
np.ndarray – A list of lists containing tokens. If tokens are not yet present, run default preprocessor and save tokens.

2.2 Preprocessor

This module provides basic functions to process *Corpus* and extract tokens from documents.

To use preprocessing you should create a corpus:

```
>>> from orangecontrib.text import Corpus
>>> corpus = Corpus.from_file('bookexcerpts')
```

And create a *Preprocessor* objects with methods you want:

```
>>> from orangecontrib.text import preprocess
>>> p = preprocess.Preprocessor(transformers=[preprocess.LowercaseTransformer()],
...                             tokenizer=preprocess.WordPunctTokenizer(),
...                             normalizer=preprocess.SnowballStemmer('english'),
...                             filters=[preprocess.StopwordsFilter('english'),
...                                      preprocess.FrequencyFilter(min_df=.1)])
```

Then you can apply you preprocessor to the corpus and access tokens via `tokens` attribute:

```
>>> new_corpus = p(corpus)
>>> new_corpus.tokens[0][:10]
['hous', 'say', ';', 'spoke', 'littl', 'one', 'hand', 'wall', 'hurt', '?']
```

This module defines `default_preprocessor` that will be used to extract tokens from a *Corpus* if no preprocessing was applied yet:

```
>>> from orangecontrib.text import Corpus
>>> corpus = Corpus.from_file('deerwester')
>>> corpus.tokens[0]
['human', 'machine', 'interface', 'for', 'lab', 'abc', 'computer', 'applications']
```

class `orangecontrib.text.preprocess.`**Preprocessor**(*transformers=None*, *tokenizer=None*, *normalizer=None*, *filters=None*, *ngrams_range=None*, *pos_tagger=None*)

Holds document processing objects.

transformers
List([BaseTransformer] – transforms strings

tokenizer
BaseTokenizer – tokenizes string

normalizer
BaseNormalizer – normalizes tokens

filters
List[BaseTokenFilter] – filters unneeded tokens

__call__(*corpus*, *inplace=True*, *on_progress=None*)
Runs preprocessing over a corpus.

Parameters

- **corpus** (*orangecontrib.text.Corpus*) – A corpus to preprocess.
- **inplace** (*bool*) – Whether to create a new Corpus instance.

set_up()
Called before every __call__. Used for setting up tokenizer & filters.

tear_down()
Called after every __call__. Used for cleaning up tokenizer & filters.

2.3 Twitter

class orangecontrib.text.twitter.**Credentials**(*consumer_key*, *consumer_secret*)
Twitter API credentials.

class orangecontrib.text.twitter.**TwitterAPI**(*credentials*, *on_progress=None*, *should_break=None*, *on_error=None*, *on_rate_limit=None*)
Fetch tweets from the Tweeter API.

Notes

Results across multiple searches are aggregated. To remove tweets form previous searches and only return results from the last search either call *reset* method before searching or provide *collecting=False* argument to search method.

reset()
Removes all downloaded tweets.

search_authors(*authors*, *, *max_tweets=0*, *collecting=False*)
Search by authors.

Parameters

- **authors** (*list of str*) – A list of authors to search for.
- **max_tweets** (*int*) – If greater than zero limits the number of downloaded tweets.
- **collecting** (*bool*) – Whether to collect results across multiple search calls.

Returns Corpus

search_content(*content*, *, *max_tweets=0*, *lang=None*, *allow_retweets=True*, *collecting=False*)
Search by content.

Parameters

- **content** (*list of str*) – A list of key words to search for.
- **max_tweets** (*int*) – If greater than zero limits the number of downloaded tweets.
- **lang** (*str*) – A language's code (either ISO 639-1 or ISO 639-3 formats).
- **allow_retweets** (*bool*) – Whether to download retweets.
- **collecting** (*bool*) – Whether to collect results across multiple search calls.

Returns Corpus

2.4 New York Times

class orangecontrib.text.nyt.**NYT**(*api_key*)
Class for fetching records from the NYT API.

__init__(*api_key*)

Parameters **api_key** (*str*) – NY Time API key.

api_key_valid()
Checks whether api key given at initialization is valid.

search(*query*, *date_from=None*, *date_to=None*, *max_docs=None*, *on_progress=None*, *should_break=None*)

Parameters

- **query** (*str*) – Search query.
- **date_from** (*date*) – Start date limit.
- **date_to** (*date*) – End date limit.
- **max_docs** (*int*) – Maximal number of documents returned.
- **on_progress** (*callback*) – Called after every iteration of downloading.
- **should_break** (*callback*) – Callback for breaking the computation before the end. If it evaluates to True, downloading is stopped and document downloaded till now are returned in a Corpus.

Returns Search results.

Return type *Corpus*

2.5 The Guardian

This module fetches data from The Guardian API.

To use first create *TheGuardianCredentials*:

```
>>> from orangecontrib.text.guardian import TheGuardianCredentials
>>> credentials = TheGuardianCredentials('<your-api-key>')
```

Then create *TheGuardianAPI* object and use it for searching:

```
>>> from orangecontrib.text.guardian import TheGuardianAPI
>>> api = TheGuardianAPI(credentials)
>>> corpus = api.search('Slovenia', max_documents=10)
>>> len(corpus)
10
```

class orangecontrib.text.guardian.**TheGuardianCredentials**(*key*)
The Guardian API credentials.

__init__(*key*)

Parameters **key** (*str*) – The Guardian API key. Use *test* for testing purposes.

valid
Check if given API key is valid.

class orangecontrib.text.guardian.**TheGuardianAPI**(*credentials*, *on_progress=None*, *should_break=None*)

__init__(*credentials*, *on_progress=None*, *should_break=None*)

Parameters

- **credentials** (*TheGuardianCredentials*) – The Guardian Creentials.
- **on_progress** (*callable*) – Function for progress reporting.
- **should_break** (*callable*) – Function for early stopping.

search(*query*, *from_date=None*, *to_date=None*, *max_documents=None*, *accumulate=False*)
Search The Guardian API for articles.

Parameters

- **query** (*str*) – A query for searching the articles by
- **from_date** (*str*) – Search only articles newer than the date provided. Date should be in ISO format; e.g. '2016-12-31'.
- **to_date** (*str*) – Search only articles older than the date provided. Date should be in ISO format; e.g. '2016-12-31'.
- **max_documents** (*int*) – Maximum number of documents to retrieve. When not given, retrieve all documents.
- **accumulate** (*bool*) – A flag indicating whether to accumulate results of multiple consequent search calls.

Returns *Corpus*

2.6 Wikipedia

class orangecontrib.text.wikipedia.**WikipediaAPI**(*on_error=None*)
Wraps Wikipedia API.

Examples

```
>>> api = WikipediaAPI()
>>> corpus = api.search('en', ['Barack Obama', 'Hillary Clinton'])
```

search(*lang*, *queries*, *articles_per_query=10*, *should_break=None*, *on_progress=None*)
Searches for articles.

Parameters

- **lang** (*str*) – A language code in ISO 639-1 format.
- **queries** (*list of str*) – A list of queries.
- **should_break** (*callback*) – Callback for breaking the computation before the end. If it evaluates to True, downloading is stopped and document downloaded till now are returned in a Corpus.
- **on_progress** (*callable*) – Callback for progress bar.

2.7 Topic Modeling

class orangecontrib.text.topics.**LdaWrapper**(***kwargs*)

fit(*corpus*, ***kwargs*)
Train the model with the corpus.

Parameters **corpus** (Corpus) – A corpus to learn topics from.

transform(*corpus*)
Create a table with topics representation.

class orangecontrib.text.topics.**LsiWrapper**(***kwargs*)

fit(*corpus*, ***kwargs*)
Train the model with the corpus.

Parameters **corpus** (Corpus) – A corpus to learn topics from.

transform(*corpus*)
Create a table with topics representation.

class orangecontrib.text.topics.**HdpWrapper**(***kwargs*)

fit(*corpus*, ***kwargs*)
Train the model with the corpus.

Parameters **corpus** (Corpus) – A corpus to learn topics from.

transform(*corpus*)
Create a table with topics representation.

2.8 Tag

A module for tagging *Corpus* instances.

This module provides a default *pos_tagger* that can be used for POSTagging an English corpus:

```
>>> from orangecontrib.text.corpus import Corpus
>>> from orangecontrib.text.tag import pos_tagger
>>> corpus = Corpus.from_file('deerwester.tab')
>>> tagged_corpus = pos_tagger.tag_corpus(corpus)
>>> tagged_corpus.pos_tags[0]  # you can use `pos_tags` attribute to access tags␣
↪directly
['JJ', 'NN', 'NN', 'IN', 'NN', 'NN', 'NN', 'NNS']
>>> next(tagged_corpus.ngrams_iterator(include_postags=True))  # or `ngrams_iterator`␣
↪to iterate over documents
['human_JJ', 'machine_NN', 'interface_NN', 'for_IN', 'lab_NN', 'abc_NN', 'computer_NN
↪', 'applications_NNS']
```

class orangecontrib.text.tag.**POSTagger**(*tagger*, *name='POS Tagger'*)
A class that wraps *nltk.TaggerI* and performs Corpus tagging.

tag_corpus(*corpus*, ***kwargs*)
Marks tokens of a corpus with POS tags.

Parameters **corpus** (orangecontrib.text.corpus.Corpus) – A corpus instance.

class orangecontrib.text.tag.**StanfordPOSTagger**(**args*, ***kwargs*)

classmethod check(*path_to_model*, *path_to_jar*)
Checks whether provided *path_to_model* and *path_to_jar* are valid.

Raises ValueError – in case at least one of the paths is invalid.

Notes

Can raise an exception if Java Development Kit is not installed or not properly configured.

Examples

```
>>> try:
...     StanfordPOSTagger.check('path/to/model', 'path/to/stanford.jar')
... except ValueError as e:
...     print(e)
Could not find stanford-postagger.jar jar file at path/to/stanford.jar
```

2.9 Async Module

Helper utils for Orange GUI programming.

Provides asynchronous() decorator for making methods calls in async mode. Once method is decorated it will have task.on_start(), task.on_result() and task.callback() decorators for callbacks wrapping.

- *on_start* must take no arguments
- *on_result* must accept one argument (the result)
- *callback* can accept any arguments

For instance:

```
class Widget(QObject):
    def __init__(self, name):
        super().__init__()
        self.name = name

    @asynchronous
    def task(self):
        for i in range(3):
            time.sleep(0.5)
            self.report_progress(i)
        return 'Done'

    @task.on_start
    def report_start(self):
        print('`{}` started'.format(self.name))

    @task.on_result
    def report_result(self, result):
        print('`{}` result: {}'.format(self.name, result))
```

```
    @task.callback
    def report_progress(self, i):
        print('`{}` progress: {}'.format(self.name, i))
```

Calling an asynchronous method will launch a daemon thread:

```
first = Widget(name='First')
first.task()
second = Widget(name='Second')
second.task()

first.task.join()
second.task.join()
```

A possible output:

```
`First` started
`Second` started
`Second` progress: 0
`First` progress: 0
`First` progress: 1
`Second` progress: 1
`First` progress: 2
`First` result: Done
`Second` progress: 2
`Second` result: Done
```

In order to terminate a thread either call `stop()` method or raise `StopExecution` exception within `task()`:

```
first.task.stop()
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

T

V

W