

---

# Orange3 Text Mining Documentation

*Release*

**Biolab**

**Feb 08, 2017**



<b>1</b>	<b>Corpus</b>	<b>1</b>
<b>2</b>	<b>NY Times</b>	<b>5</b>
<b>3</b>	<b>The Guardian</b>	<b>9</b>
<b>4</b>	<b>Twitter</b>	<b>11</b>
<b>5</b>	<b>Wikipedia</b>	<b>15</b>
<b>6</b>	<b>Pubmed</b>	<b>19</b>
<b>7</b>	<b>Corpus Viewer</b>	<b>23</b>
<b>8</b>	<b>Preprocess Text</b>	<b>27</b>
<b>9</b>	<b>Bag of Words</b>	<b>33</b>
<b>10</b>	<b>Topic Modelling</b>	<b>37</b>
<b>11</b>	<b>Word Enrichment</b>	<b>41</b>
<b>12</b>	<b>Word Cloud</b>	<b>45</b>
<b>13</b>	<b>GeoMap</b>	<b>49</b>
<b>14</b>	<b>Corpus</b>	<b>53</b>
<b>15</b>	<b>Preprocessor</b>	<b>57</b>
<b>16</b>	<b>Twitter</b>	<b>59</b>
<b>17</b>	<b>New York Times</b>	<b>61</b>
<b>18</b>	<b>The Guardian</b>	<b>63</b>
<b>19</b>	<b>Wikipedia</b>	<b>65</b>
<b>20</b>	<b>Topic Modeling</b>	<b>67</b>

<b>21 Tag</b>	<b>69</b>
<b>22 Async Module</b>	<b>71</b>
<b>23 Indices and tables</b>	<b>73</b>
<b>Python Module Index</b>	<b>75</b>



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

## 1.1 Signals

### Inputs:

- (None)

### Outputs:

- **Corpus**

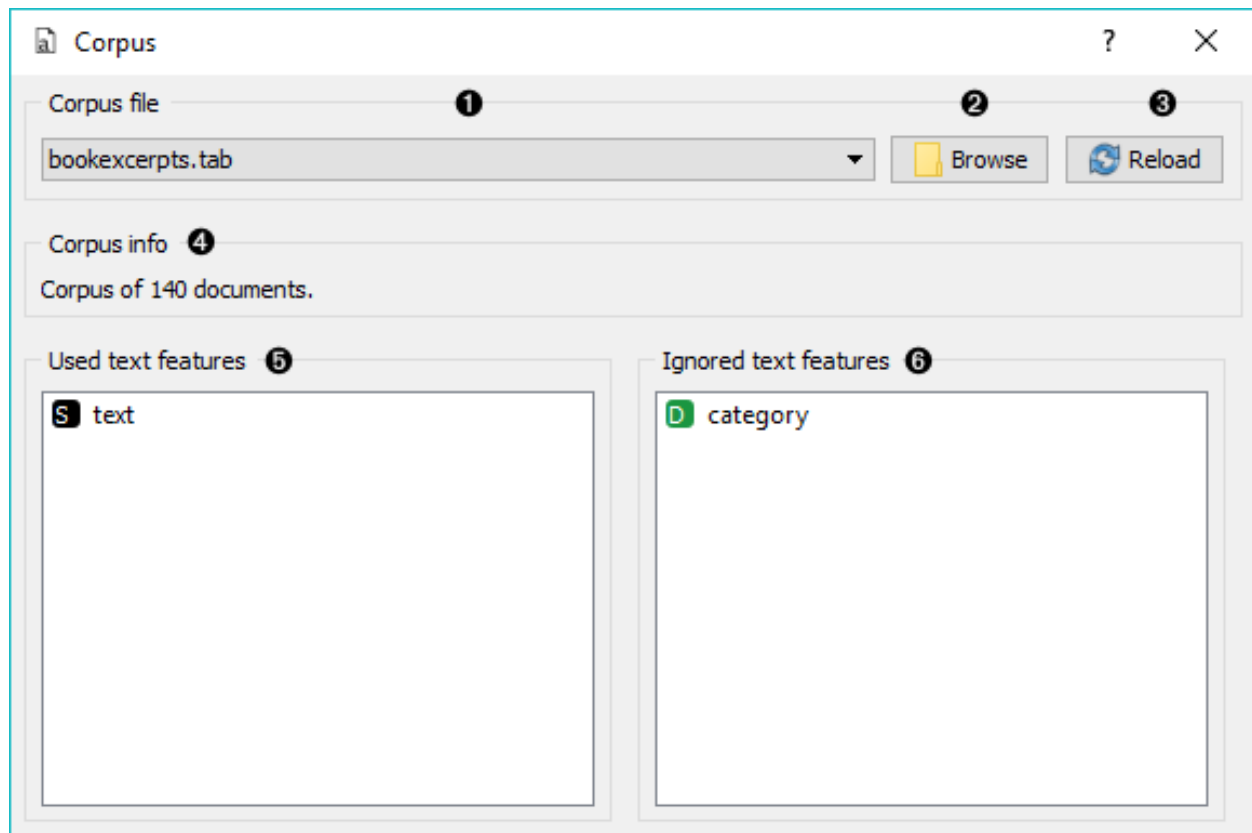
A *Corpus* instance.

## 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.



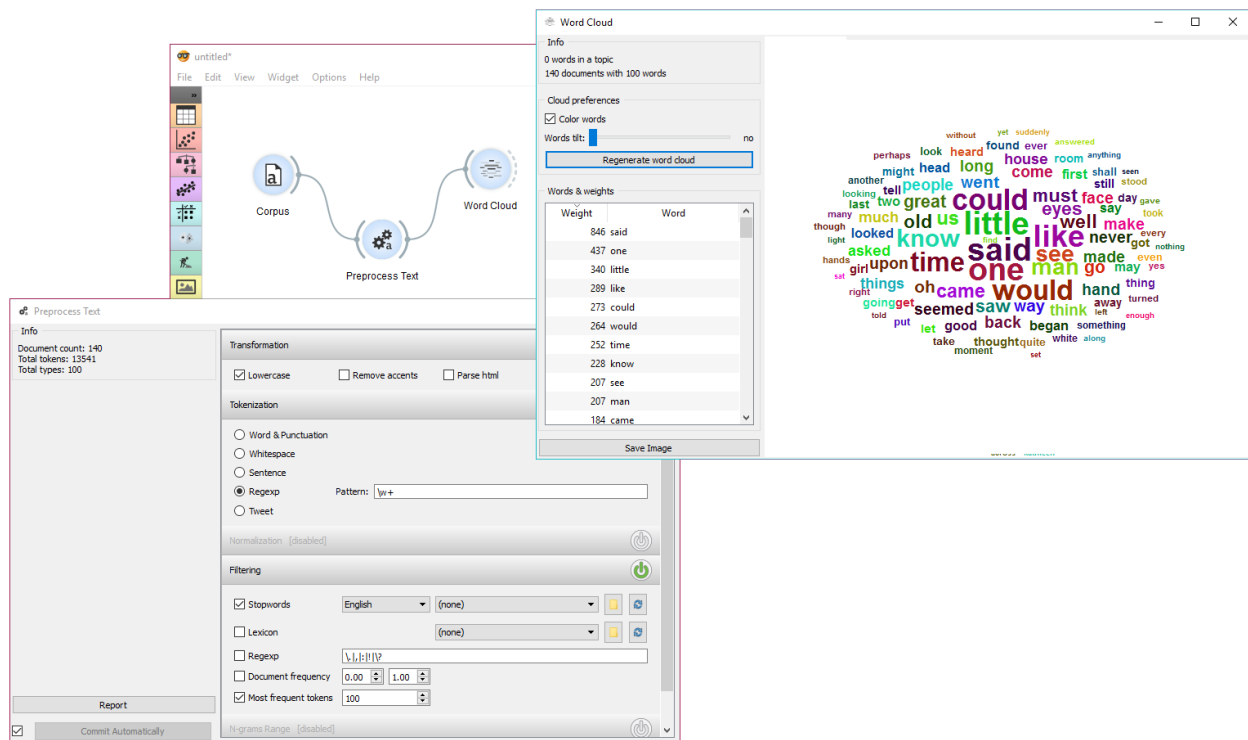
4. Information on the loaded data set.
5. Features that will be used in text analysis.
6. Features that won't be used in text analysis and serve as labels or class.

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.









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*

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

Article API Key 1

Query 2

slovenia

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

Text includes 3

☒ Headline ☐ URL

☒ Abstract ☐ Locations

☐ Snippet ☐ Persons

☐ Lead Paragraph ☐ Organizations

☐ Subject Keywords ☐ Creative Works

Output 4

Articles: 20/410

Report 5 Stop 6

New York Times API key ? X

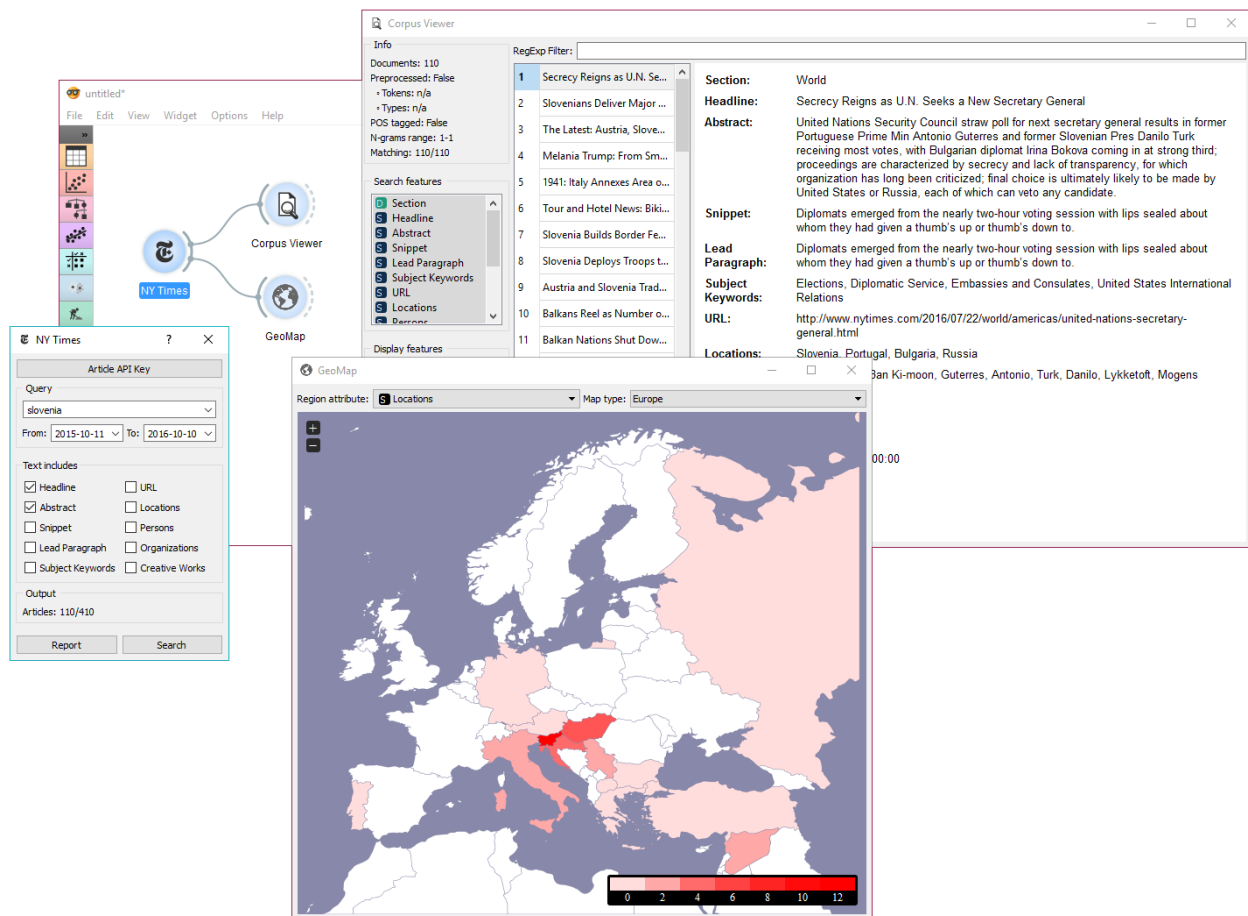
Key:

OK

- 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.

## 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.





Fetching data from [The Guardian Open Platform](#).

### 3.1 Signals

**Inputs:**

- (None)

**Outputs:**

- **Corpus**

A *Corpus* instance.

### 3.2 Description

No description for this widget yet.

### 3.3 Examples

No examples for this widget yet.





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

## 4.1 Signals

### Inputs:

- (None)

### Outputs:


- **Corpus**

A *Corpus* instance.

## 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.

 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



- ## 4.3 Examples

[illegible]





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

## 5.1 Signals

### Inputs:

- (None)

### Outputs:

- **Corpus**

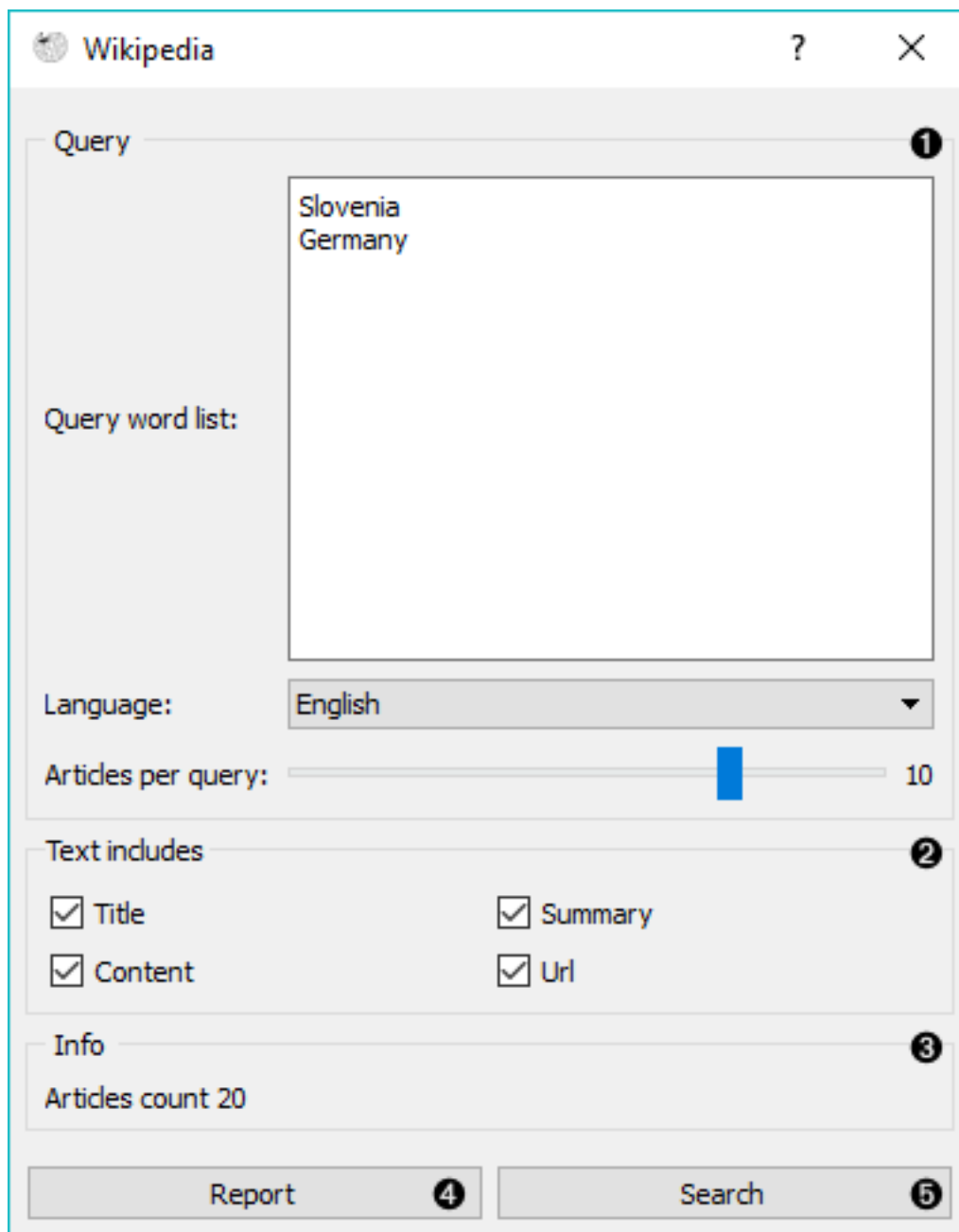
A *Corpus* instance.

## 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:

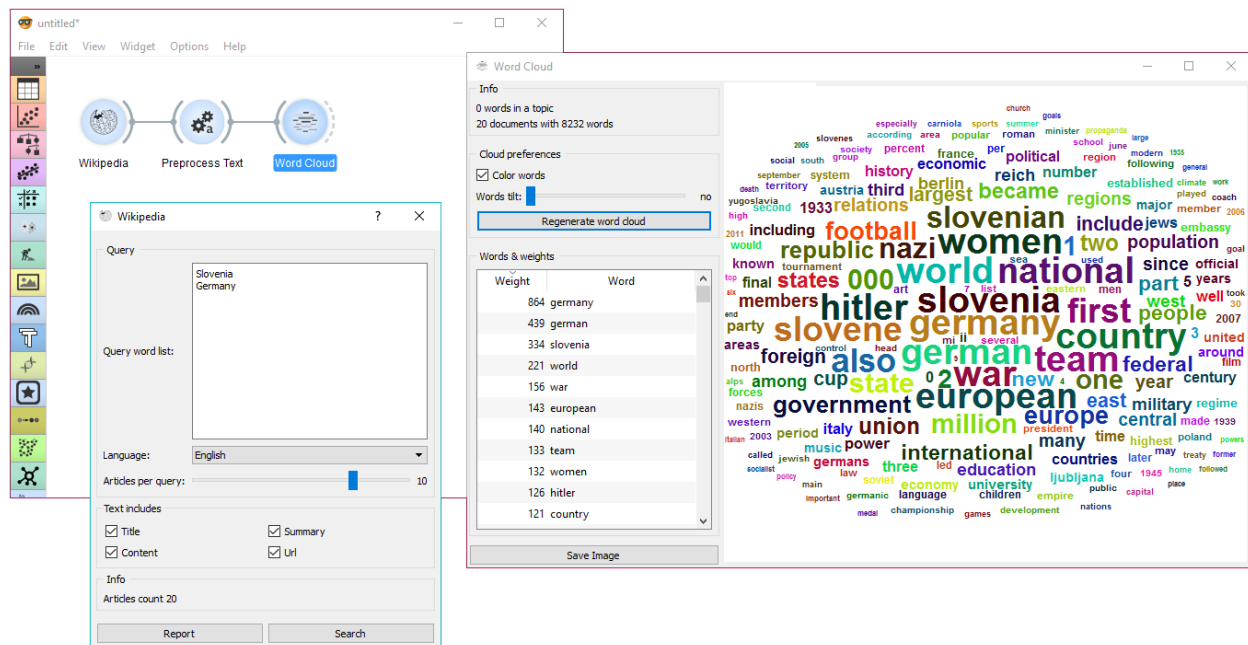
- Query word list, where each query is listed in a new line.
- Language of the query. English is set by default.
- Number of articles to retrieve per query (range 1-25). Please note that querying is done recursively and that disambiguations are also retrieved, sometimes resulting in a larger number of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

### 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.





Fetch data from [PubMed](#) journals.

## 6.1 Signals

### Inputs:

- (None)

### Outputs:

- **Corpus**

A *Corpus* instance.

## 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.

Pubmed

Email:  ❶

Regular search Advanced search ❷

Author:

From:  to:

Query:

Number of retrievable records for this search query: 1482

❸

Text includes ❹

☒ Authors

☒ Article title

☒ Mesh headings

☒ Abstract

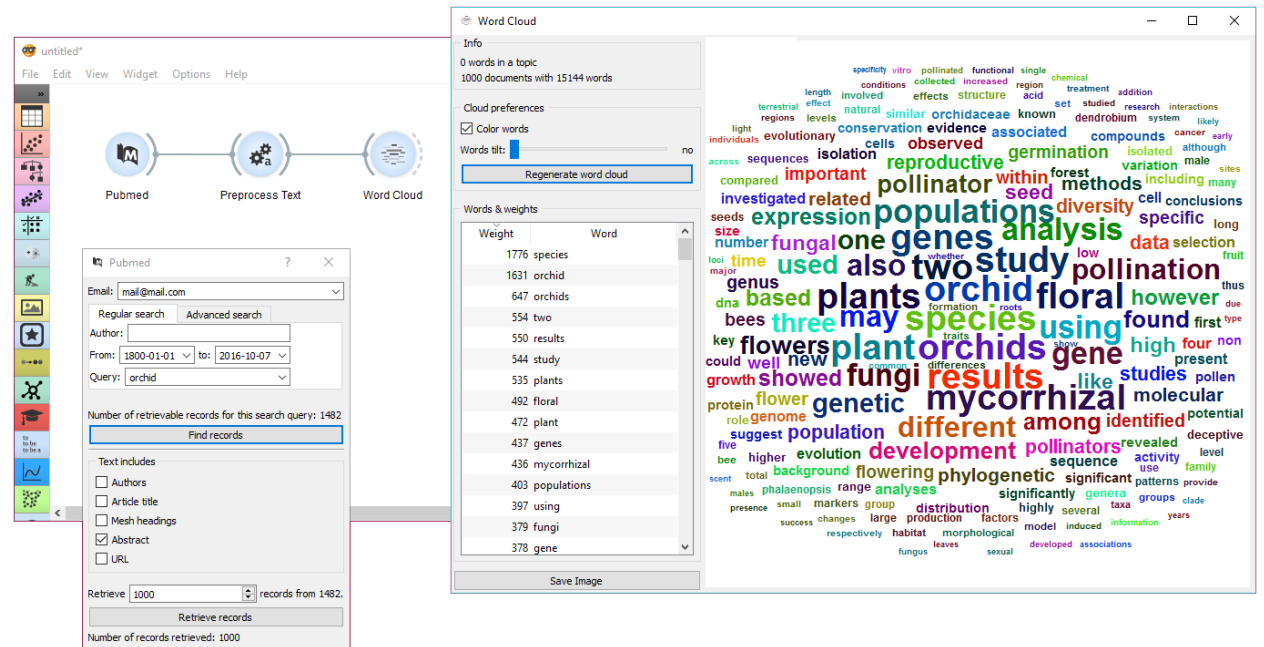
☒ URL

Retrieve  records from 1482.

❺

Number of records retrieved: 1000









Displays corpus content.

## 7.1 Signals

### Inputs:

- **Data**  
Data instance.

### Outputs:

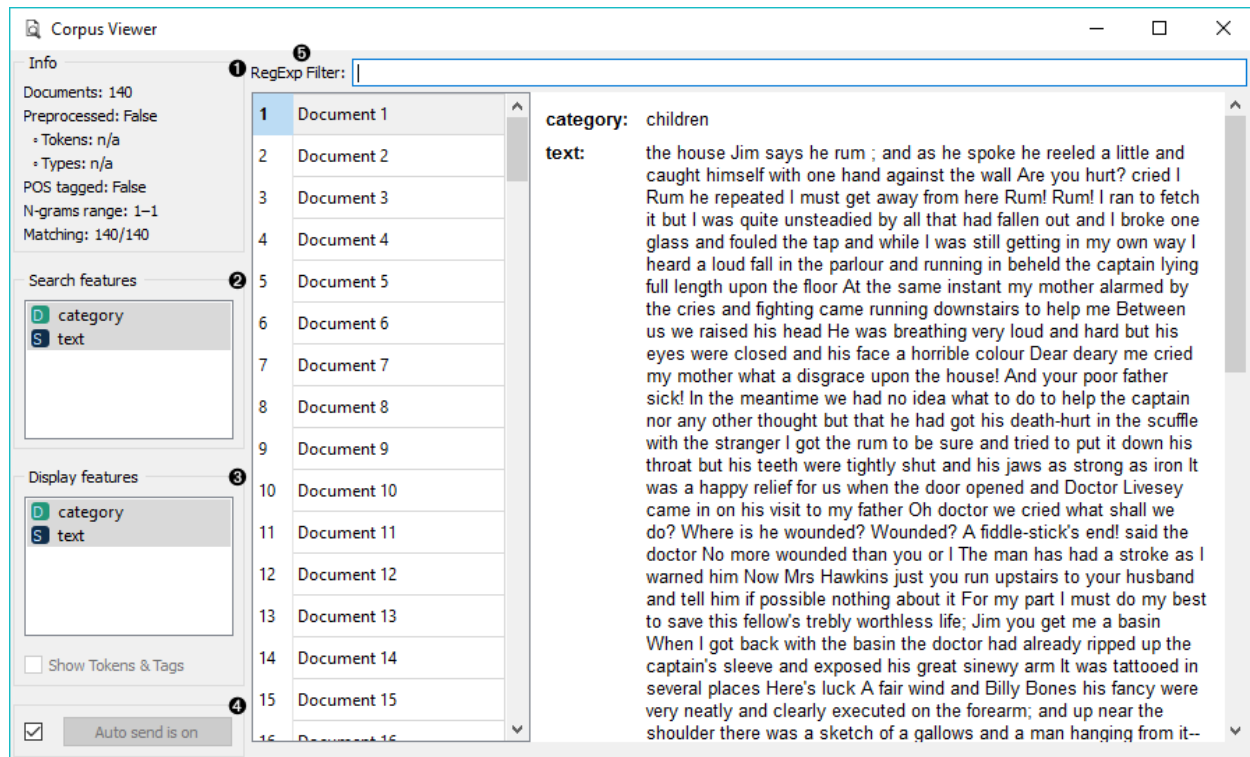
- **Corpus**  
A *Corpus* instance.

## 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*.

## 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.

The screenshot displays the Orange3 Text Mining workflow and its results. The main workflow consists of a **Corpus** widget connected to a **Preprocess Text** widget, which is then connected to a **Corpus Viewer** widget.

The **Preprocess Text** widget's configuration is as follows:

- Info:** Document count: 140, Total tokens: 60576, Unique tokens: 10681.
- Transformation:** ☒ Lowercase, ☐ Remove accents.
- Tokenization:** ☐ Word & Punctuation, ☐ Whitespace, ☐ Sentence, ☒ Regexp (Pattern: `\w+`), ☐ Tweet.
- Normalization:** [disabled]
- Filtering:**
  - ☒ Stopwords (English, (none))
  - ☐ Lexicon ((none))
  - ☐ Regexp (`\d,|:|!|?|`)
  - ☐ Document frequency (0.00 to 1.00)
  - ☐ Most frequent tokens (100)
- N-grams Range:** Range: 1 to 2.
- POS Tagger:**
  - ☒ Averaged Perceptron Tagger
  - ☐ Treebank POS Tagger (MaxEnt)
  - ☐ Stanford POS Tagger ((none))

The **Corpus Viewer** widget shows a list of documents (1-13) and a **RegExp Filter** set to `\b\w\b`. The **Search features** and **Display features** are both set to `category` and `text`. The **Show Tokens & Tags** checkbox is checked, and **Auto send** is on.

The **Tokens & Tags** output shows the following text and its corresponding POS tags:

the moonlight head and shoulders and addressed the blind beggar on the road below him Pew he cried they've been before us Someone's turned the chest out alow and alot is it there? roared Pew The money's there The blind man cursed the money Flint's fat I mean he cried We don't see it here nohow returned the man Here you below there is it on **Bill**? cried the blind man again At that another fellow probably him who had remained below to search the captain's body came to the door of the inn **Bill**'s been overhauled a'ready said he; nothin' left It's these people of the inn—it's that boy I wish I had put his eyes out! cried the blind man Pew There were no time ago—they had the door bolted when I tried it Scatter lads and find 'em Sure enough they left their glim here said the fellow from the window Scatter and find 'em! Rout the house out! reiterated Pew striking with his stick upon the road

The **Tokens & Tags** output shows the following tokens and their corresponding POS tags:

empty_JJ	chest_JJS	next_JJ	opened_JJ	door_NN	full_JJ	retreat_NN
started_VBD	moment_NN	soon_RB	fog_RB	rapidly_RB	dispersing_VBG	
already_RB	moon_RB	shone_JJ	quite_RB	clear_JJ	high_JJ	ground_NN
either_DT	side_NN	exact_JJ	bottom_NN	dell_NN	round_NN	tavern_JJ
door_NN	thin_JJ	veil_NN	still_RB	hung_VBZ	unbroken_JJ	conceal_NN
first_JJ	steps_NNS	escape_VBP	far_RB	less_JJR	half_JJ	way_NN
hamlet_NN	little_JJ	beyond_IN	bottom_NN	hill_NN	must_MD	come_VB
forth_NN	moonlight_VBN	sound_JJ	several_JJ	footsteps_NNS		
running_VBG	came_VBD	already_RB	ears_VBZ	looked_VBD	back_RP	
direction_NN	light_NN	tossing_VBG	fro_NN	still_RB	rapidly_RB	





Preprocesses corpus with selected methods.

### 8.1 Signals

#### Inputs:

- **Corpus**  
Corpus instance.

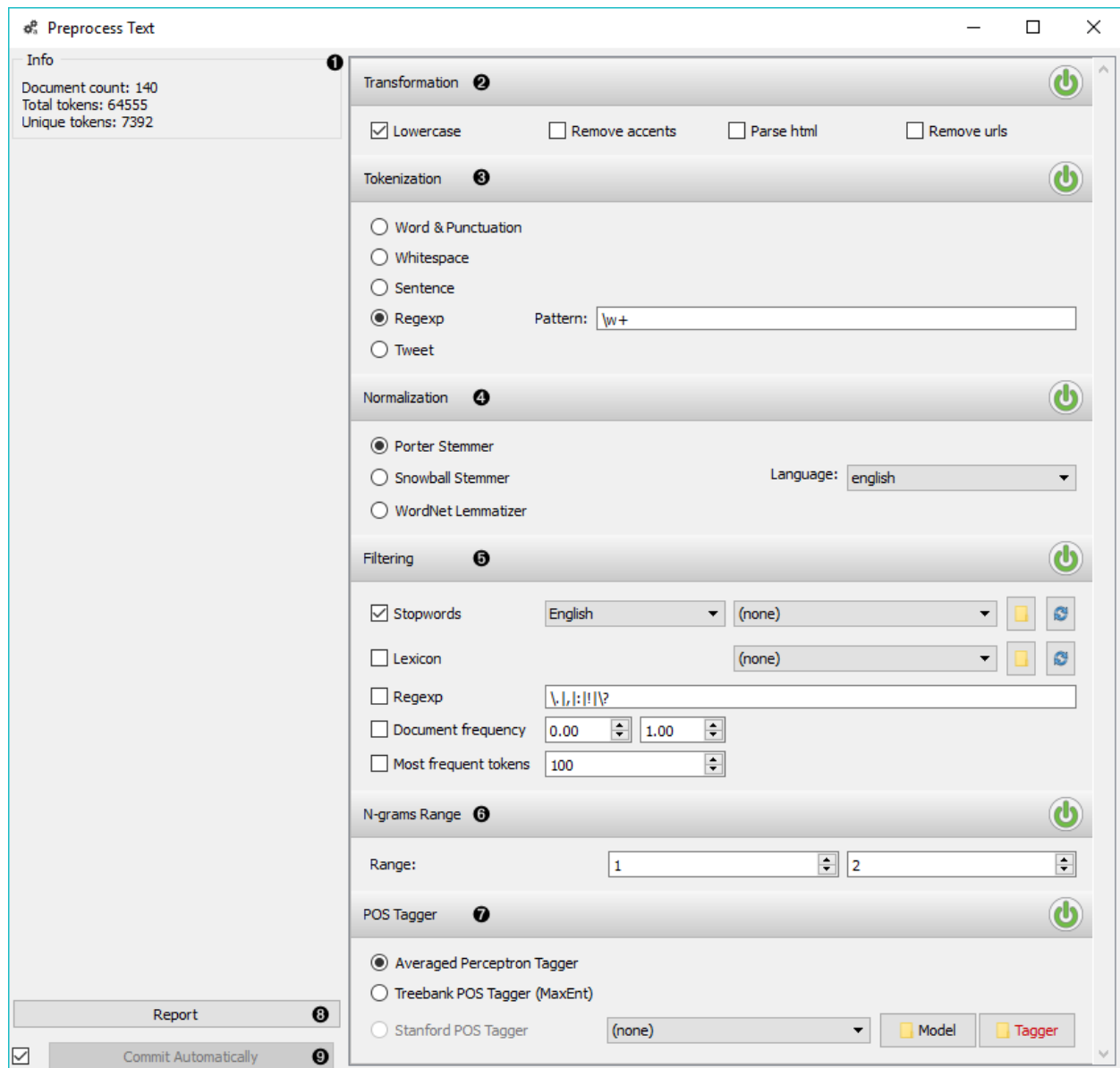
#### Outputs:

- **Corpus**  
Preprocessed corpus.

### 8.2 Description

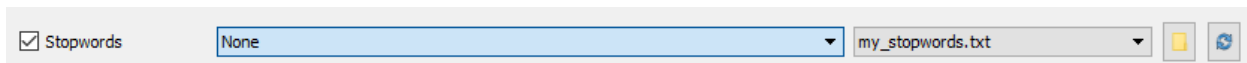
**Preprocess Text** splits your text into smaller units (tokens), filters them, runs **normalization** (stemming, lemmatization), creates **n-grams** and tags tokens with **part-of-speech** labels. Steps in the analysis are applied sequentially and can be turned on or off.

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</a> → 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.)
  - *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 always 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.

---

## 8.3 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

## 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 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 queried 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.







Generates a bag of words from the input corpus.

### 9.1 Signals

#### Inputs:

- **Corpus**  
Corpus instance.

#### Outputs:

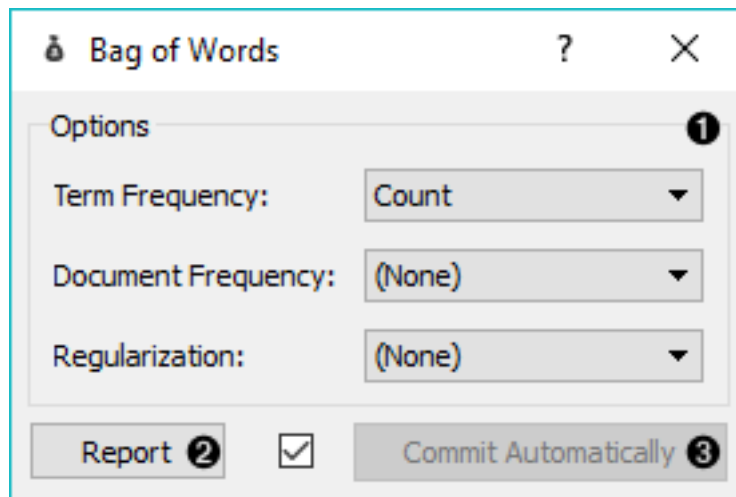
- **Corpus**  
Corpus with bag of words.

### 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 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*.

## 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*.

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 an Orange3 workflow for text mining. 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:**

Learners: SVM

Show: Number of instances

Select: Select Correct, Select Misclassified, Clear Selection

Output: ☒ Predictions, ☐ Probabilities

☒ Send Automatically

**Corpus Viewer Widget:**

Info:

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

Search features:

- category
- text
- category(SVM)

Display features:

- category
- text
- category(SVM)

☐ Show Tokens & Tags

RegEx Filter:

Document	category	text
1 Document 1	children	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.
2 Document 2		
3 Document 3		
4 Document 4		



# CHAPTER 10

---

## Topic Modelling

---



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

### 10.1 Signals

**Inputs:**

- **Corpus**

Corpus instance.

**Outputs:**

- **Data**

Data with topic weights appended.

- **Topics**

Selected topics with word weights.

### 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 ( $\gamma$ ): distribution at the first (corpus) level of Dirichlet Process
- Second level concentration ( $\alpha$ ): distribution at the second (document) level of Dirichlet Process
- The topic Dirichlet ( $\alpha$ ): 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 ( $\kappa$ ): step size
- Slow down parameter ( $\tau$ )

## 3. Produce a report.

4. If *Commit Automatically* is on, changes are communicated automatically. Alternatively press *Commit*.

## 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.

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*.

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. Document 1 is selected, showing its text content. The category is set to "children".
- Word Cloud:** Shows a word cloud generated from the corpus. The top words and their weights are listed in a table:
 

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
- Word Enrichment:** Shows a table of enriched words with their p-values and FDR values. The top words are:
 

Word	p-value	FDR
isis	0.0e+00	0.0e+00
terrorism	0.0e+00	0.0e+00
germany	2.1e-119	8.0e-116
drops	1.8e-89	5.3e-86
whose	2.1e-89	5.3e-86
comic	3.5e-88	6.4e-85
investigation	3.5e-88	6.4e-85
poem	1.7e-87	2.7e-84
turkey	5.8e-72	8.1e-69
us	1.1e-66	1.4e-63
resort	2.0e-43	2.4e-40
npr	2.3e-35	2.4e-32
uk	2.9e-30	3.5e-27
india	6.3e-28	5.7e-25
law	2.3e-17	1.9e-14
france	5.6e-15	4.4e-12
talks	2.9e-11	2.2e-08
amp	5.6e-10	3.9e-07
german	9.0e-10	6.0e-07
db	7.5e-08	4.7e-05
new	3.2e-07	1.9e-04
eu	7.7e-05	0.04432
- Select Rows:** Shows a condition: "Topic1 (german) is greater than 0.9". The data table shows 6526 rows and 12686 variables.
- Word Cloud (Bottom):** Shows a word cloud generated from the selected rows. The top words are:
 

Weight	Word
0.830	germany
0.166	terrorism
0.161	isis
0.161	belgium
0.159	operations
0.159	ministry
0.159	terrorist

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.



Word enrichment analysis for selected documents.

### 11.1 Signals

#### Inputs:

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

#### Outputs:

- (None)

### 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.**

**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

- 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)

## 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.

The screenshot displays an Orange3 workflow for text mining. The main workflow consists of four widgets: **Twitter**, **Preprocess Text**, **Bag of Words**, and **Word Enrichment**, connected in sequence. A **Corpus Viewer** widget is also connected to the **Bag of Words** widget.

The **Twitter** widget shows a query for tweets by **Author** with the list: `realDonaldTrump` and `HillaryClinton`. The search criteria include: **Search by:** Author, **Allow retweets:** unchecked, **Date:** since 2016-09-25 until 2016-10-05, **Language:** English, **Max tweets:** 100, **Accumulate results:** unchecked, **Text includes:** Content (checked), Author Description (unchecked). The **Info** section shows **Tweets on output:** 354.

The **Corpus Viewer** widget shows the following information: **Info:** Documents: 354, Preprocessed: True, Tokens: 4309, Types: 1632, POS tagged: False, N-grams range: 1-1, Matching: 98/354. The **Search features** list includes Author, Content, Date, Language, and Location. The **Display features** list includes the same. The **RegEx Filter:** is set to `Trump`. The **Info** section shows a list of tweets, with the first one selected: **1** Wow, did you just ... **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.

The **Word Enrichment** widget shows the following information: **Info:** Cluster words: 1632, Selected words: 614, After filtering: 15. The **Filter** section shows **p-value** set to 0.0100 and **FDR** set to 0.2000. The **Word** table shows the following results:

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

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.







Generates a word cloud from corpus.

## 12.1 Signals

### Inputs:

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

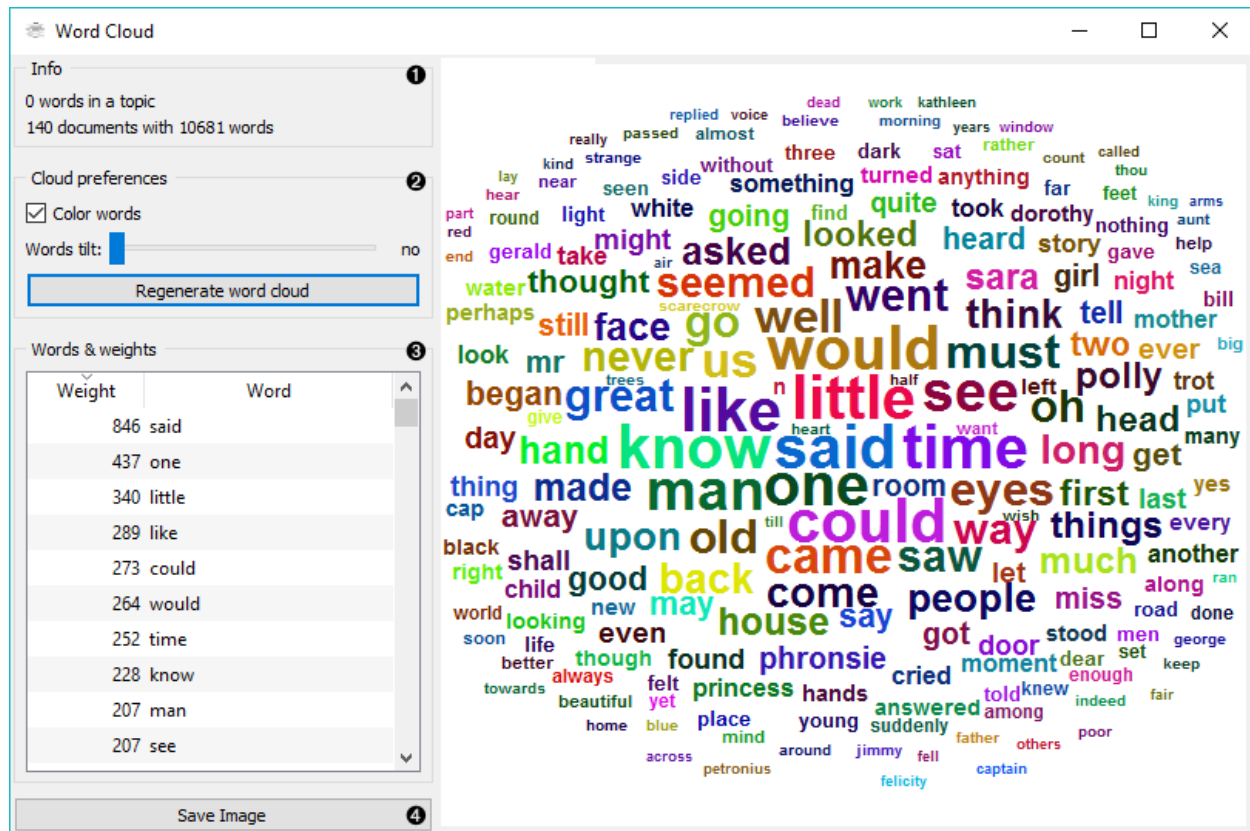
### Outputs:

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

## 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.

## 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.







Displays geographic distribution of data.

## 13.1 Signals

### Inputs:

- **Data**  
Data set.

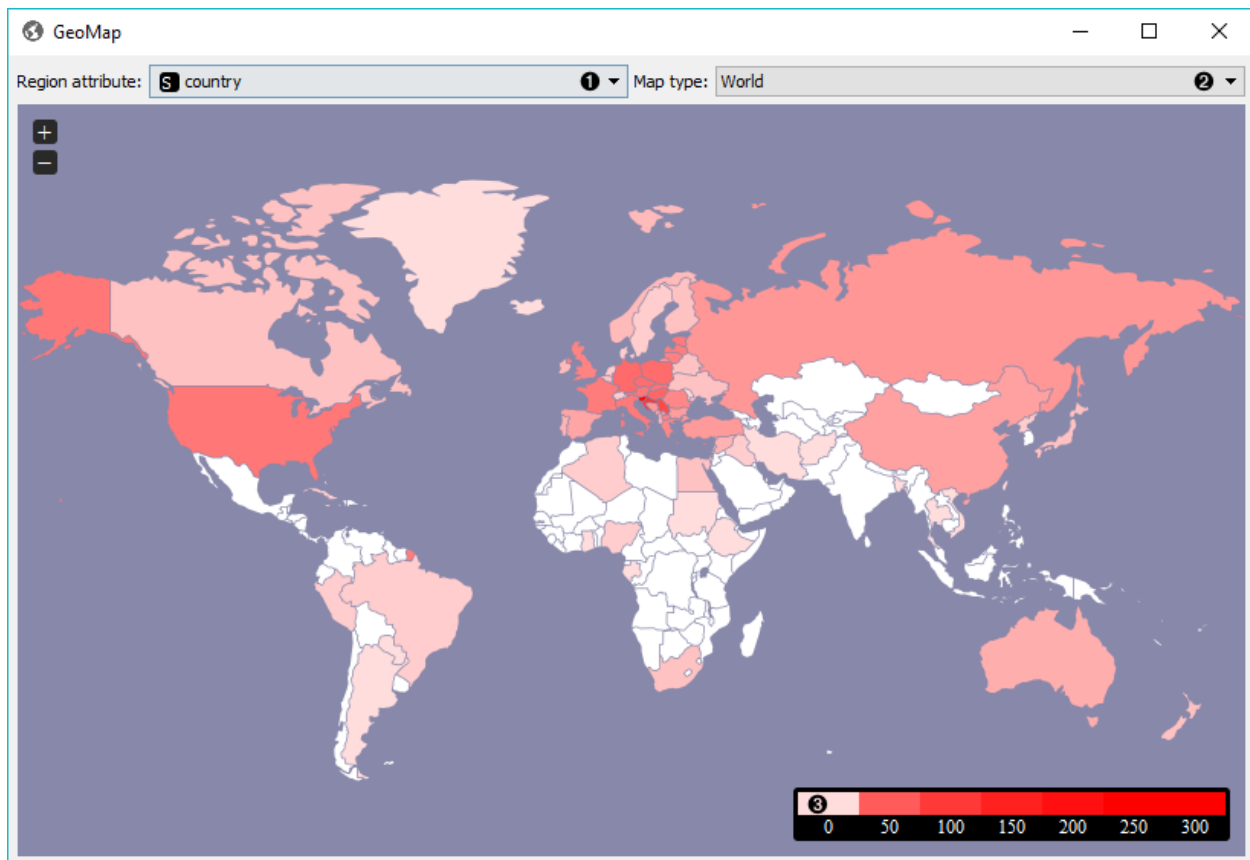
### Outputs:

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

## 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 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.

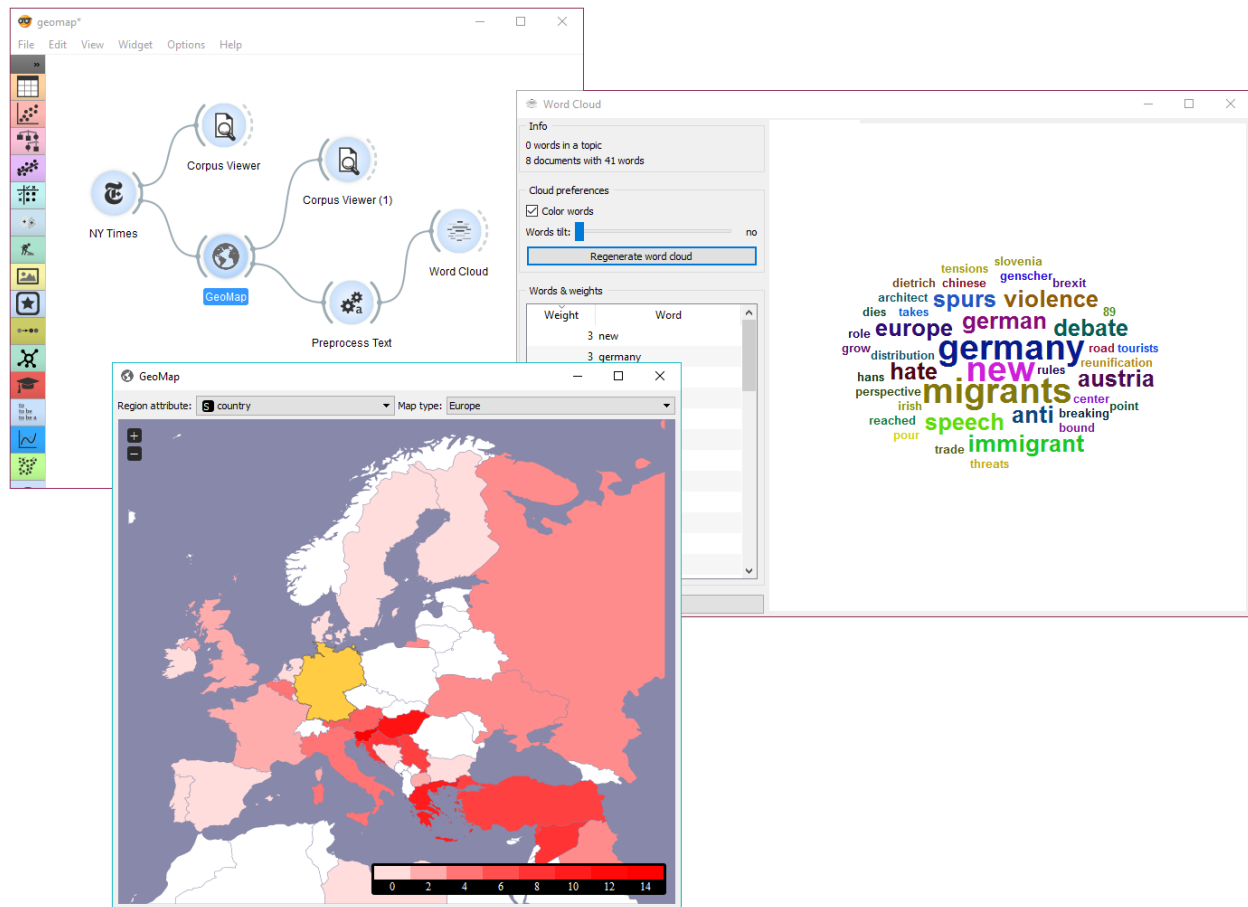
## 13.3 Example

**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*.

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*.







```
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.



## CHAPTER 15

---

### Preprocessor

---



**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



**class** orangecontrib.text.nytimes.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*



# 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 Credentials.
- **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*

**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** (*callable*) – 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.



```
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.
```





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
```

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 23

---

### Indices and tables

---

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



### O

`orangecontrib.text.guardian`, [63](#)  
`orangecontrib.text.tag`, [69](#)  
`orangecontrib.text.twitter`, [59](#)  
`orangecontrib.text.widgets.utils.concurrent`,  
[70](#)

## Symbols

\_\_init\_\_() (orangecontrib.text.corpus.Corpora method), 53  
 \_\_init\_\_() (orangecontrib.text.guardian.TheGuardianAPI method), 63  
 \_\_init\_\_() (orangecontrib.text.guardian.TheGuardianCredentials method), 63  
 \_\_init\_\_() (orangecontrib.text.nyt.NYT method), 61

## A

api\_key\_valid() (orangecontrib.text.nyt.NYT method), 61

## C

check() (orangecontrib.text.tag.StanfordPOSTagger class method), 69  
 copy() (orangecontrib.text.corpus.Corpora method), 53  
 Corpora (class in orangecontrib.text.corpus), 53  
 Credentials (class in orangecontrib.text.twitter), 59

## D

dictionary (orangecontrib.text.corpus.Corpora attribute), 53  
 documents (orangecontrib.text.corpus.Corpora attribute), 53  
 documents\_from\_features() (orangecontrib.text.corpus.Corpora method), 53

## E

extend\_attributes() (orangecontrib.text.corpus.Corpora method), 53  
 extend\_corpus() (orangecontrib.text.corpus.Corpora method), 54

## F

fit() (orangecontrib.text.topics.HdpWrapper method), 67  
 fit() (orangecontrib.text.topics.LdaWrapper method), 67  
 fit() (orangecontrib.text.topics.LsiWrapper method), 67  
 from\_documents() (orangecontrib.text.corpus.Corpora static method), 54

## H

has\_tokens() (orangecontrib.text.corpus.Corpora method), 54  
 HdpWrapper (class in orangecontrib.text.topics), 67

## L

LdaWrapper (class in orangecontrib.text.topics), 67  
 LsiWrapper (class in orangecontrib.text.topics), 67

## N

ngrams (orangecontrib.text.corpus.Corpora attribute), 54  
 NYT (class in orangecontrib.text.nyt), 61

## O

orangecontrib.text.guardian (module), 63  
 orangecontrib.text.tag (module), 69  
 orangecontrib.text.twitter (module), 59  
 orangecontrib.text.widgets.utils.concurrent (module), 70

## P

POSTagger (class in orangecontrib.text.tag), 69

## R

reset() (orangecontrib.text.twitter.TwitterAPI method), 59  
 retain\_preprocessing() (orangecontrib.text.corpus.Corpora static method), 54

## S

search() (orangecontrib.text.guardian.TheGuardianAPI method), 63  
 search() (orangecontrib.text.nyt.NYT method), 61  
 search() (orangecontrib.text.wikipedia.WikipediaAPI method), 65  
 search\_authors() (orangecontrib.text.twitter.TwitterAPI method), 59  
 search\_content() (orangecontrib.text.twitter.TwitterAPI method), 59  
 set\_text\_features() (orangecontrib.text.corpus.Corpora method), 54



StanfordPOSTagger (class in orangecontrib.text.tag), [69](#)  
store\_tokens() (orangecontrib.text.corpus.Corpus  
method), [54](#)

## T

tag\_corpus() (orangecontrib.text.tag.POSTagger method),  
[69](#)  
TheGuardianAPI (class in orangecontrib.text.guardian),  
[63](#)  
TheGuardianCredentials (class in orangecon-  
trib.text.guardian), [63](#)  
titles (orangecontrib.text.corpus.Corpus attribute), [54](#)  
tokens (orangecontrib.text.corpus.Corpus attribute), [54](#)  
transform() (orangecontrib.text.topics.HdpWrapper  
method), [67](#)  
transform() (orangecontrib.text.topics.LdaWrapper  
method), [67](#)  
transform() (orangecontrib.text.topics.LsiWrapper  
method), [67](#)  
TwitterAPI (class in orangecontrib.text.twitter), [59](#)

## V

valid (orangecontrib.text.guardian.TheGuardianCredentials  
attribute), [63](#)

## W

WikipediaAPI (class in orangecontrib.text.wikipedia), [65](#)