
Orange3 Text Mining Documentation

Release

Biolab

January 26, 2017

1	Corpus	1
2	NY Times	5
3	Twitter	9
4	Wikipedia	13
5	Pubmed	17
6	Corpus Viewer	21
7	Preprocess Text	25
8	Bag of Words	31
9	Topic Modelling	35
10	Word Enrichment	39
11	Word Cloud	43
12	GeoMap	47
13	Corpus	51
14	Preprocessor	53
15	Twitter	55
16	New York Times	57
17	Wikipedia	59
18	Topic Modeling	61
19	Tag	63
20	Indices and tables	65

Corpus



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

1.1 Signals

Inputs:

- (None)

Outputs:

- **Corpus**

A *Corpus* instance.

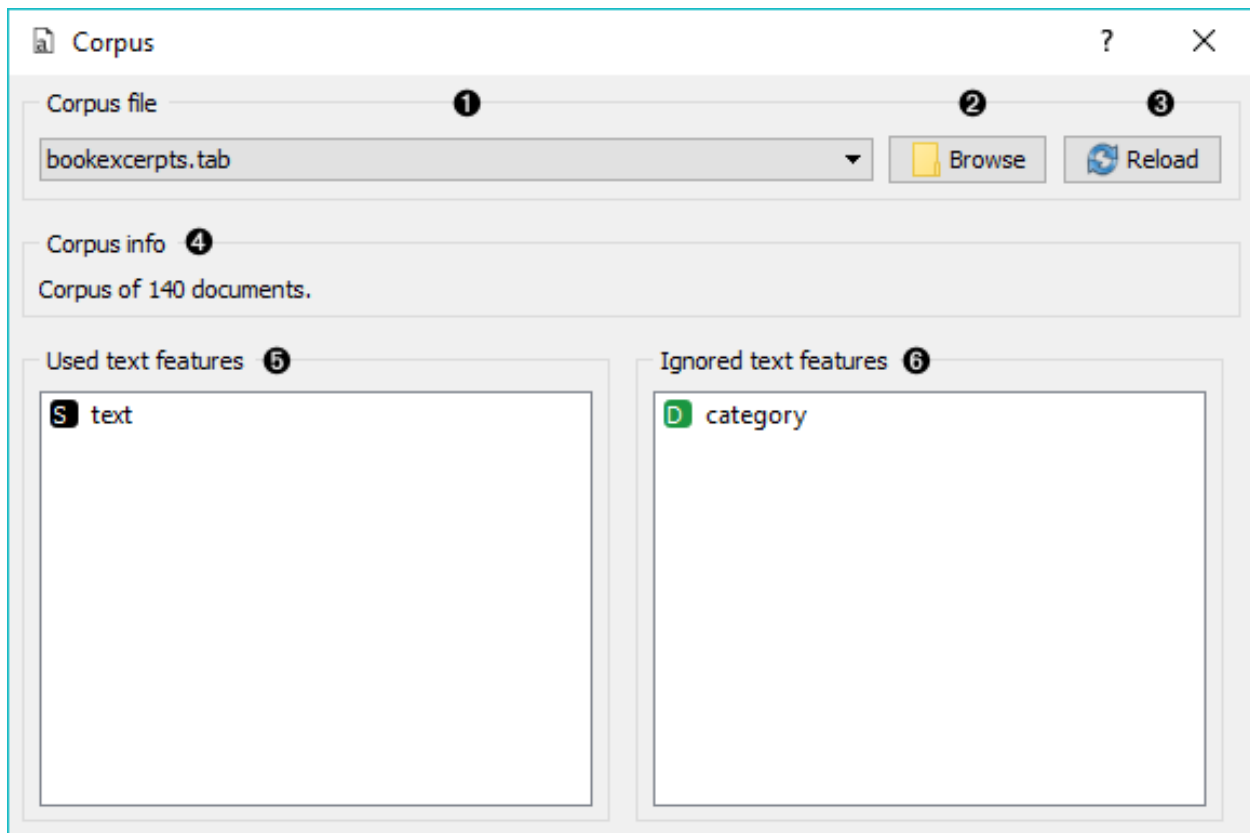
1.2 Description

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.

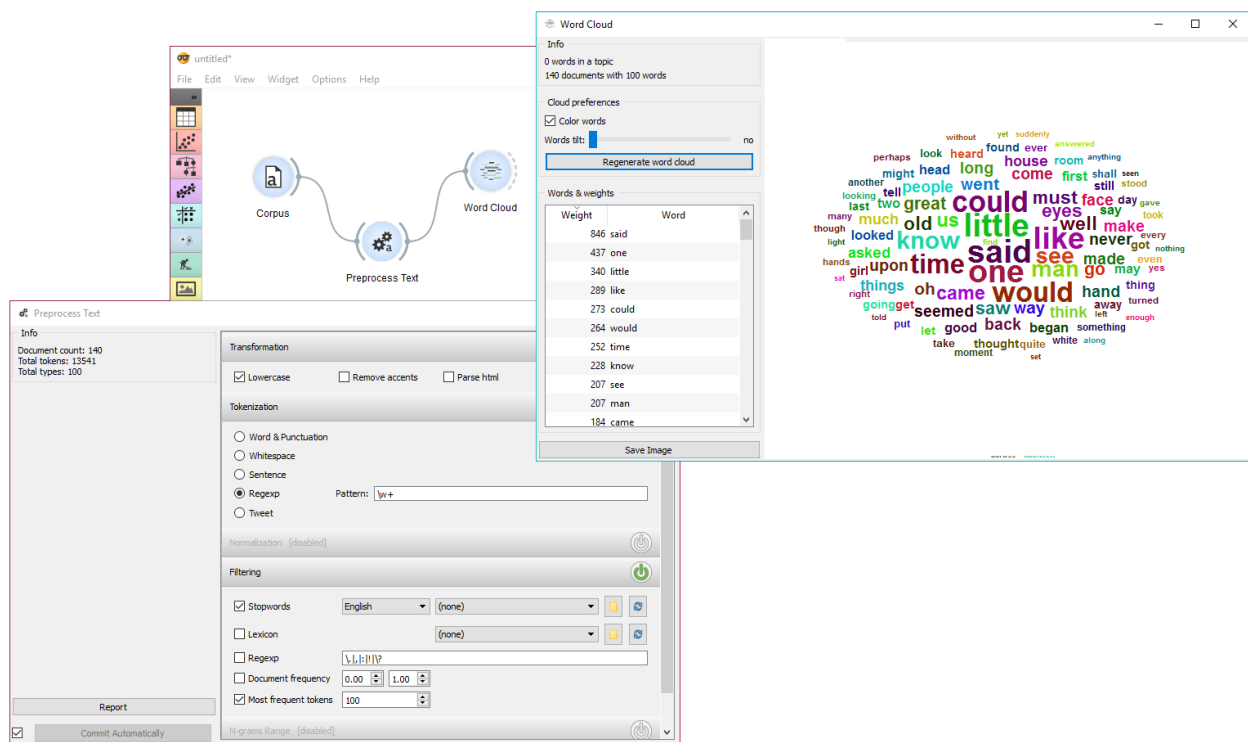
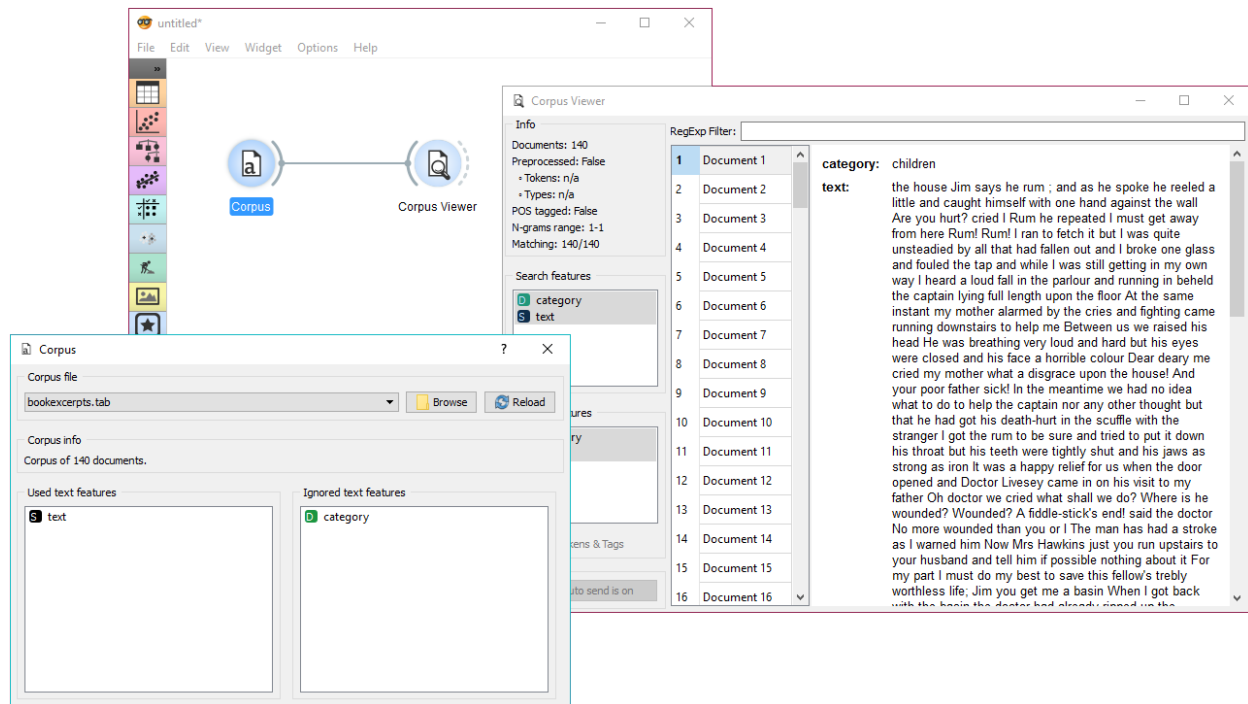
You can drag and drop features between the two boxes and also change the order in which they appear.



1.3 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 *bookexcerpts.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.



1.3. Example

NY Times



Loads data from the New York Times' [Article Search API](#).

2.1 Signals

Inputs:

- (None)

Outputs:

- **Corpus**
A *Corpus* instance.

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.

NY Times (4%, ETA: 0:... ? X

Article API Key ❶

Query ❷

slovenia

From: 2015-10-11 To: 2016-10-10

Text includes ❸

☒ Headline ☐ URL

☒ Abstract ☐ Locations

☐ Snippet ☐ Persons

☐ Lead Paragraph ☐ Organizations

☐ Subject Keywords ☐ Creative Works

Output ❹

Articles: 20/410

Report ❺ Stop ❻

New York Times API key ? X

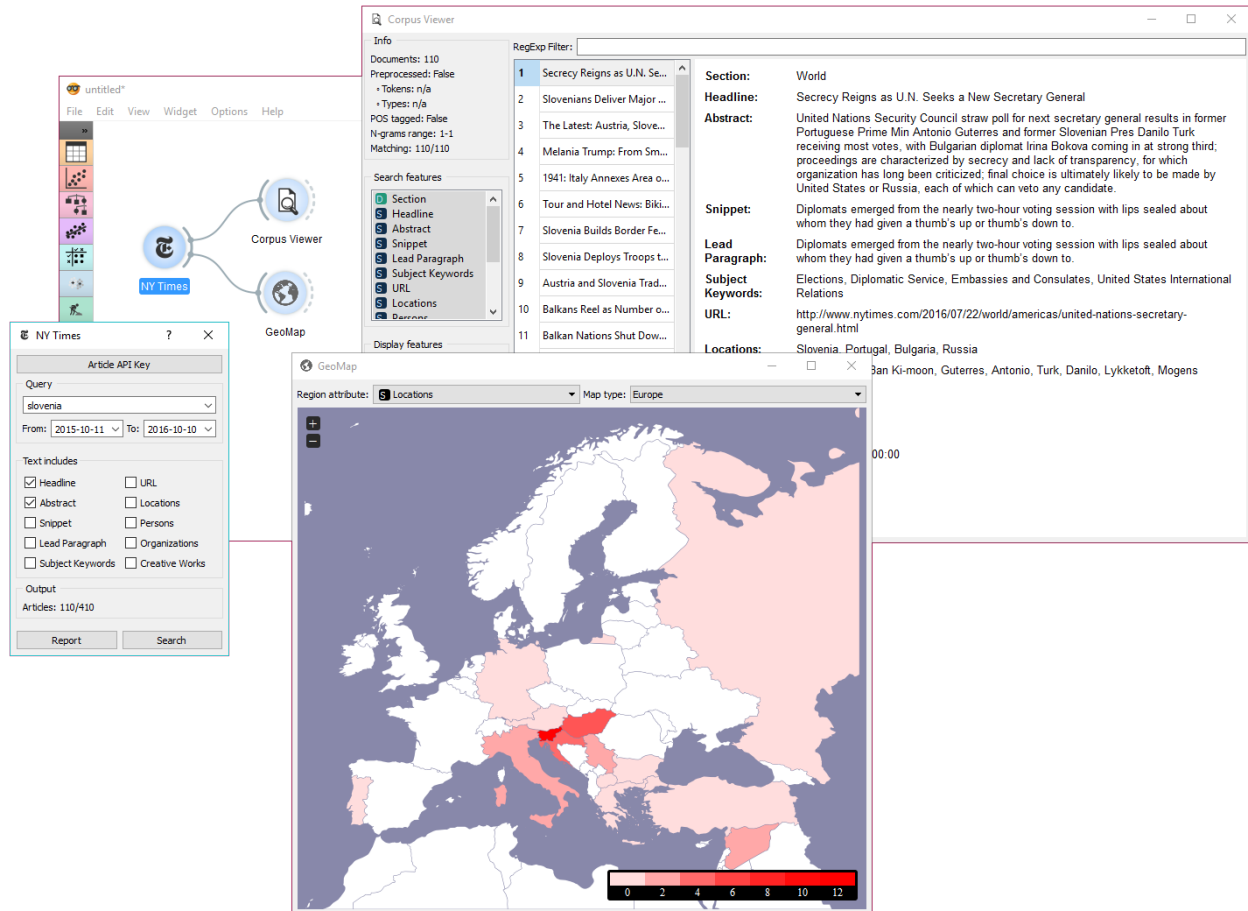
Key:

OK

6. Run or stop the query.

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**. Unsurprisingly, 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.

Twitter



Fetching data from [The Twitter Search API](#).

3.1 Signals

Inputs:

- (None)

Outputs:


- **Corpus**

A *Corpus* instance.

3.2 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.

 Twitter ? ×

Twitter API Key **1**

Query **2**

Query word list:

Multiple lines are automatically joined with OR.

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

- ### 3.3 Examples

[illegible]

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

Wikipedia



Fetching data from [MediaWiki RESTful web service API](#).

4.1 Signals

Inputs:

- (None)

Outputs:

- **Corpus**

A *Corpus* instance.

4.2 Description

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

1. Query parameters:

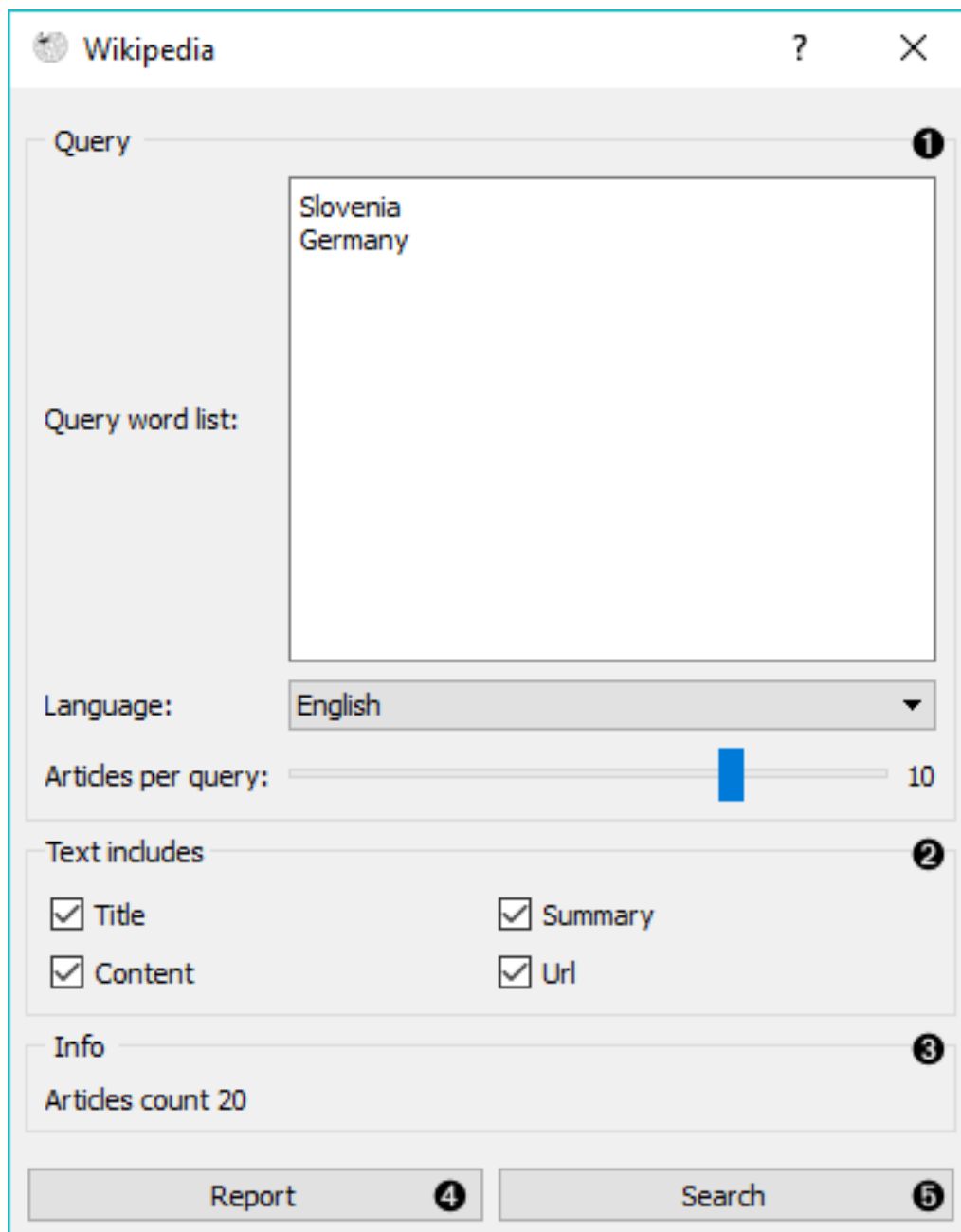
- Query word list, where each query is listed in a new line.
- Language of the query. English is set by default.
- Number of articles to retrieve per query (range 1-25). Please note that querying is done recursively and that disambiguations are also retrieved, sometimes resulting in a larger number of 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.



Pubmed



Fetch data from [PubMed](#) journals.

5.1 Signals

Inputs:

- (None)

Outputs:

- **Corpus**
A *Corpus* instance.

5.2 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.

Pubmed

Email: ①

Regular search Advanced search ②

Author:

From: to:

Query:

Number of retrievable records for this search query: 1482

③

Text includes ④

- ☒ Authors
- ☒ Article title
- ☒ Mesh headings
- ☒ Abstract
- ☒ URL

Retrieve records from 1482.

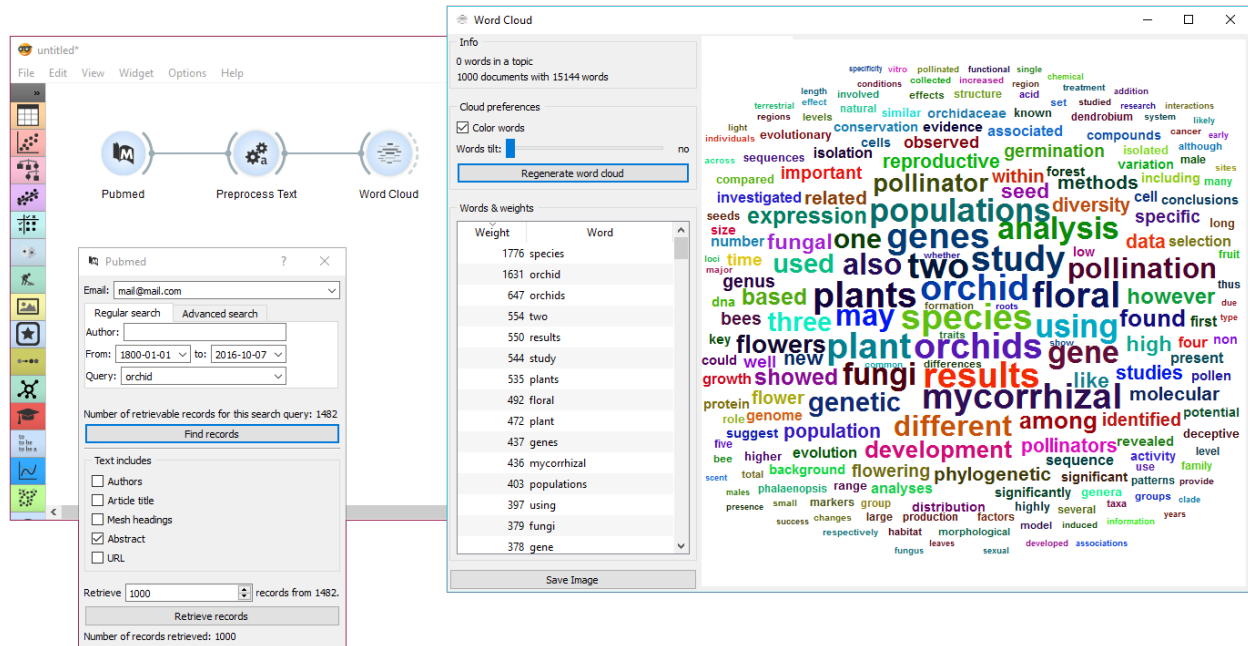
⑤

Number of records retrieved: 1000

- 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.

5.3 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 stopwords and words shorter than 3 characters (regex `\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](#).

Corpus Viewer



Displays corpus content.

6.1 Signals

Inputs:

- **Data**
Data instance.

Outputs:

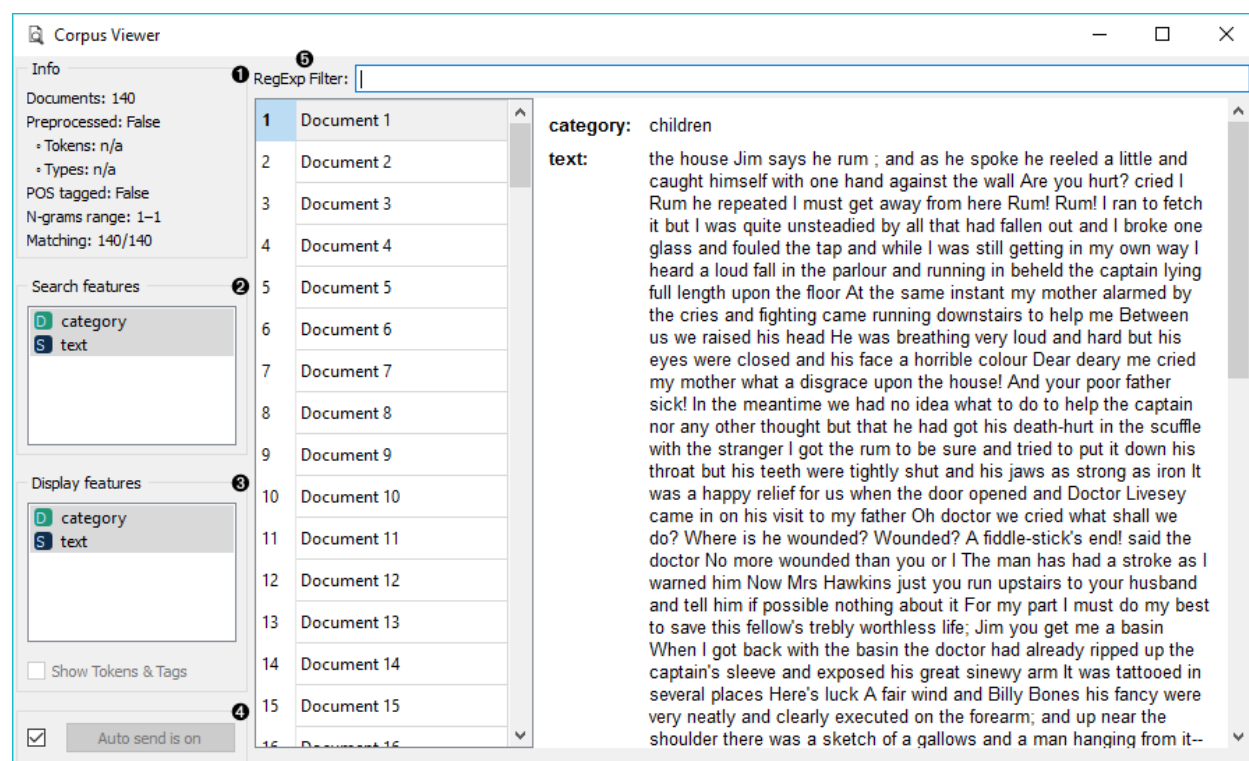
- **Corpus**
A *Corpus* instance.

6.2 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*.

6.3 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.

Preprocess Text



Preprocesses corpus with selected methods.

7.1 Signals

Inputs:

- **Corpus**
Corpus instance.

Outputs:

- **Corpus**
Preprocessed corpus.

7.2 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.

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:
☐ Tweet

Normalization

☒ Porter Stemmer
☐ Snowball Stemmer Language:
☐ WordNet Lemmatizer

Filtering

☒ Stopwords
☐ Lexicon
☐ Regexp
☐ Document frequency
☐ Most frequent tokens

N-grams Range

Range:

POS Tagger

☒ Averaged Perceptron Tagger
☐ Treebank POS Tagger (MaxEnt)
☐ Stanford POS Tagger

☒ Commit Automatically

3. **Tokenization** is the method of breaking the text into smaller components (words, sentences, bigrams).

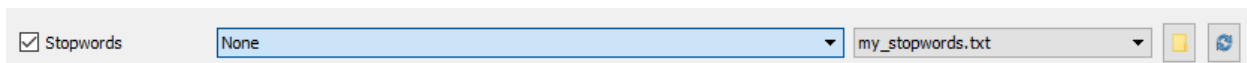
- **Word & Punctuation** will split the text by words and keep punctuation symbols. This example. → (This), (example), (.)
- **Whitespace** will split the text by whitespace only. This example. → (This), (example.)
- **Sentence** will split the text by fullstop, retaining only full sentences. This example. Another example. → (This example.), (Another example.)
- **Regex** 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 symbols. 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 language

- **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.
 - **Regex** 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.

7.3 Useful Regular Expressions

Here are some useful regular expressions for quick filtering:

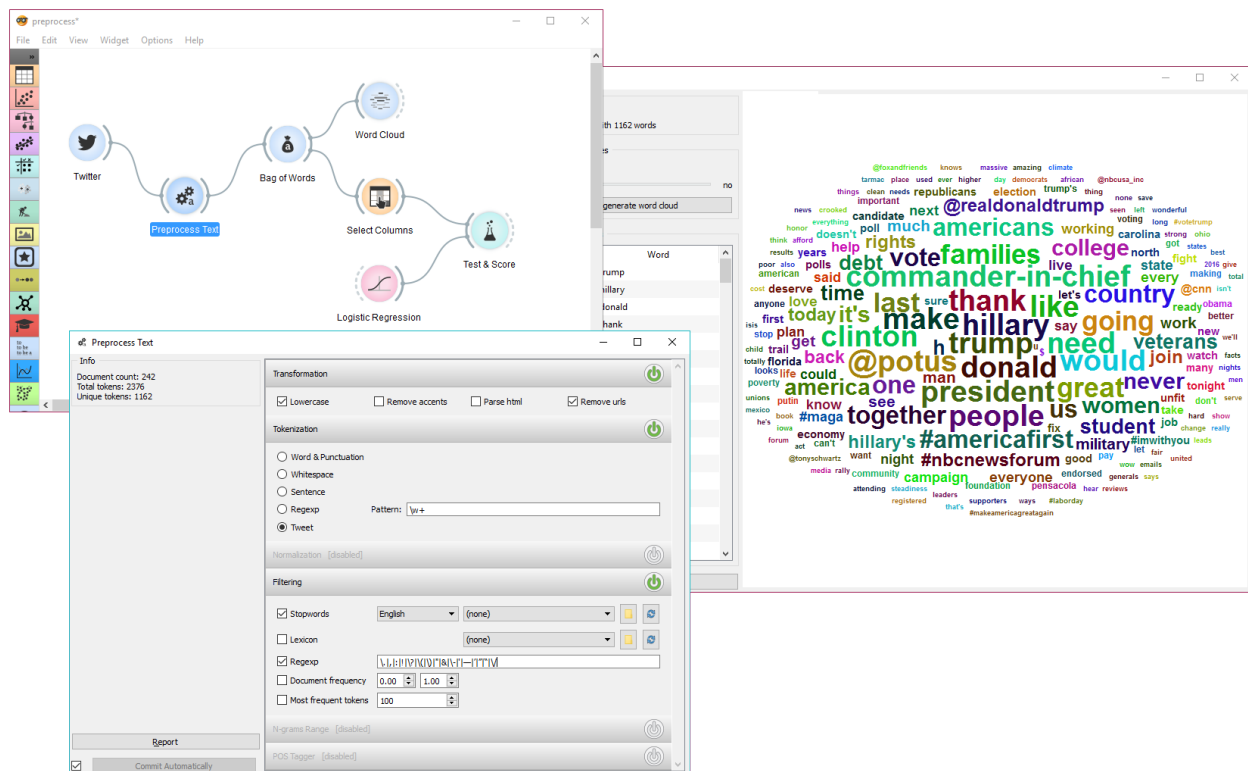
<code>\bword\b</code>	matches exact word
<code>\w+</code>	matches only words, no punctuation
<code>\b(B b)\w+\b</code>	matches words beginning with the letter b
<code>\w{4,}</code>	matches words that are longer than 4 characters
<code>\b\w+(Y y)\b</code>	matches words ending with the letter y

7.4 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 stopwords 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.



Bag of Words



Generates a bag of words from the input corpus.

8.1 Signals

Inputs:

- **Corpus**
Corpus instance.

Outputs:

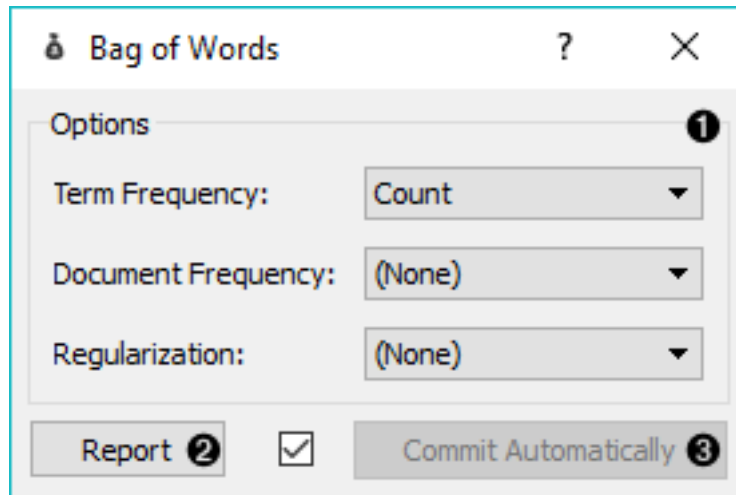
- **Corpus**
Corpus with bag of words.

8.2 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 occurrences 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.

- **Regularization:**

- (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*.

8.3 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**.

The screenshot displays the Orange3 Text Mining workflow and the Data Table widget output.

Workflow: Corpus → Bag of Words → Data Table

Bag of Words Widget Options:

- Term Frequency: Count
- Document Frequency: (None)
- Regularization: (None)
- Report: ☒
- Commit Automatically: ☐

Data Table Widget Info:

- 140 instances
- 10865 features (sparse, density 0.05%)
- Discrete class with 2 values (no missing values)
- 1 meta attribute (no missing values)

Data Table Widget Variables:

- ☒ Show variable labels (if present)
- ☐ Visualize continuous values
- ☒ Color by instance classes

Data Table Widget Selection:

- ☒ Select full rows

Data Table Widget Output:

hidden	category	text	True	{...}
1	children	the house Jim s...	broke=1.000, by=4.000, trebly=1.000, basin=3.000, executed=1.000, picture=1.000, se...	
2	children	has lived rough...	golden=1.000, carried=1.000, bar=2.000, confessions=1.000, air=1.000, again=5.000, r...	
3	children	Now boy he sai...	gathering=1.000, letter=1.000, bring=1.000, resolved=1.000, payment=1.000, peculiar...	
4	children	thanks to you b...	despair=1.000, thanks=1.000, finely=1.000, swift=1.000, terrors=1.000, rogues=1.000, ...	
5	children	the empty ches...	curiosity=1.000, drag=1.000, retreat=1.000, beyond=1.000, brief=1.000, cowardice=1....	
6	children	stood irresolute...	dance=3.000, furious=1.000, such=1.000, matter=1.000, fools=1.000, nearest=1.000, p...	
7	children	WE rode hard al...	son=1.000, rascal=1.000, smoke=1.000, proud=1.000, hearty=1.000, villains=1.000, co...	
8	children	same as the tatt...	entry=1.000, roll=1.000, cache=1.000, blank=1.000, rank=1.000, manned=1.000, houn...	
9	children	IT was longer t...	transparent=1.000, housekeeper=1.000, explored=1.000, fancies=1.000, plans=1.000, ...	
10	children	treasure Long J...	dream=1.000, picked=1.000, telescope=1.000, substance=1.000, unearthed=1.000, ro...	
11	children	We are so grate...	whatever=1.000, favor=1.000, therefore=1.000, beam=1.000, dismay=1.000, dwelt=1....	
12	children	I am told said t...	loudly=1.000, frock=1.000, bread=2.000, brook=1.000, around=1.000, grieve=1.000, g...	
13	children	to find the one ...	watched=2.000, chin=1.000, merrily=1.000, earnestly=1.000, stalks=1.000, stop=1.000...	
14	children	take away the p...	unfriendly=1.000, nest=1.000, bites=1.000, truly=1.000, partv=1.000, lonesome=1.000...	

The screenshot displays the Orange3 Text Mining workflow and its results. The workflow consists of the following widgets: Corpus, Preprocess Text, Bag of Words, SVM, Test & Score, Confusion Matrix, and Corpus Viewer.

Bag of Words Widget Options:

- Term Frequency: Count
- Document Frequency: IDF
- Regularization: (None)
- Report: ☒
- Commit Automatically: ☐

Test & Score Widget Evaluation Results:

Method	AUC	CA	F1	Precision	Recall
SVM	0.971	0.971	0.971	1.000	0.943

Confusion Matrix Widget Results:

Actual \ Predicted	adult	children	Σ
	adult	70	0
children	4	66	70
Σ	74	66	140

Corpus Viewer Widget Information:

- Documents: 4
- Preprocessed: False
- + Tokens: n/a
- + Types: n/a
- POS tagged: False
- N-grams range: 1-1
- Matching: 4/4

Corpus Viewer Search and Display Features:

- Search features: category, text, category(SVM)
- Display features: category, text, category(SVM)
- Show Tokens & Tags: ☐

Corpus Viewer Document 1 Text:

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 other's hands we advanced into the parlour.

Topic Modelling



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

9.1 Signals

Inputs:

- **Corpus**

Corpus instance.

Outputs:

- **Data**

Data with topic weights appended.

- **Topics**

Selected topics with word weights.

9.2 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*.

9.3 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.

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**.

The collage displays several Orange3 Text Mining widgets and their results:

- Topic Modelling:** Shows a workflow from Corpus to Preprocess Text to Word Cloud to Corpus Viewer. The widget settings include Latent Semantic Indexing with 10 topics. The output shows topic keywords for 10 topics, such as "said, one, little, like, could, would, time, know, see, man" for topic 1.
- Corpus Viewer:** Displays a list of 10 documents. The first document is titled "category: children" and contains a paragraph of text.
- Word Cloud:** Shows a word cloud generated from the corpus. The most prominent words are "said", "one", "little", "like", "could", "would", "time", "know", "see", "man", "polly", "said", "phronsie", "princess", "gerald", "clem", "jimmy", "oh", "little", "trot", "n", "cap", "bill", "water", "said", "mermaids", "sea", "polly", "asked", "sara", "miss", "minchin", "little", "george", "polly", "child", "chloe", "looked", "french", "dorothy", "scarecrow", "lion", "tin", "woodman", "toto", "witch", "oz", "asked", "road", "aunt", "george", "gerald", "chloe", "princess", "jimmy", "de", "t", "mas", "ye", "aulus", "lygia", "thou", "thee", "pomponia", "petronius", "house", "cser", "george", "vinic", "m", "little", "us", "lion", "story", "princess", "people", "witch", "mary", "man", "may", "us", "ladies", "mr", "young", "miss", "men", "doctor", "wilde", "mother".
- Word Enrichment:** Shows a table of words and their p-values and FDR values. The table is filtered by p-value < 0.0100 and FDR < 0.2000. The words listed are: isis, terrorism, germany, drops, whose, comic, investigation, poem, turkey, us, resort, npr, uk, india, law, france, talks, amp, german, db, new, eu.
- Select Rows:** Shows a table of rows selected based on the condition "Topic1 (german) is greater than 0.9". The table has 10 rows and 10 columns.
- Word Cloud (Enriched):** Shows a word cloud generated from the selected rows. The most prominent words are "isis", "terrorism", "germany", "drops", "whose", "comic", "investigation", "poem", "turkey", "us", "resort", "npr", "uk", "india", "law", "france", "talks", "amp", "german", "db", "new", "eu".

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.

Word Enrichment



Word enrichment analysis for selected documents.

10.1 Signals

Inputs:

- **Data**
Corpus instance.
- **Selected Data**
Selected instances from corpus.

Outputs:

- (None)

10.2 Description

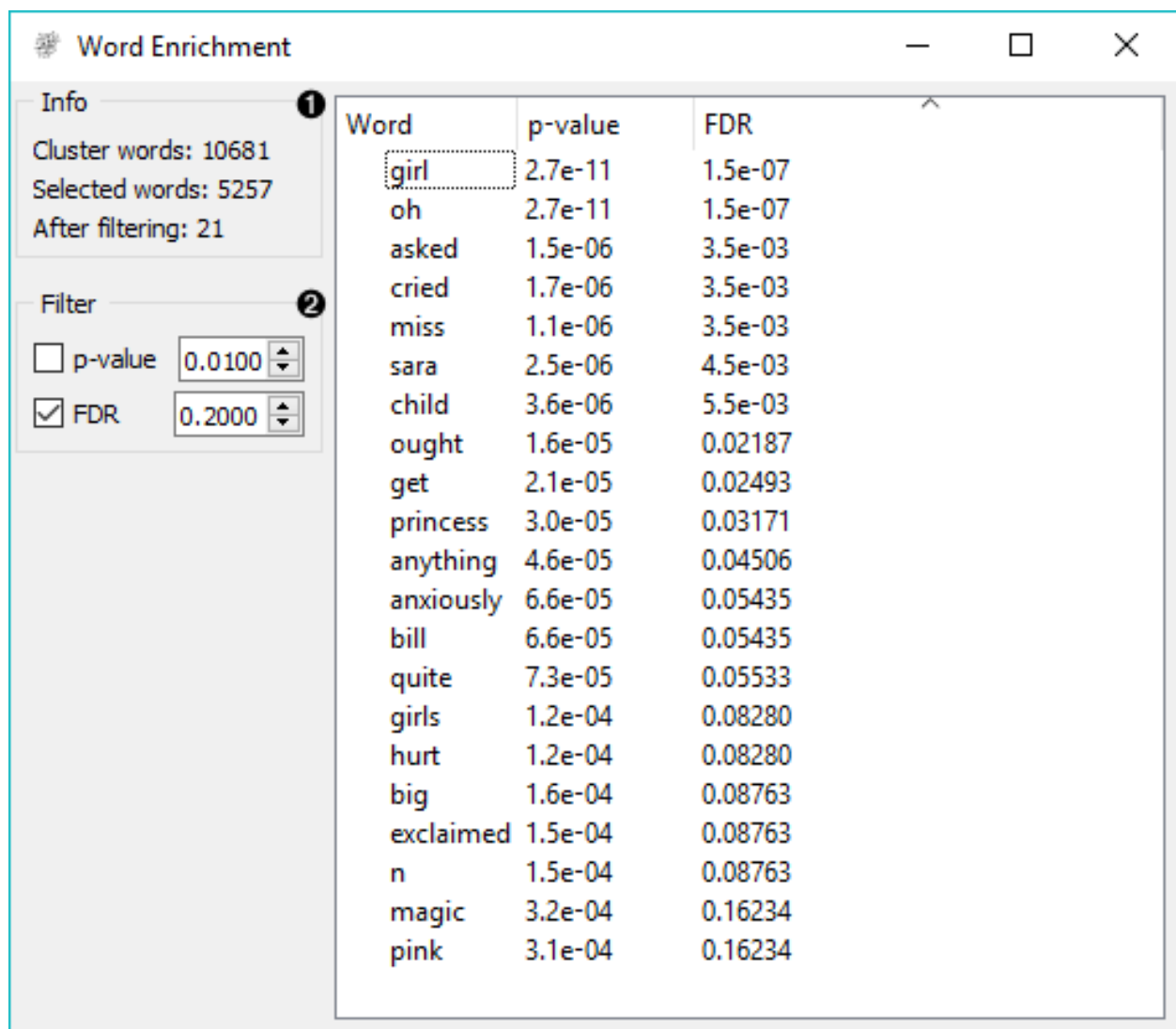
Word Enrichment displays a list of words with lower p-values (higher significance) for a selected subset compared to the entire corpus. Lower p-value indicates a higher likelihood that the word is significant for the selected subset (not randomly occurring in a text). FDR (False Discovery Rate) is linked 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



Word Enrichment

Info

Cluster words: 10681
Selected words: 5257
After filtering: 21

Filter

☐ p-value 0.0100

☒ FDR 0.2000

Word	p-value	FDR
girl	2.7e-11	1.5e-07
oh	2.7e-11	1.5e-07
asked	1.5e-06	3.5e-03
cried	1.7e-06	3.5e-03
miss	1.1e-06	3.5e-03
sara	2.5e-06	4.5e-03
child	3.6e-06	5.5e-03
ought	1.6e-05	0.02187
get	2.1e-05	0.02493
princess	3.0e-05	0.03171
anything	4.6e-05	0.04506
anxiously	6.6e-05	0.05435
bill	6.6e-05	0.05435
quite	7.3e-05	0.05533
girls	1.2e-04	0.08280
hurt	1.2e-04	0.08280
big	1.6e-04	0.08763
exclaimed	1.5e-04	0.08763
n	1.5e-04	0.08763
magic	3.2e-04	0.16234
pink	3.1e-04	0.16234

- false discovery rate (FDR)

10.3 Example

In the example below, we've 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.

The screenshot displays an Orange3 workflow with the following widgets and settings:

- Twitter**: Query word list: realDonaldTrump, HillaryClinton; Search by: Author; Date: since 2016-09-25 until 2016-10-05; Language: English; Max tweets: 100; Text includes: Content.
- Preprocess Text**: Connected to Twitter.
- Bag of Words**: Connected to Preprocess Text.
- Corpus Viewer**: Connected to Bag of Words. Info: Documents: 354, Preprocessed: True, Tokens: 4309, Types: 1632, POS tagged: False, N-grams range: 1-1, Matching: 98/354. Search features: Author, Content, Date, Language, Location. Display features: Author, Content, Date, Language, Location. Show Tokens & Tags: unchecked. Auto send is on: checked.
- Word Enrichment**: Connected to Corpus Viewer. Info: Cluster words: 1632, Selected words: 614, After filtering: 15. Filter: p-value 0.0100, FDR 0.2000. Table of enriched words:

Word	p-value	FDR
maga	4.6e-10	3.8e-07
thank	3.4e-10	3.8e-07
americafirst	1.9e-06	1.0e-03
clinton	4.4e-06	1.8e-03
join	8.1e-06	2.7e-03
debates2016	2.8e-05	6.5e-03
hillaryclinton	2.8e-05	6.5e-03
bad	1.1e-04	0.01576
crooked	1.1e-04	0.01576
poll	1.1e-04	0.01576
tickets	1.1e-04	0.01576
movement	1.9e-04	0.02622
great	9.6e-04	0.12025
supporters	1.5e-03	0.16408
wow	1.5e-03	0.16408

The **Corpus Viewer** also shows a list of tweets filtered by the regex "Trump":

- Wow, did you just ...
- Join the MOVEME...
- Thank you ARIZON...
- My childcare plan ...
- I will be watching t...
- Join me in Reno, N...
- Join me in Reno, N...
- Thank you Colorad...
- We must bring the ...
- Join me in Henders...
- Just announced th...
- Melania and I exten...
- Bernie should pull ...
- "@trumplican2016...
- I have created tens ...
- I know our comple...

The **Word Enrichment** widget also displays the following details for the selected tweet:

Author: @realDonaldTrump
Content: Wow, did you just hear Bill Clinton's statement on how bad ObamaCare is. Hillary not happy. As I have been saying, REPEAL AND REPLACE!
Date: 2016-10-04 21:55:55

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.

Word Cloud



Generates a word cloud from corpus.

11.1 Signals

Inputs:

- **Topic**
Selected topic.
- **Corpus**
A *Corpus* instance.

Outputs:

- **Corpus**
Documents that match the selection.

11.2 Description

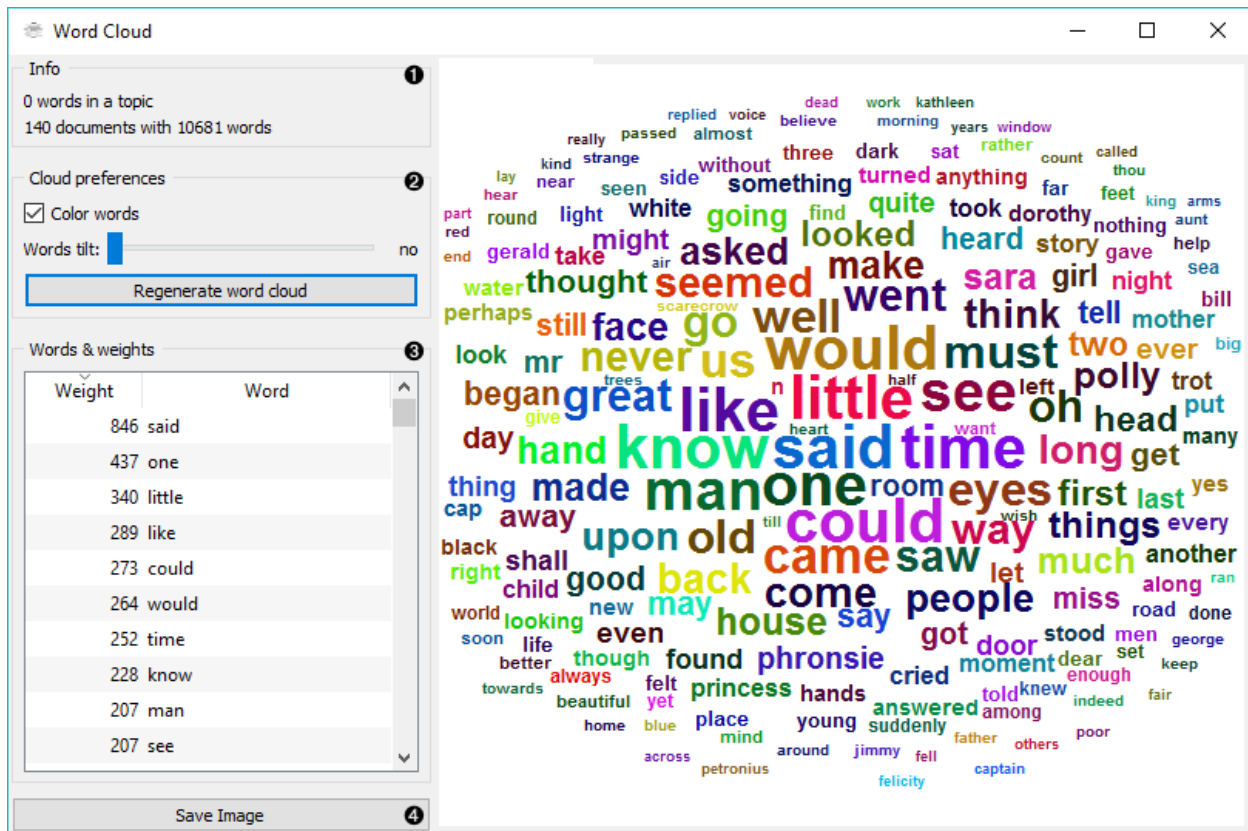
Word Cloud displays tokens in the corpus, their size denoting the frequency of the word in corpus. Words are listed by their frequency (weight) in the widget. The widget outputs documents, containing selected tokens from the word cloud.

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.



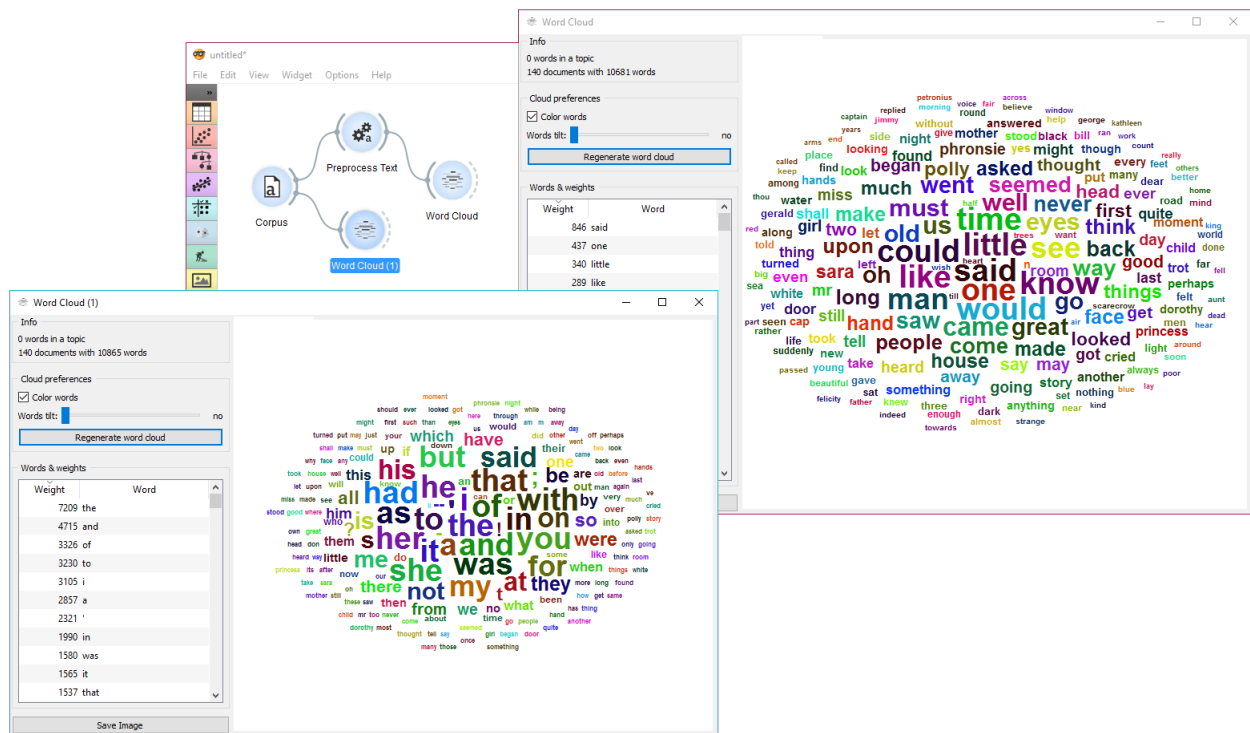
- *Word tilt* adjust the tilt of words. The current state of tilt is displayed next to the slider ('no' is the default).
 - *Regenerate word cloud* plot the cloud anew.
3. *Words & weights* displays a sorted list of words (tokens) by their frequency in the corpus or topic. Clicking on a word will select 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.

11.3 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.





Displays geographic distribution of data.

12.1 Signals

Inputs:

- **Data**
Data set.

Outputs:

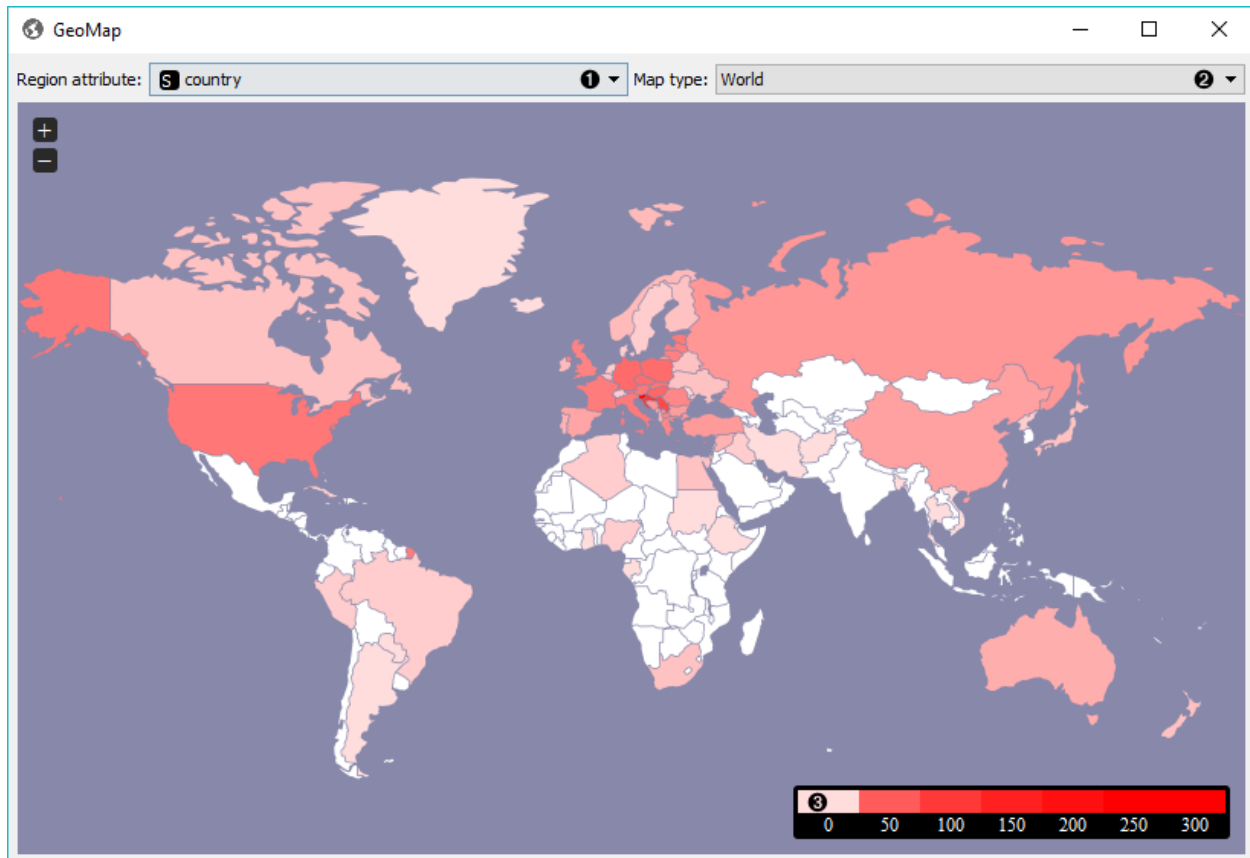
- **Corpus**
A *Corpus* instance.

12.2 Description

GeoMap widget shows geolocations from textual (string) data. It finds mentions of geographic names (countries and capitals) and displays distributions (frequency of mentions) 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.



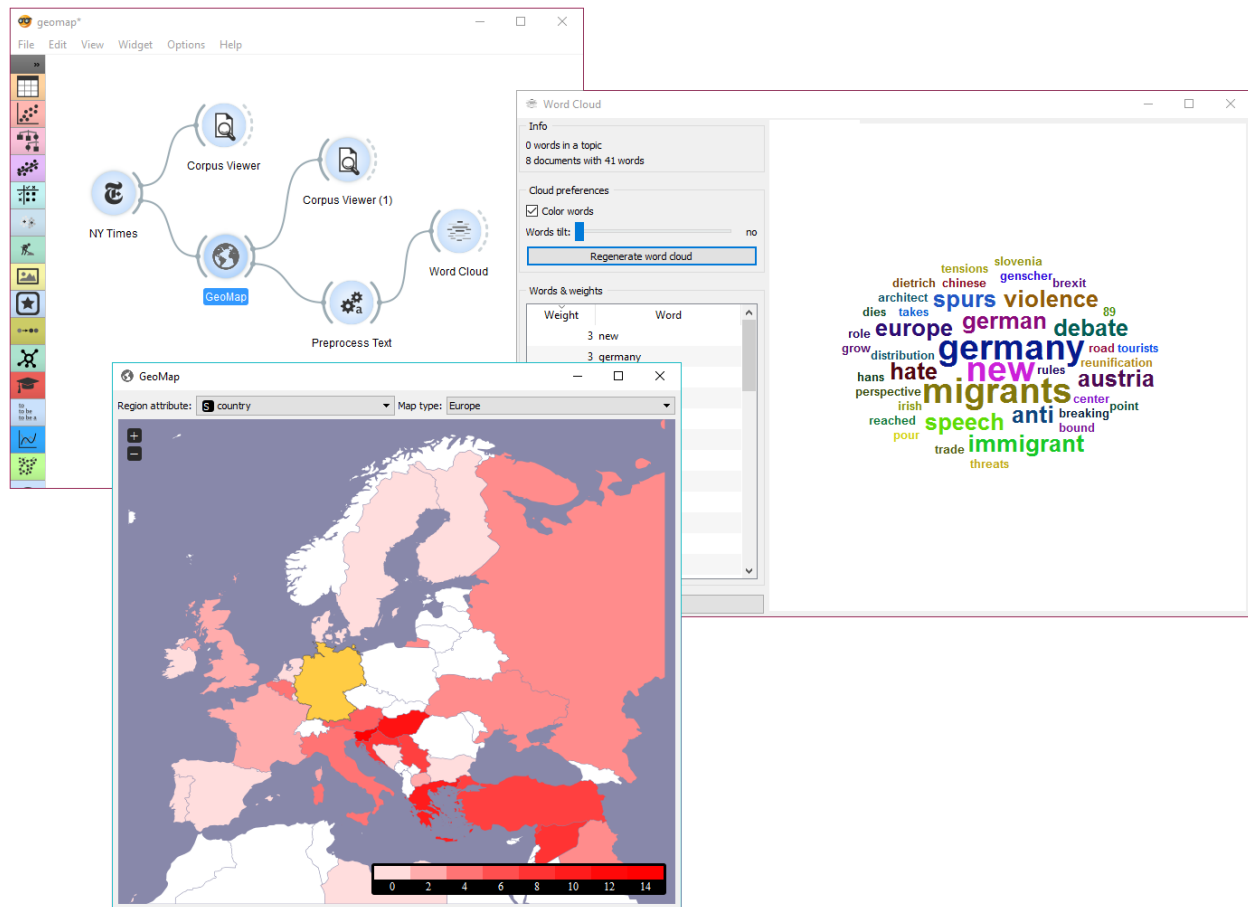
12.3 Example

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](#).

Then we sent the data to **GeoMap** to see distributions 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](#).



Corpus

Preprocessor

Twitter

New York Times

Wikipedia

Topic Modeling

Tag

Indices and tables

- `genindex`
- `modindex`
- `search`