Orange3 Text Mining Documentation

Biolab

Dec 11, 2018

Contents

CHAPTER 1

Widgets

1.1 Corpus

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

Inputs None

Outputs

Corpus A collection of documents.

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

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

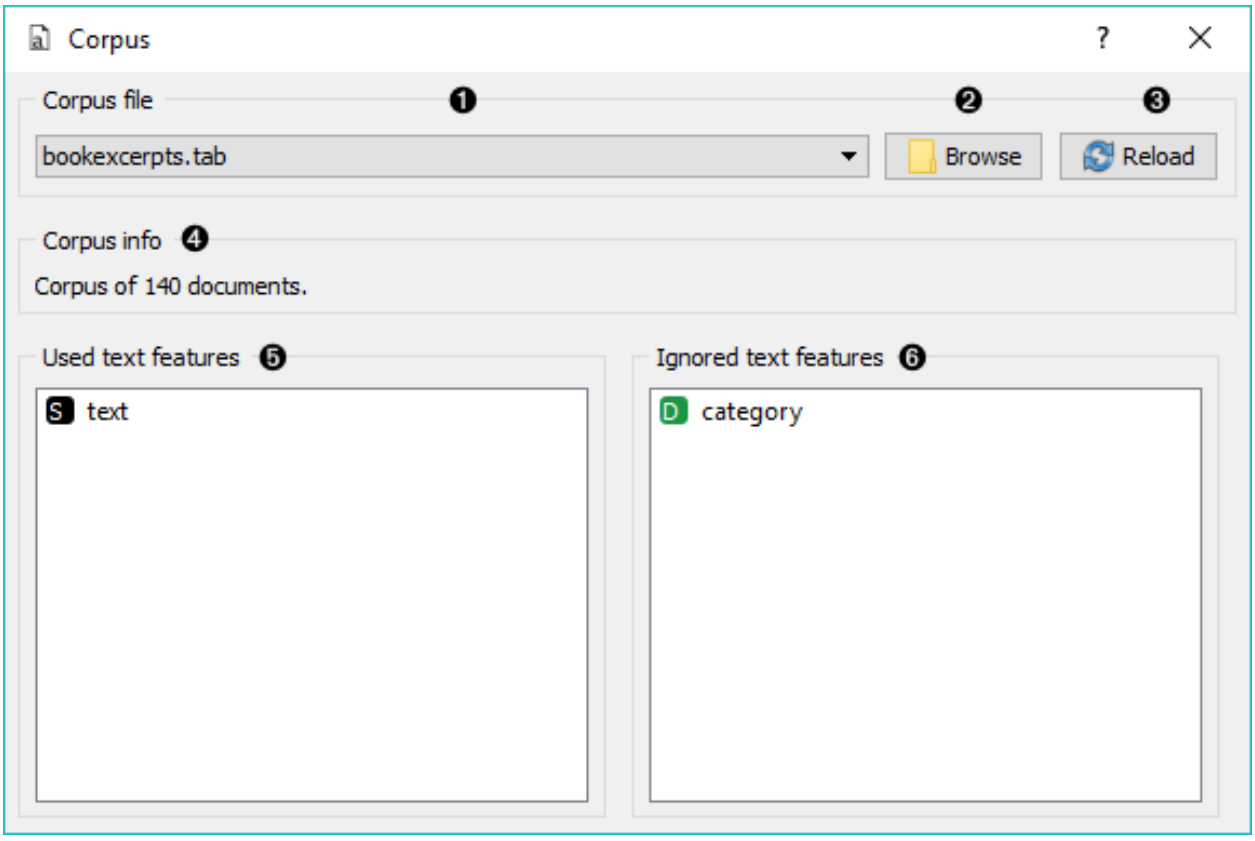
1. Browse through previously opened data files, or load any of the sample ones.
2. Browse for a data file.
3. Reloads currently selected data file.
4. Information on the loaded data set.
5. Features that will be used in text analysis.
6. Features that won't be used in text analysis and serve as labels or class.

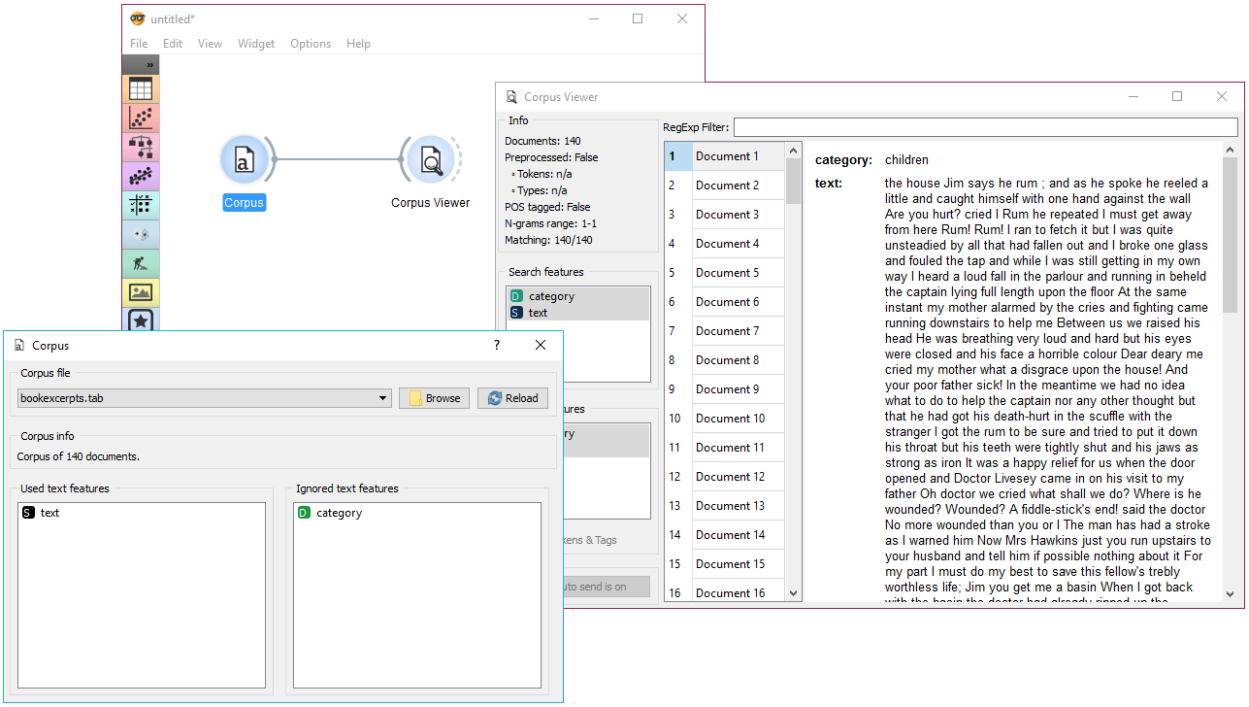
You can drag and drop features between the two boxes and also change the order in which they appear.

1.1.1 Example

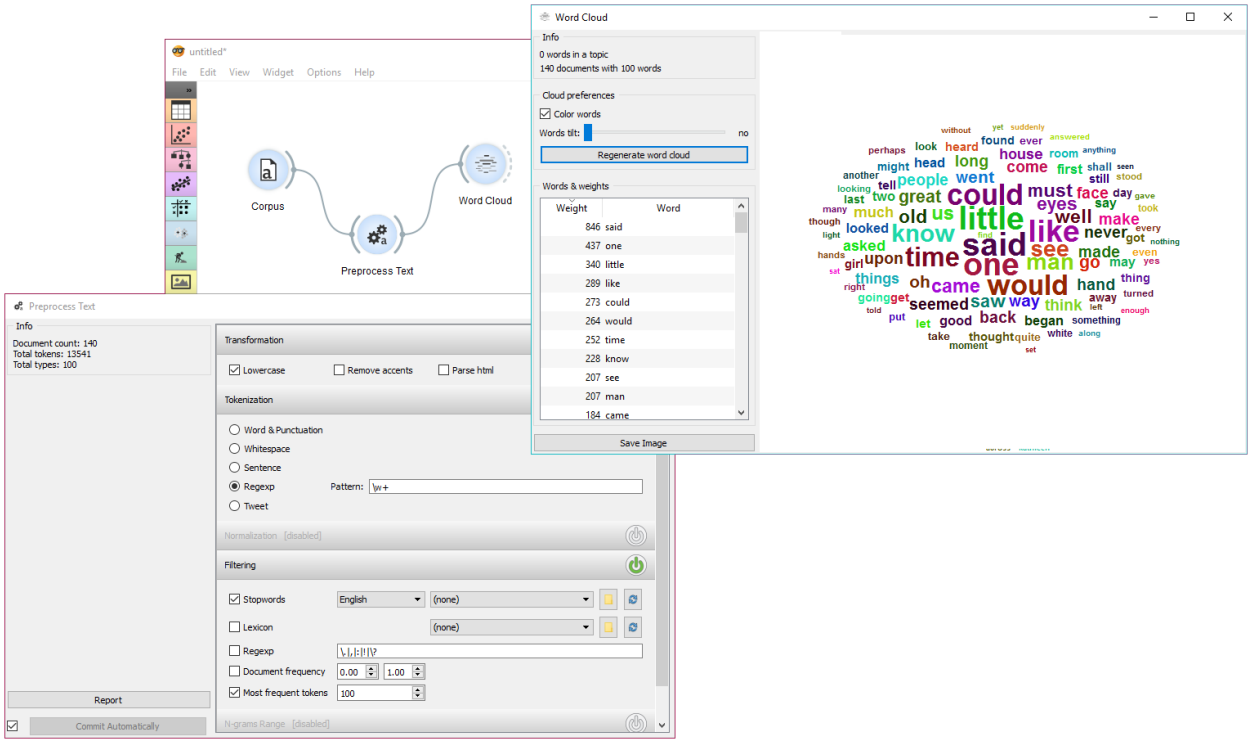
The first example shows a very simple use of **Corpus** widget. Place **Corpus** onto canvas and connect it to *Corpus Viewer*. We've used *booxexcerpts.tab* data set, which comes with the add-on, and inspected it in **Corpus Viewer**.

The second example demonstrates how to quickly visualize your corpus with *Word Cloud*. We could connect **Word Cloud** directly to **Corpus**, but instead we decided to apply some preprocessing with *Preprocess Text*. We are again


Corpus
Corpus file
1
2
3
bookexcerpts.tab
Browse
Reload
Corpus info
4
Corpus of 140 documents.
Used text features
5
text
Ignored text features
6
category


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

working with *book-excerpts.tab*. We've put all text to lowercase, tokenized (split) the text to words only, filtered out English stopwords and selected a 100 most frequent tokens.



1.2 Import Documents

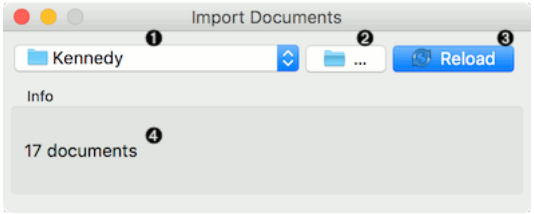
Import text documents from folders.

Inputs None

Outputs

Corpus A collection of documents from the local machine.

Import Documents widget retrieves text files from folders and creates a corpus. The widget reads .txt, .docx, .odt, .pdf and .xml files. If a folder contains subfolders, they will be used as class labels.

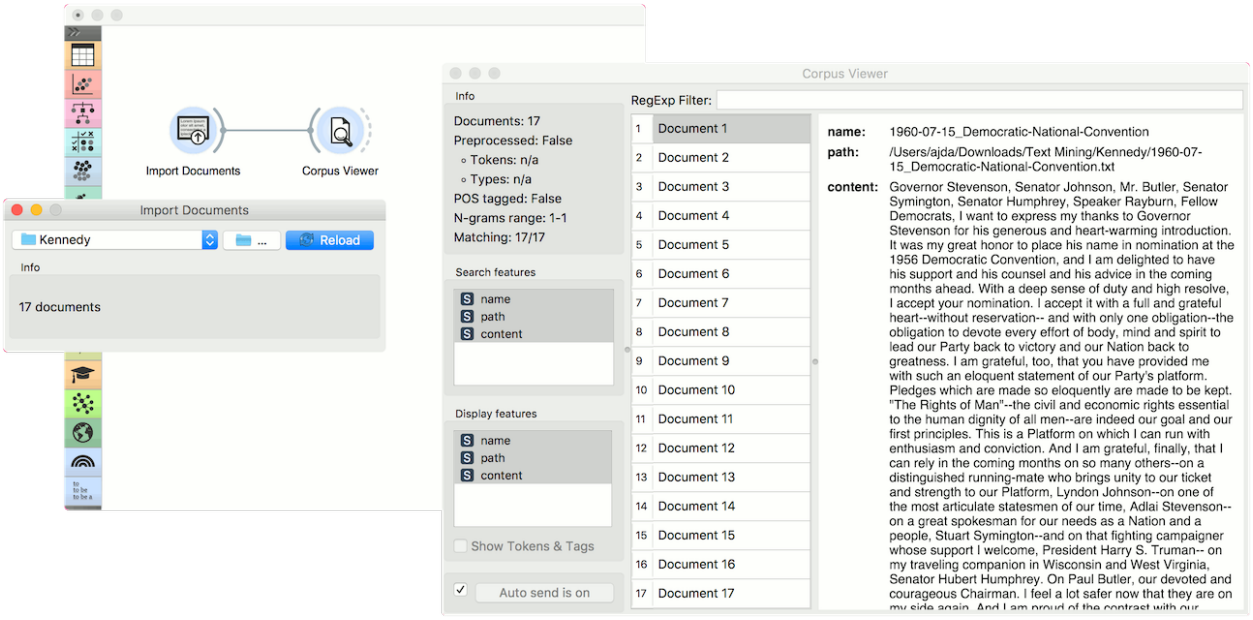


1. Folder being loaded.
2. Load folder from a local machine.
3. Reload the data.
4. Number of documents retrieved.

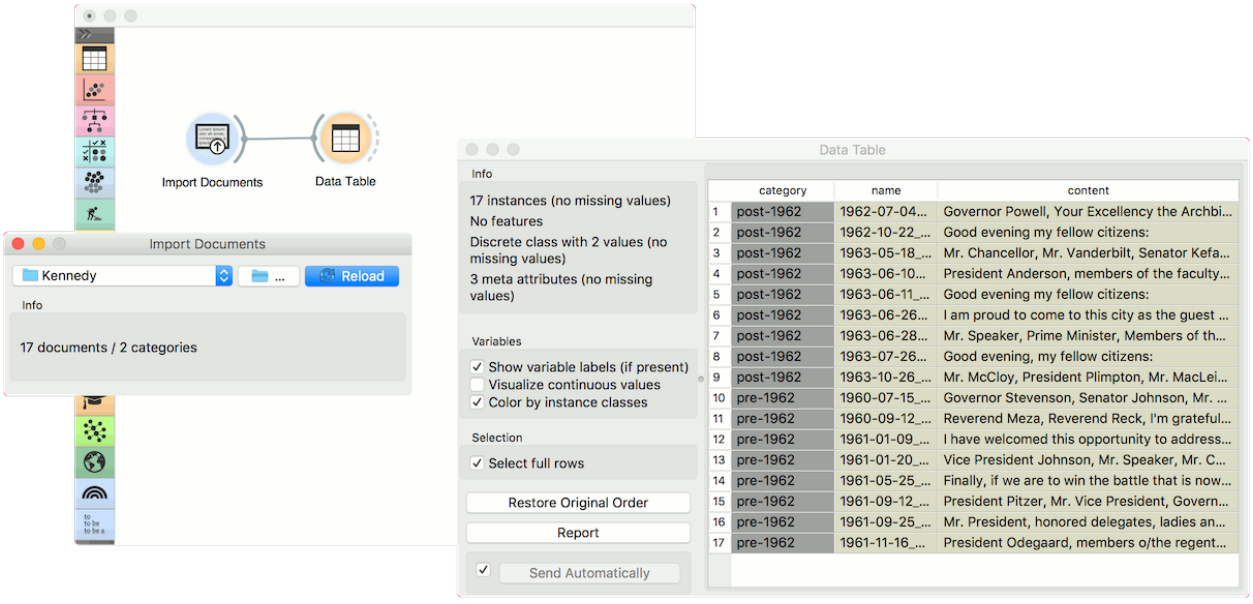
If the widget cannot read the file for some reason, the file will be skipped. Files that were successfully retrieved will still be on the output.

1.2.1 Example

To retrieve the data, select the folder icon on the right side of the widget. Select the folder you wish to turn into coprus. Once the loading is finished, you will see how many documents the widget retrieved. To inspect them, connect the widget to *Corpus Viewer*. We've used a set of Kennedy's speeches in a plain text format.



Now let us try it with subfolders. We have placed Kennedy's speeches in two folders - pre-1962 and post-1962. If I load the parent folder, these two subfolders will be used as class labels. Check the output of the widget in a **Data Table**.



1.3 The Guardian

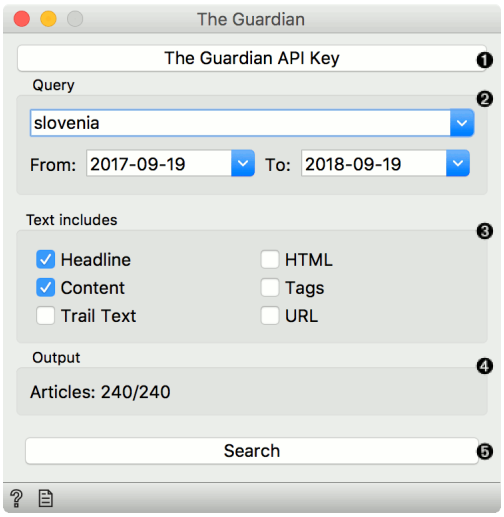
Fetching data from The Guardian Open Platform.

Inputs None

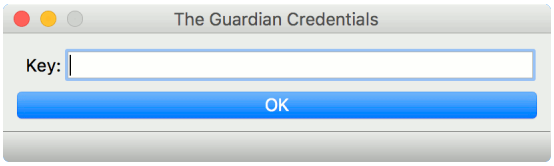
Outputs

Corpus A collection of documents from the Guardian newspaper.

Guardian retrieves articles from the Guardian newspaper via their API. For the widget to work, you need to provide the API key, which you can get at their access platform.



1. Insert the API key for the widget to work.



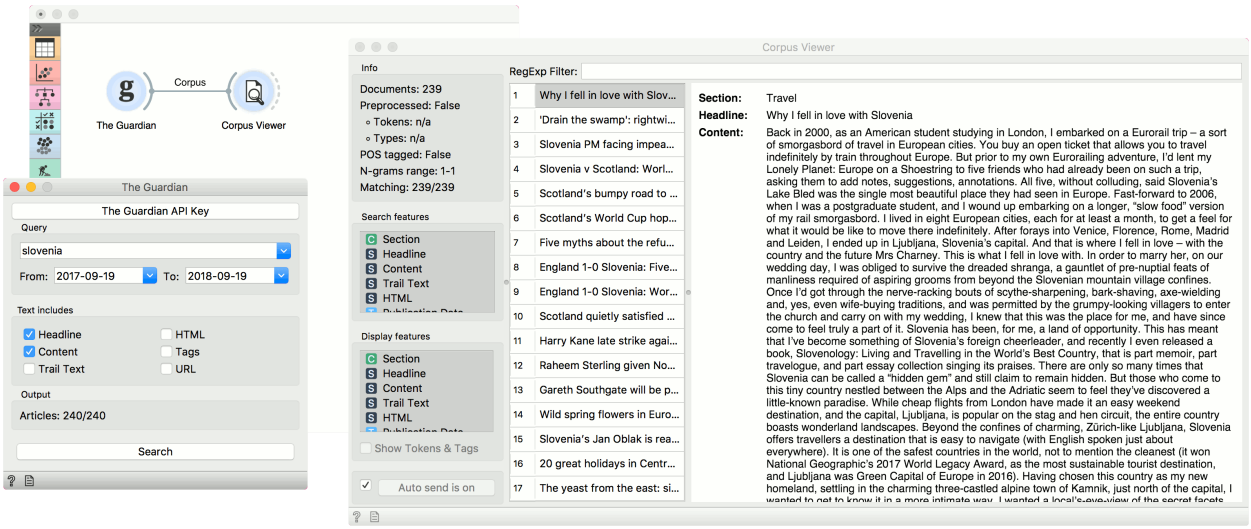
2. Provide the query and set the time frame from which to retrieve the articles.
3. Define which features to retrieve from the Guardian platform.
4. Information on the output.
5. Press *Search* to start retrieving the articles or *Stop* to stop the retrieval.

1.3.1 Example

Guardian can be used just like any other data retrieval widget in Orange, namely *NY Times*, *Wikipedia*, *Twitter* or *PubMed*.

We will retrieve 240 articles mentioning *slovenia* between september 2017 and september 2018. The text will include article headline and content. Upon pressing *Search*, the articles will be retrieved.

We can observe the results in the *Corpus Viewer* widget.



1.4 NY Times

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

Inputs None

Outputs

Corpus A collection of documents from the New York Times newspaper.

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

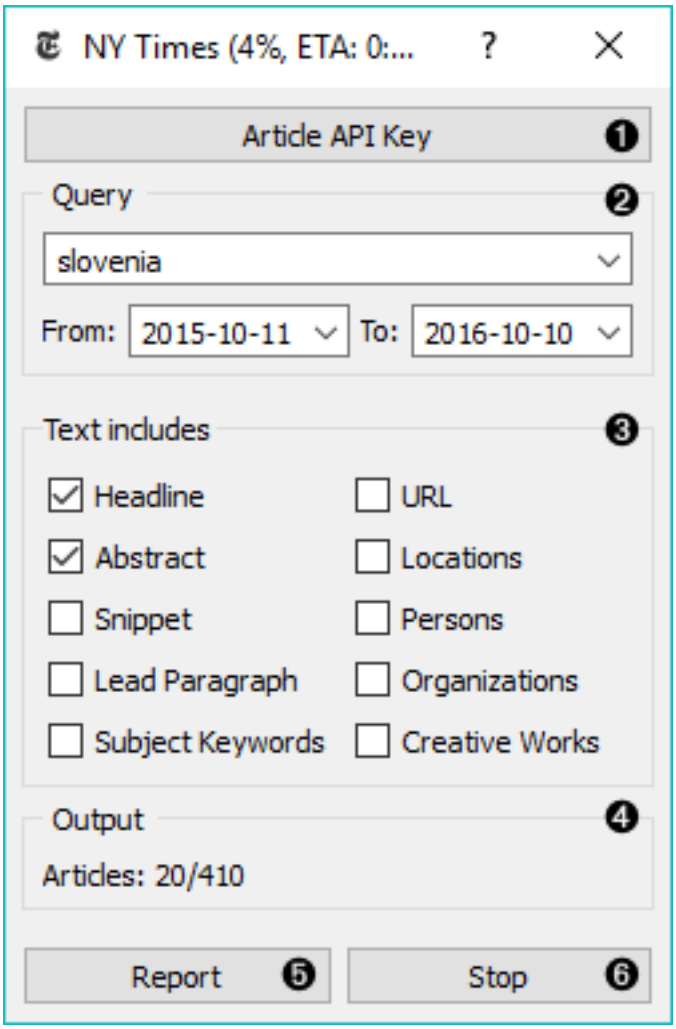
To use the widget, you must enter your own API key.

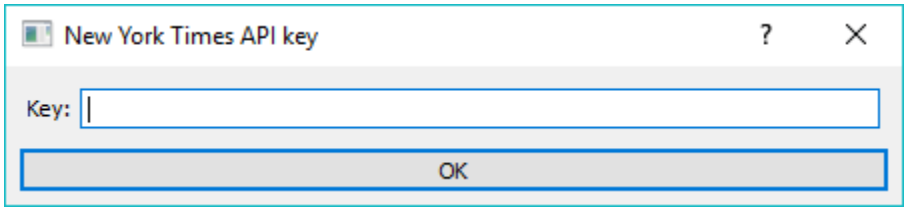
1. To begin your query, insert NY Times' Article Search API key. The key is securely saved in your system keyring service (like Credential Vault, Keychain, KWallet, etc.) and won't be deleted when clearing widget settings.
2. **Set query parameters:**
   - *Query*
   - Query time frame. The widget allows querying articles from September 18, 1851 onwards. Default is set to 1 year back from the current date.
3. Define which features to include as text features.
4. Information on the output.
5. Produce report.
6. Run or stop the query.

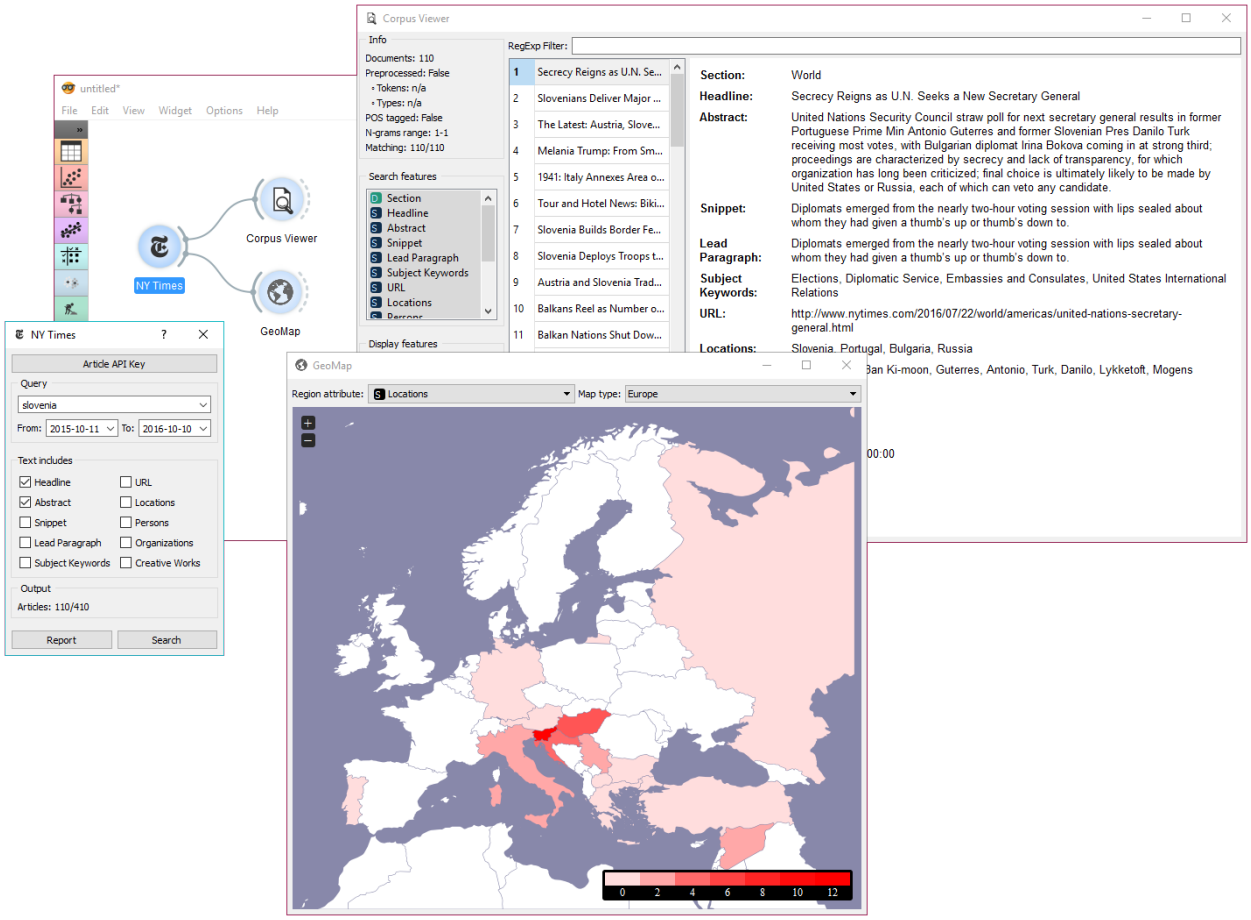
1.4.1 Example

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.


NY Times (4%, ETA: 0:...
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
Key:
OK


Corpus Viewer
Info
Documents: 110
Preprocessed: False
Tokens: n/a
Types: n/a
POS tagged: False
N-grams range: 1-1
Matching: 110/110
Search features
Section
Headline
Abstract
Snippet
Lead Paragraph
Subject Keywords
URL
Locations
Display features
RegExp Filter:
1 Secrecy Reigns as U.N. Se...
2 Slovenians Deliver Major ...
3 The Latest: Austria, Slove...
4 Melania Trump: From Sm...
5 1941: Italy Annexes Area o...
6 Tour and Hotel News: Biki...
7 Slovenia Builds Border Fe...
8 Slovenia Deploys Troops t...
9 Austria and Slovenia Trad...
10 Balkans Reel as Number o...
11 Balkan Nations Shut Dow...
Section: World
Headline: Secrecy Reigns as U.N. Seeks a New Secretary General
Abstract: United Nations Security Council straw poll for next secretary general results in former Portuguese Prime Min Antonio Guterres and former Slovenian Pres Danilo Turk receiving most votes, with Bulgarian diplomat Irina Bokova coming in at strong third; proceedings are characterized by secrecy and lack of transparency, for which organization has long been criticized; final choice is ultimately likely to be made by United States or Russia, each of which can veto any candidate.
Snippet: Diplomats emerged from the nearly two-hour voting session with lips sealed about whom they had given a thumb's up or thumb's down to.
Lead Paragraph: Diplomats emerged from the nearly two-hour voting session with lips sealed about whom they had given a thumb's up or thumb's down to.
Subject Keywords: Elections, Diplomatic Service, Embassies and Consulates, United States International Relations
URL: http://www.nytimes.com/2016/07/22/world/americas/united-nations-secretary-general.html
Locations: Slovenia, Portugal, Bulgaria, Russia
Ban Ki-moon, Guterres, Antonio, Turk, Danilo, Lykketoft, Mogens
00:00
untitled*
File Edit View Widget Options Help
Corpus Viewer
NY Times
GeoMap
NY Times
Article API Key
Query
slovenia
From: 2015-10-11 To: 2016-10-10
Text includes
Headline
URL
Abstract
Locations
Snippet
Persons
Lead Paragraph
Organizations
Subject Keywords
Creative Works
Output
Articles: 110/410
Report
Search
GeoMap
Region attribute: Locations
Map type: Europe
0 2 4 6 8 10 12

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

1.5 Pubmed

Fetch data from PubMed journals.

Inputs None

Outputs

Corpus A collection of documents from the PubMed online service.

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

1. Enter a valid e-mail to retrieve queries.
2. ***Regular search*:**
    - *Author*: queries entries from a specific author. Leave empty to query by all authors.
    - *From*: define the time frame of publication.
    - *Query*: enter the query.

    Advanced search: enables you to construct complex queries. See PubMed's website to learn how to construct such queries. You can also copy-paste constructed queries from the website.
3. *Find records* finds available data from PubMed matching the query. Number of records found will be displayed above the button.
4. Define the output. All checked features will be on the output of the widget.
5. Set the number of record you wish to retrieve. Press *Retrieve records* to get results of your query on the output. Below the button is an information on the number of records on the output.

1.5.1 Example

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

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

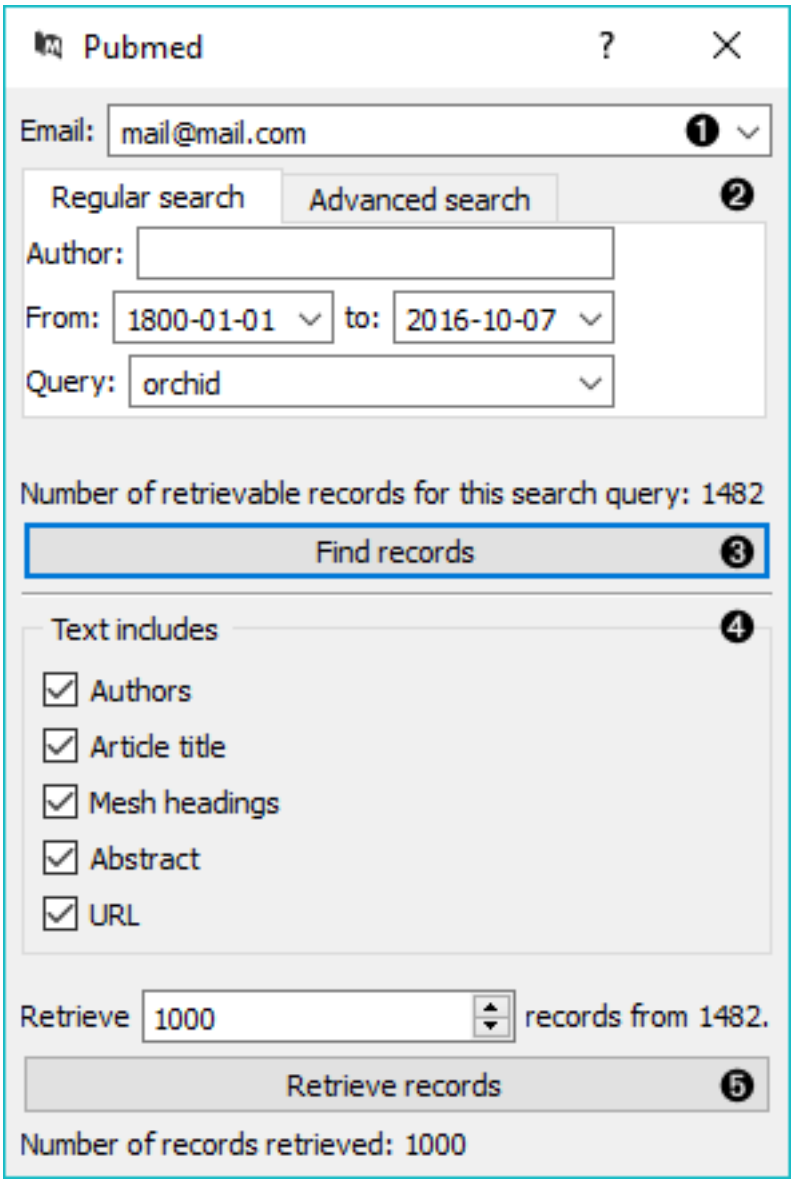
1.6 Twitter

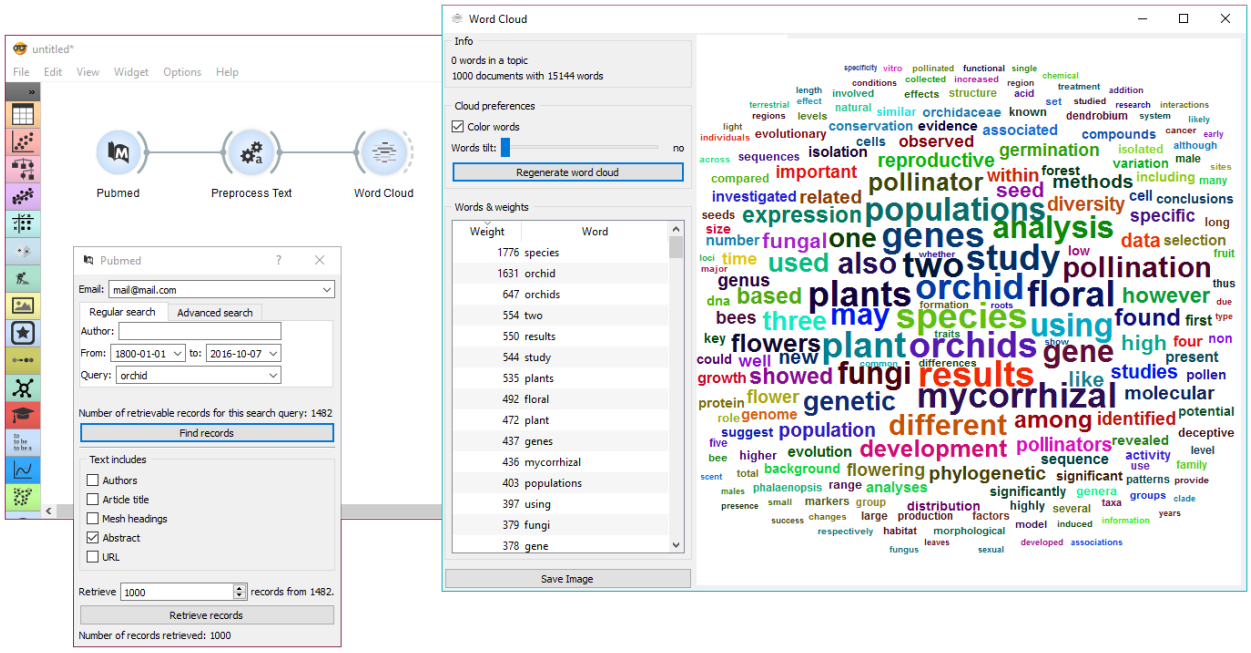
Fetching data from The Twitter Search API.

Inputs None

Outputs

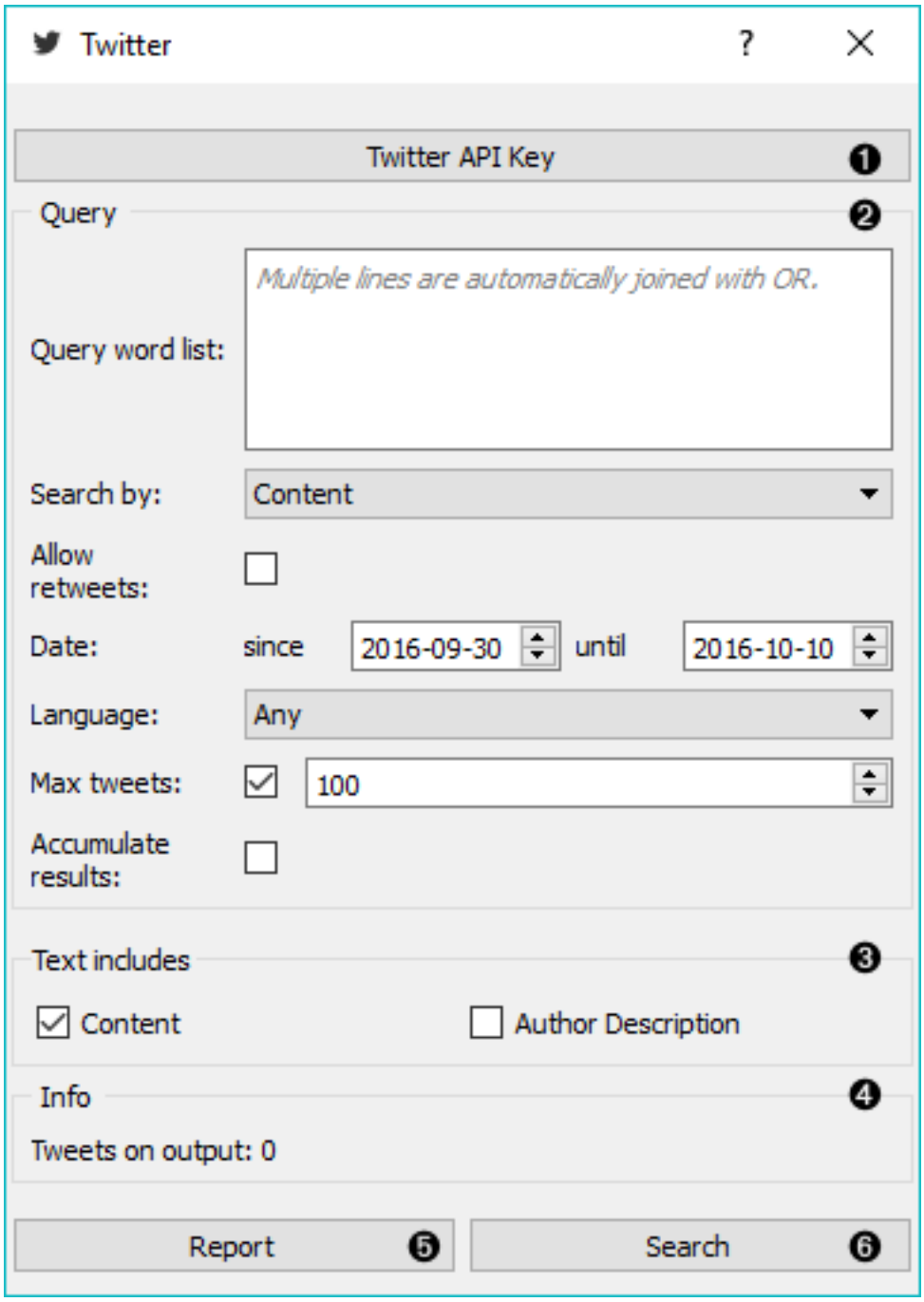
Corpus A collection of tweets from the Twitter API.

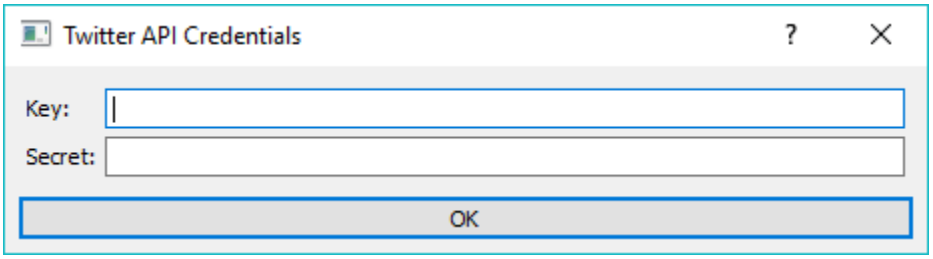

Pubmed
Email:
mail@mail.com
1
Regular search
Advanced search
2
Author:
From:
1800-01-01
to:
2016-10-07
Query:
orchid
Number of retrievable records for this search query: 1482
Find records
3
Text includes
4
Authors
Article title
Mesh headings
Abstract
URL
Retrieve
1000
records from 1482.
Retrieve records
5
Number of records retrieved: 1000



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

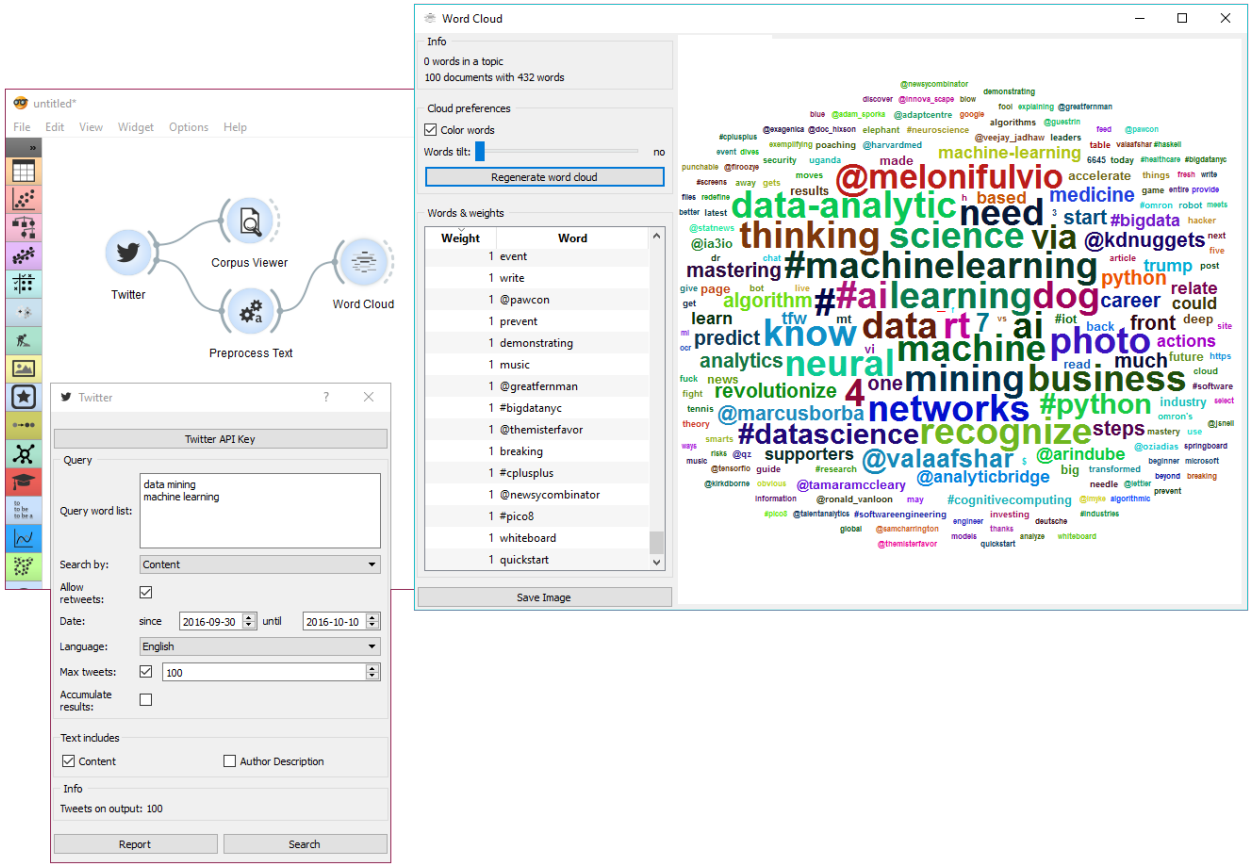
1. To begin your queries, insert Twitter key and secret. They are securely saved in your system keyring service (like Credential Vault, Keychain, KWallet, etc.) and won't be deleted when clearing widget settings. You must first create a Twitter app to get API keys.
2. **Set query parameters:**
    -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


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


Twitter API Credentials
Key:
Secret:
OK

1.6.1 Examples

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



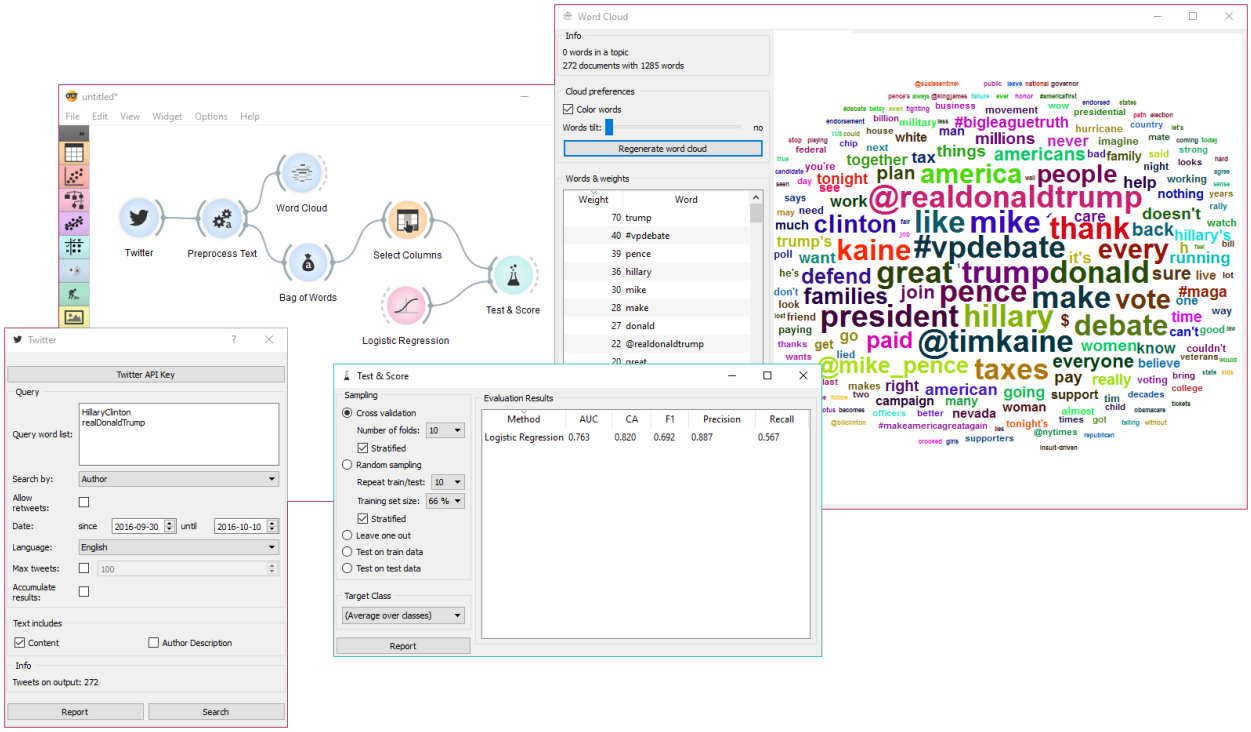
First, we're checking the output in *Corpus Viewer* to get the initial idea about our results. Then we're preprocessing the tweets with lowercase, url removal, tweet tokenizer and removal of stopword and punctuation. The best way to see the results is with *Word Cloud*. This will display the most popular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

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

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

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

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



1.7 Wikipedia

Fetching data from MediaWiki RESTful web service API.

Inputs None

Outputs

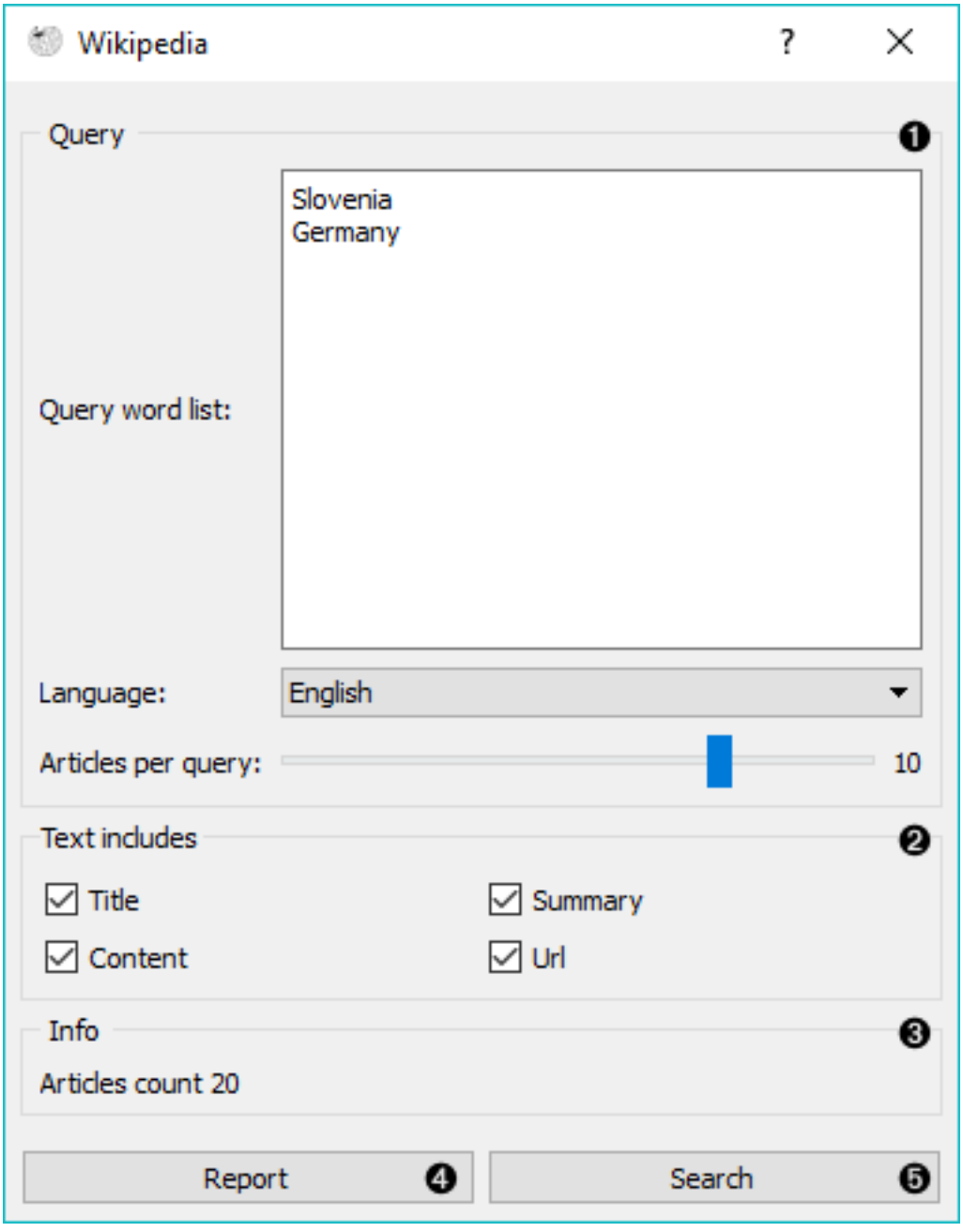
Corpus A collection of documents from the Wikipedia.

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

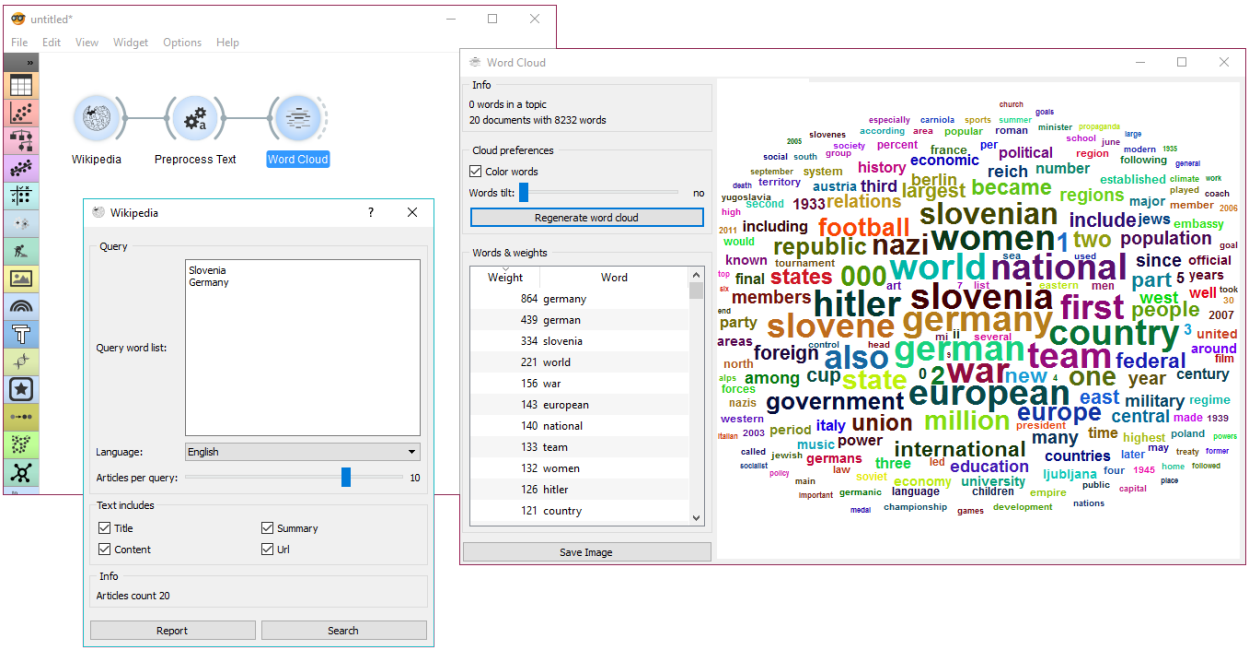
1. **Query parameters:**
   - Query word list, where each query is listed in a new line.
   - Language of the query. English is set by default.
   - Number of articles to retrieve per query (range 1-25). Please note that querying is done recursively and that disambiguations are also retrieved, sometimes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

1.7.1 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


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

Word Cloud.



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

1.8 Preprocess Text

Preprocesses corpus with selected methods.

Inputs

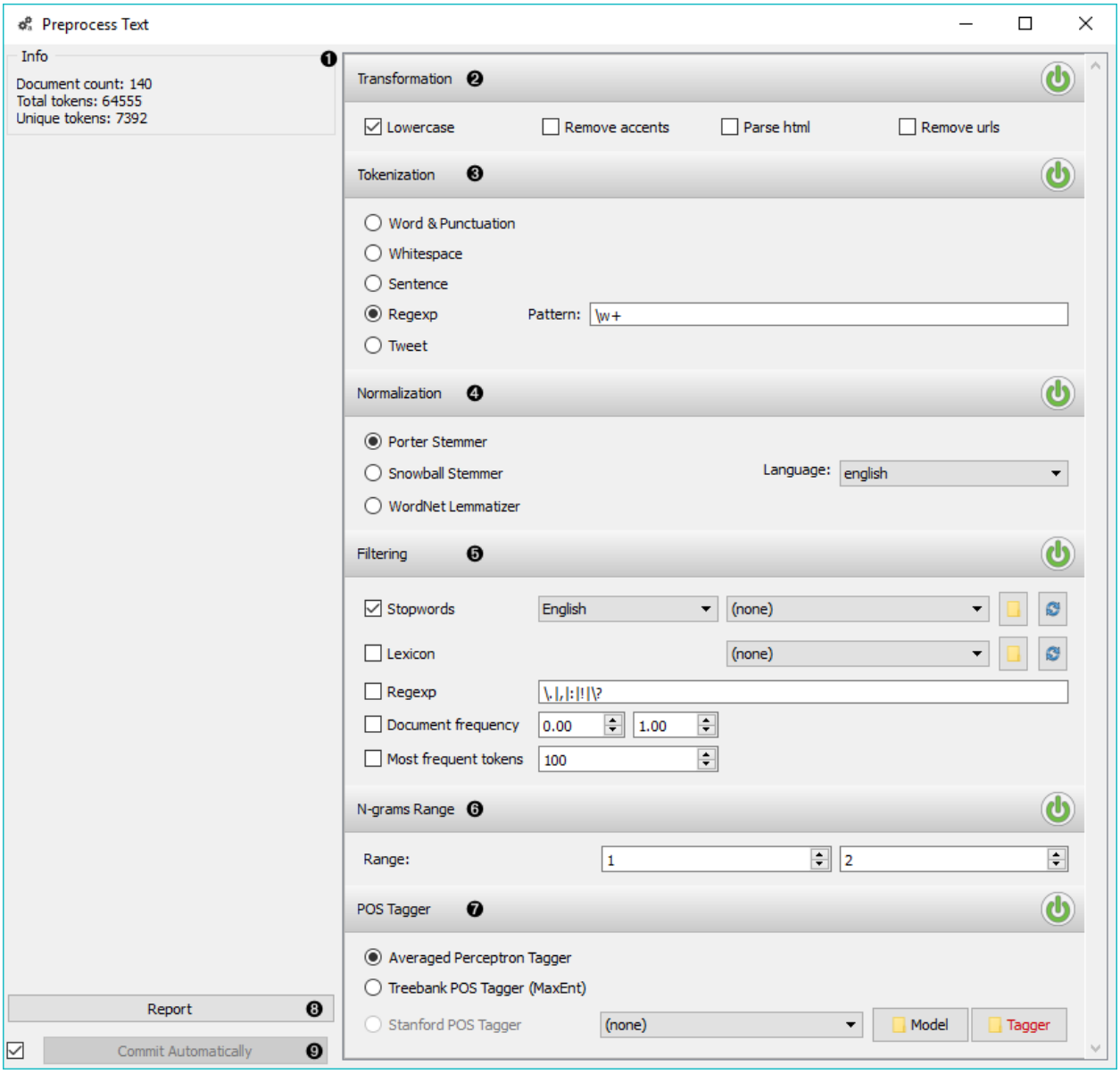
Corpus A collection of documents.

Outputs

Corpus Preprocessed corpus.

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

1. **Information on preprocessed data**. *Document count* reports on the number of documents on the input. *Total tokens* counts all the tokens in corpus. *Unique tokens* excludes duplicate tokens and reports only on unique tokens in the corpus.
2. **Transformation transforms input data. It applies lowercase transformation by default.**
   - *Lowercase* will turn all text to lowercase.
   - ***Remove accents* will remove all diacritics/accents in text.** naïve → naive
   - ***Parse html* will detect html tags and parse out text only.** <a href. . . >Some text → Some text
   - ***Remove urls* will remove urls from text.** This is a http://orange.biolab.si/ url. → This is a url.
3. **Tokenization is the method of breaking the text into smaller components (words, sentences, bigrams).**
   - ***Word & Punctuation* will split the text by words and keep punctuation symbols.** This example. → (This), (example), (.)


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

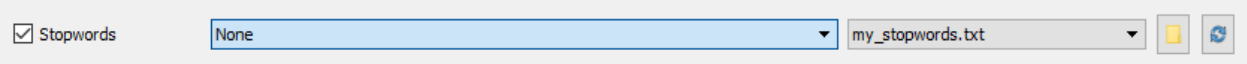
- ***Whitespace* will split the text by whitespace only.** This example. → (This), (example.)
- ***Sentence* will split the text by fullstop, retaining only full sentences.** This example. Another example. → (This example.), (Another example.)
- Regexp will split the text by provided regex. It splits by words only by default (omits punctuation).
- ***Tweet* will split the text by pre-trained Twitter model, which keeps hashtags, emoticons and other special symbol** This example. :-) #simple → (This), (example), (.), (:-)), (#simple)

4. **Normalization applies stemming and lemmatization to words. (I've always loved cats. → I have alway love cat.) For langu**

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

5. **Filtering removes or keeps a selection of words.**

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



Click 'browse' icon to select the file containing stopwords. If the file was properly loaded, its name will be displayed next to pre-loaded stopwords. Change 'English' to 'None' if you wish to filter out only the provided stopwords. Click 'reload' icon to reload the list of stopwords.

- *Lexicon* keeps only words provided in the file. Load a *.txt file with one word per line to use as lexicon. Click 'reload' icon to reload the lexicon.
- *Regexp* removes words that match the regular expression. Default is set to remove punctuation.
- *Document frequency* keeps tokens that appear in not less than and not more than the specified number / percentage of documents. If you provide integers as parameters, it keeps only tokens that appear in the specified number of documents. E.g. DF = (3, 5) keeps only tokens that appear in 3 or more and 5 or less documents. If you provide floats as parameters, it keeps only tokens that appear in the specified percentage of documents. E.g. DF = (0.3, 0.5) keeps only tokens that appear in 30% to 50% of documents. Default returns all tokens.
- *Most frequent tokens* keeps only the specified number of most frequent tokens. Default is a 100 most frequent tokens.

6. **N-grams Range** creates n-grams from tokens. Numbers specify the range of n-grams. Default returns one-grams and two-grams.
7. **POS Tagger runs part-of-speech tagging on tokens.**

- Averaged Perceptron Tagger runs POS tagging with Matthew Honnibal's averaged perceptron tagger.
- Treebank POS Tagger (MaxEnt) runs POS tagging with a trained Penn Treebank model.
- Stanford POS Tagger runs a log-linear part-of-speech tagger designed by Toutanova et al. Please download it from the provided website and load it in Orange.

8. Produce a report.

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

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

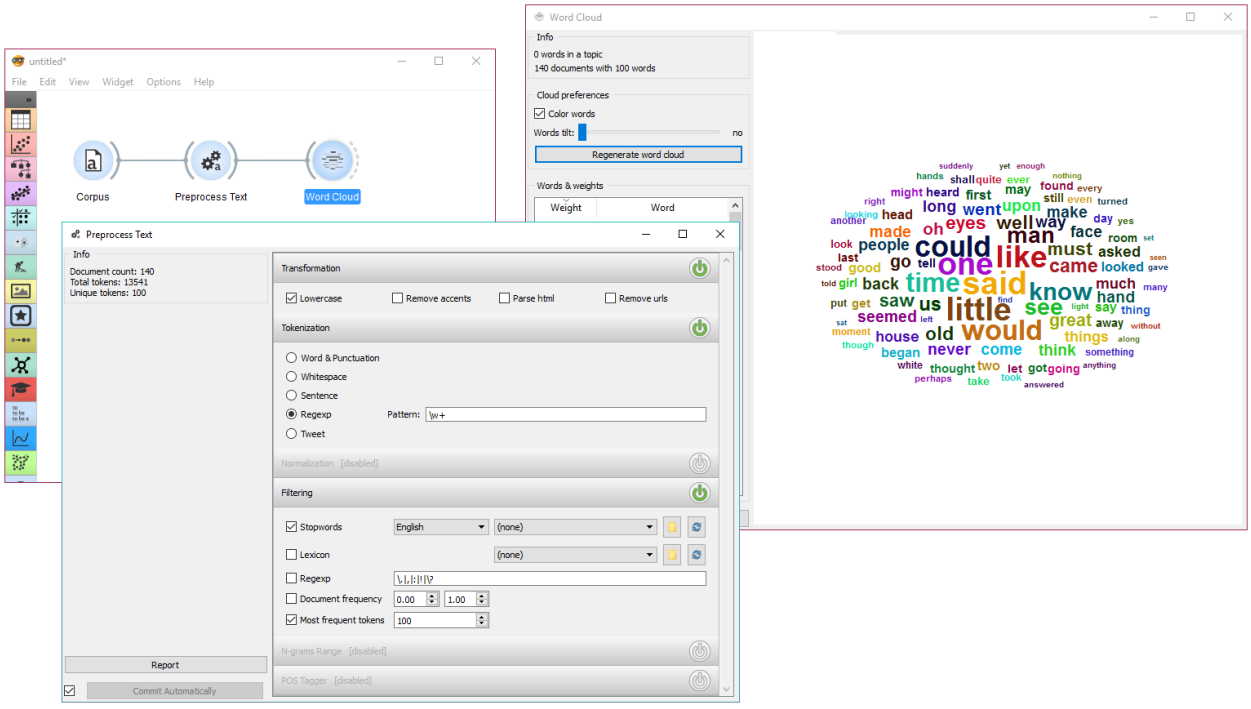
1.8.1 Useful Regular Expressions

Here are some useful regular expressions for quick filtering:

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

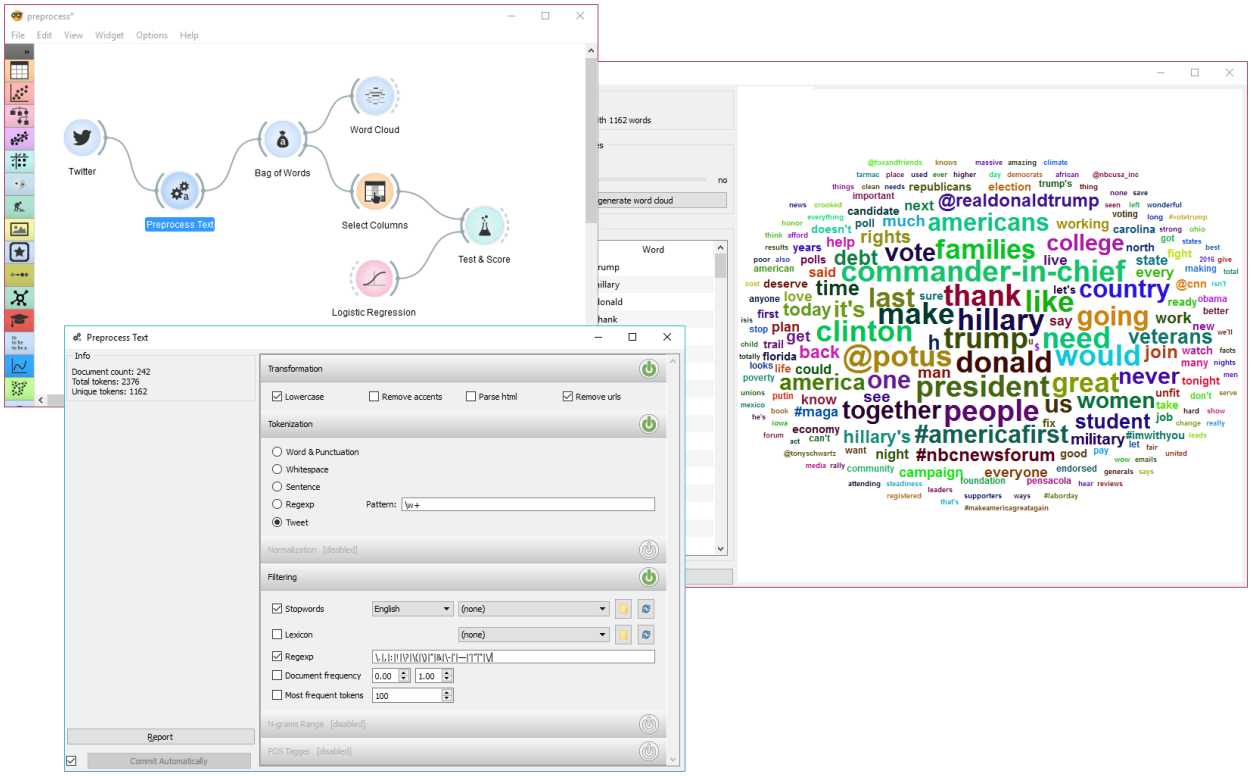
1.8.2 Examples

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



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

total.



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

1.9 Bag of Words

Generates a bag of words from the input corpus.

Inputs

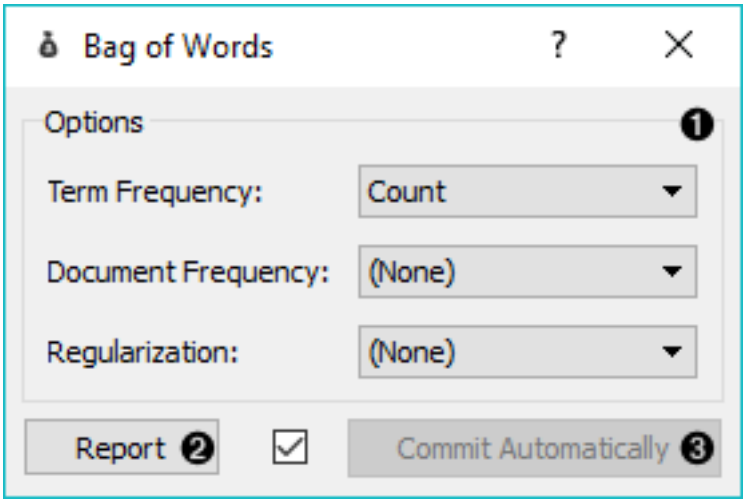
Corpus A collection of documents.

Outputs

Corpus Corpus with bag of words features appended.

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

1. **Parameters for bag of words model:**
    - **Term Frequency:**
        - Count: number of occurences of a word in a document
        - Binary: word appears or does not appear in the document
        - Sublinear: logarithm of term frequency (count)



- **Document Frequency:**
  - (None)
  - IDF: inverse document frequency
  - Smooth IDF: adds one to document frequencies to prevent zero division.
- **Regulariation:**
  - (None)
  - L1 (Sum of elements): normalizes vector length to sum of elements
  - L2 (Euclidean): normalizes vector length to sum of squares

2. Produce a report.
3. If *Commit Automatically* is on, changes are communicated automatically. Alternatively press *Commit*.

1.9.1 Example

In the first example we will simply check how the bag of words model looks like. Load *book-excerpts.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.

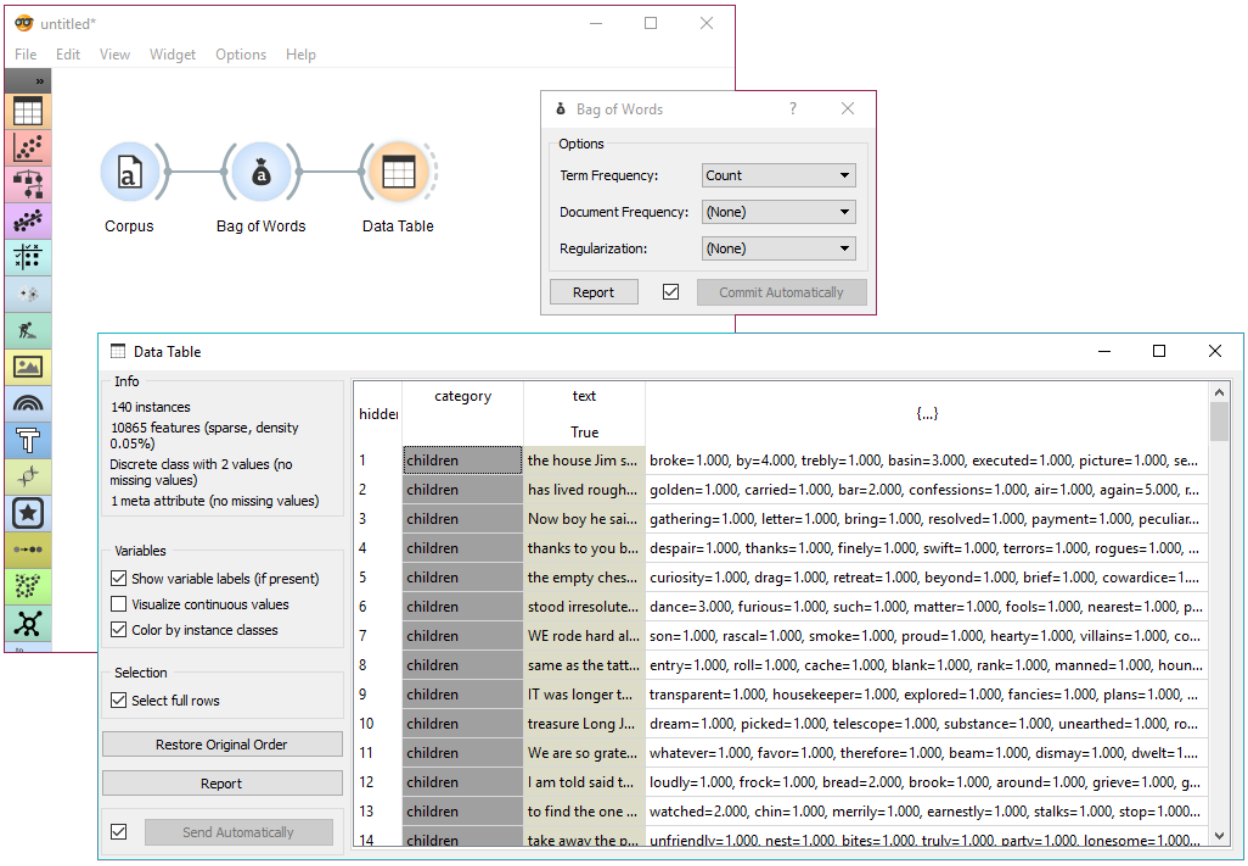
In the second example we will try to predict document category. We are still using the *book-excerpts.tab* data set, which we sent through *Preprocess Text* with default parameters. Then we connected **Preprocess Text** to **Bag of Words** to obtain term frequencies by which we will compute the model.

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

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

1.10 Similarity Hashing

Computes documents hashes.



Inputs

Corpus A collection of documents.

Outputs

Corpus Corpus with simhash value as attributes.

Similarity Hashing is a widget that transforms documents into similarity vectors. The widget uses SimHash from Moses Charikar.

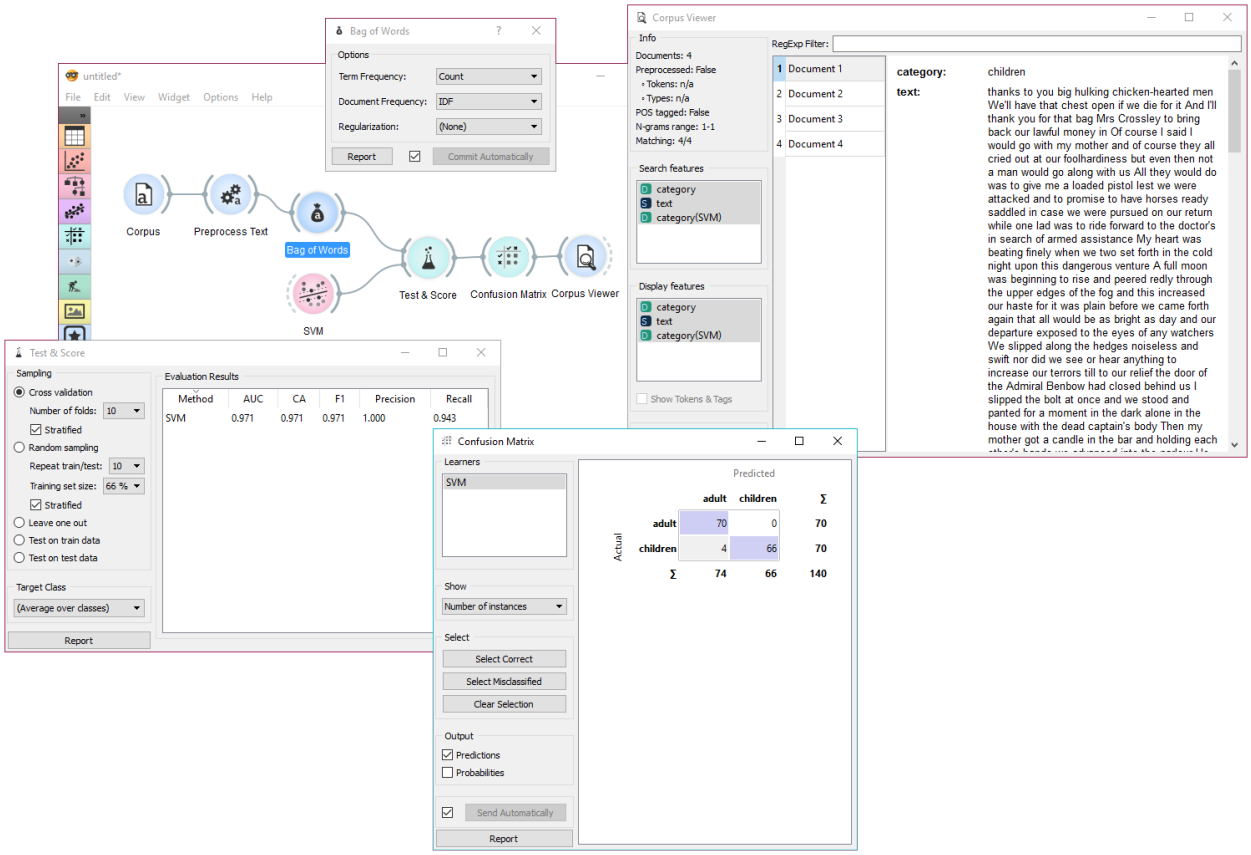
1. Set Simhash size (how many attributes will be on the output, corresponds to bits of information) and shingle lenght (how many tokens are used in a shingle).
2. *Commit Automatically* output the data automatically. Alternatively, press *Commit*.

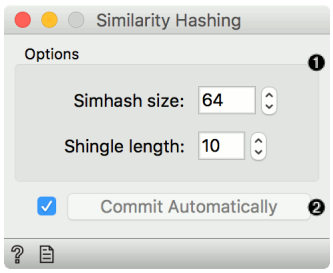
1.10.1 Example

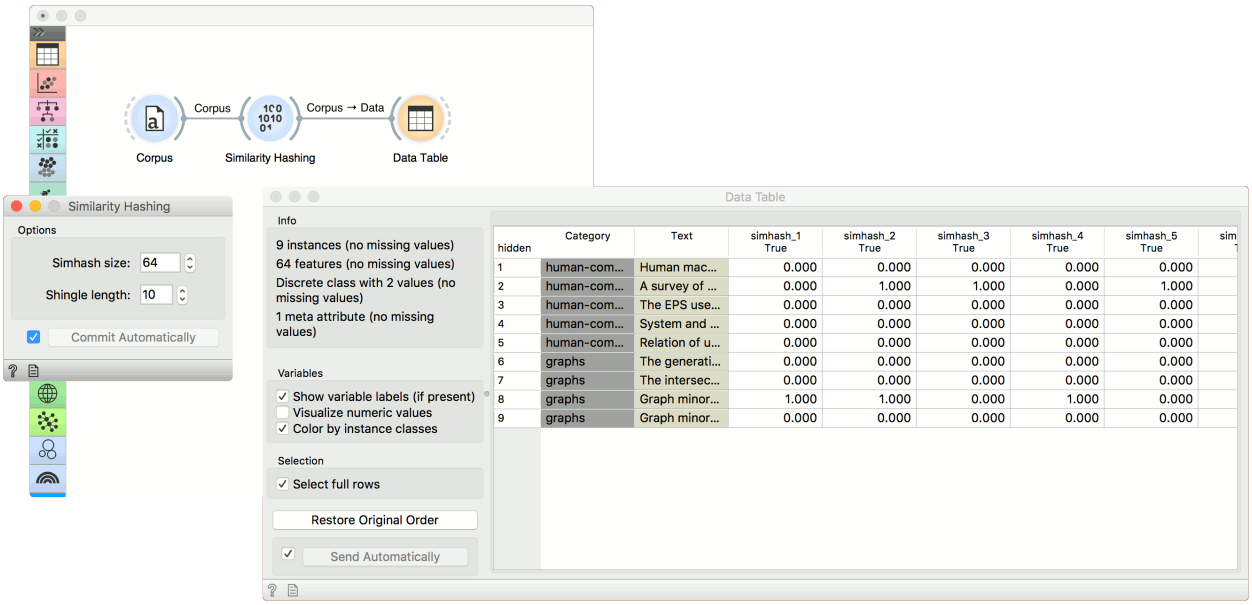
We will use *deerwester.tab* to find similar documents in this small corpus. Load the data with *Corpus* and pass it to **Similarity Hashing**. We will keep the default hash size and shingle length. We can observe what the widget outputs in a **Data Table**. There are 64 new attributes available, corresponding to the *Simhash size* parameter.

1.10.2 References

Charikar, M. (2002) Similarity estimation techniques from rounding algorithms. STOC '02 Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, p. 380-388.


Bag of Words
Options
Term Frequency: Count
Document Frequency: IDF
Regularization: (None)
Report
Commit Automatically
untitled*
File Edit View Widget Options Help
Corpus
Preprocess Text
Bag of Words
SVM
Test & Score
Confusion Matrix
Corpus Viewer
Corpus Viewer
Info
Documents: 4
Preprocessed: False
Tokens: n/a
Types: n/a
POS tagged: False
N-grams range: 1-1
Matching: 4/4
RegExp Filter:
1 Document 1
2 Document 2
3 Document 3
4 Document 4
Search features
category
text
category(SVM)
Display features
Show Tokens & Tags
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
Test & Score
Sampling
Cross validation
Number of folds: 10
Stratified
Random sampling
Repeat train/test: 10
Training set size: 66 %
Stratified
Leave one out
Test on train data
Test on test data
Target Class
(Average over classes)
Report
Evaluation Results
Method AUC CA F1 Precision Recall
SVM 0.971 0.971 0.971 1.000 0.943
Confusion Matrix
Learners
SVM
Show
Number of instances
Select
Select Correct
Select Misclassified
Clear Selection
Output
Predictions
Probabilities
Send Automatically
Report
Predicted
adult children Σ
Actual
adult 70 0 70
children 4 66 70
Σ 74 66 140


Similarity Hashing
Options
Simhash size: 64
Shingle length: 10
Commit Automatically



hidden	Category	Text	simhash_1 True	simhash_2 True	simhash_3 True	simhash_4 True	simhash_5 True
1	human-com...	Human mac...	0.000	0.000	0.000	0.000	0.000
2	human-com...	A survey of ...	0.000	1.000	1.000	0.000	1.000
3	human-com...	The EPS use...	0.000	0.000	0.000	0.000	0.000
4	human-com...	System and ...	0.000	0.000	0.000	0.000	0.000
5	human-com...	Relation of u...	0.000	0.000	0.000	0.000	0.000
6	graphs	The generati...	0.000	0.000	0.000	0.000	0.000
7	graphs	The intersec...	0.000	0.000	0.000	0.000	0.000
8	graphs	Graph minor...	1.000	1.000	0.000	1.000	0.000
9	graphs	Graph minor...	0.000	0.000	0.000	0.000	0.000

1.11 Sentiment Analysis

Predict sentiment from text.

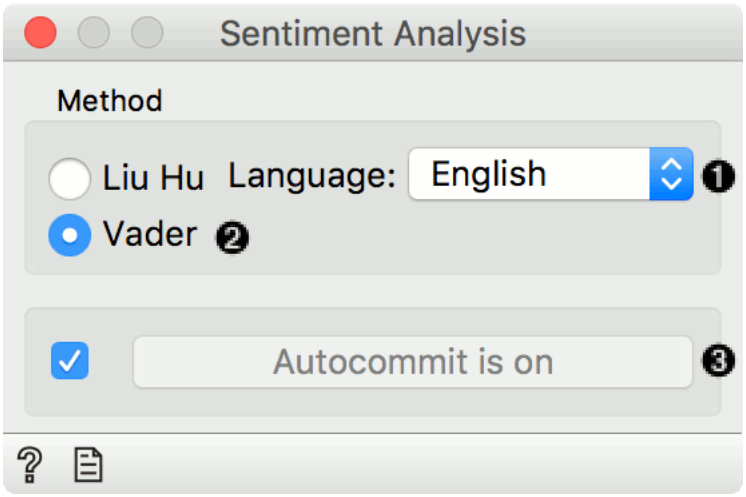
Inputs

Corpus A collection of documents.

Outputs

Corpus A corpus with information on the sentiment of each document.

Sentiment Analysis predicts sentiment for each document in a corpus. It uses Liu Hu and Vader sentiment modules from NLTK. Both of them are lexicon-based. For Liu Hu, you can choose English or Slovenian version.



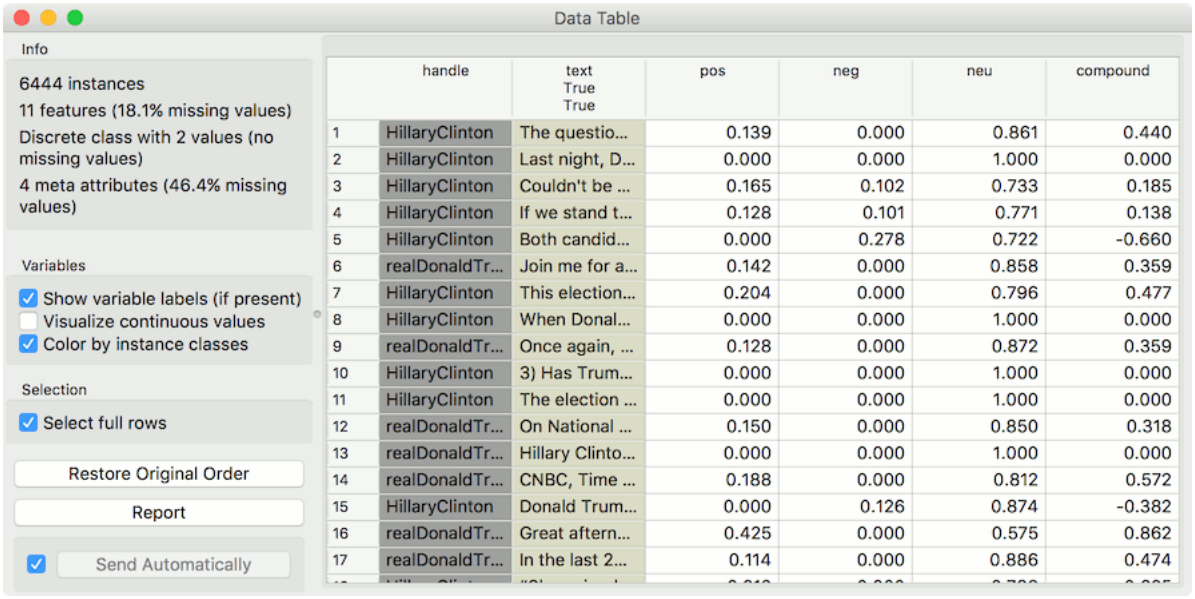
1. ***Method*:**
   - *Liu Hu*: lexicon-based sentiment analysis (supports English and Slovenian)
   - *Vader*: lexicon- and rule-based sentiment analysis

2. Produce a report.
3. If *Auto commit is on*, sentiment-tagged corpus is communicated automatically. Alternatively press *Commit*.

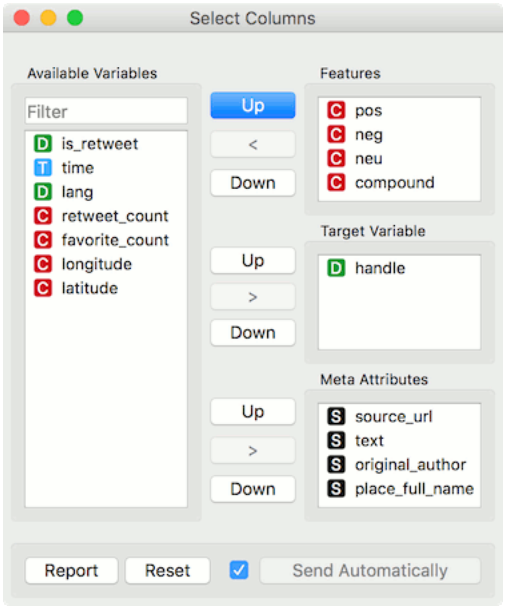
1.11.1 Example

Sentiment Analysis can be used for constructing additional features with sentiment prediction from corpus. First, we load *Election-2016-tweets.tab* in *Corpus*. Then we connect **Corpus** to **Sentiment Analysis**. The widget will append 4 new features for Vader method: positive score, negative score, neutral score and compound (combined score).

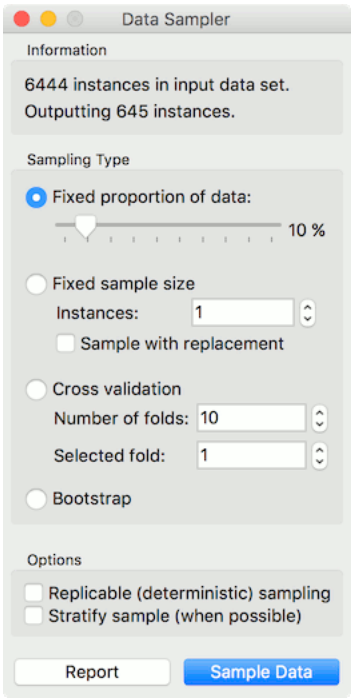
We can observe new features in a **Data Table**, where we sorted the *compound* by score. Compound represents the total sentiment of a tweet, where -1 is the most negative and 1 the most positive.



Now let us visualize the data. We have some features we are currently not interested in, so we will remove them with **Select Columns**.



Then we will make our corpus a little smaller, so it will be easier to visualize. Pass the data to **Data Sampler** and retain a random 10% of the tweets.



Now pass the filtered corpus to **Heat Map**. Use *Merge by k-means* to merge tweets with the same polarity into one line. Then use *Cluster* by *rows* to create a clustered visualization where similar tweets are grouped together. Click on a cluster to select a group of tweets - we selected the negative cluster.

To observe the selected subset, pass the tweets to *Corpus Viewer*.

1.11.2 References

Hutto, C.J. and E. E. Gilbert (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Eighth International Conference on Weblogs and Social Media (ICWSM-14). Ann Arbor, MI, June 2014.

Hu, Minqing and Bing Liu (2004). Mining opinion features in customer reviews. In Proceedings of AAAI Conference on Artificial Intelligence, vol. 4, pp. 755–760. Available online.

Kadunc, Klemen and Marko Robnik-Šikonja (2016). Analiza mnenj s pomočjo strojnega učenja in slovenskega leksikona sentimenta. Conference on Language Technologies & Digital Humanities, Ljubljana (in Slovene). Available online.

1.12 Tweet Profiler

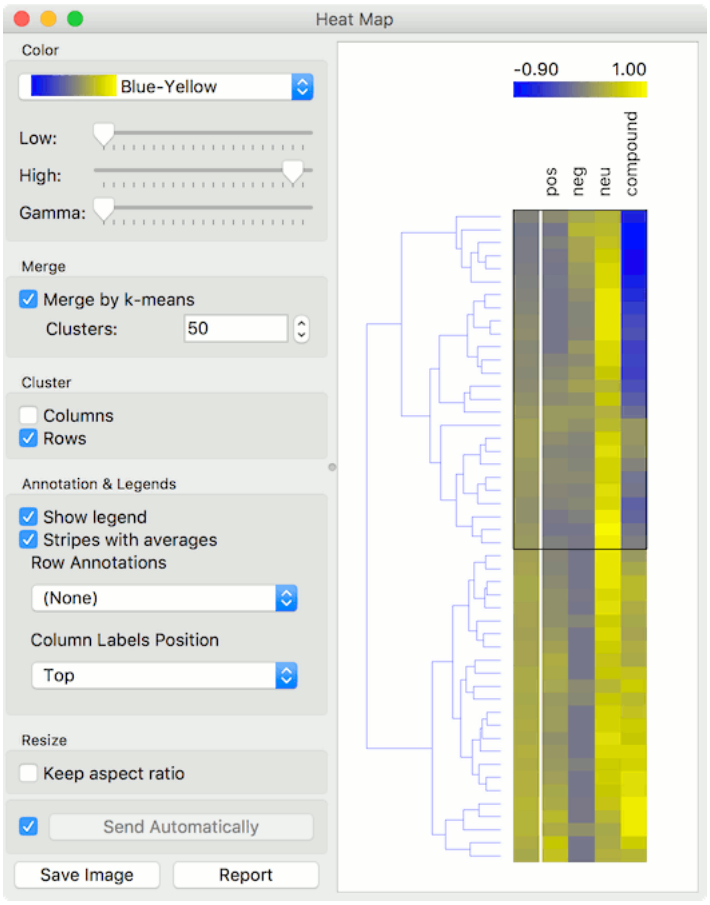
Detect Ekman's, Plutchik's or Profile of Mood States' emotions in tweets.

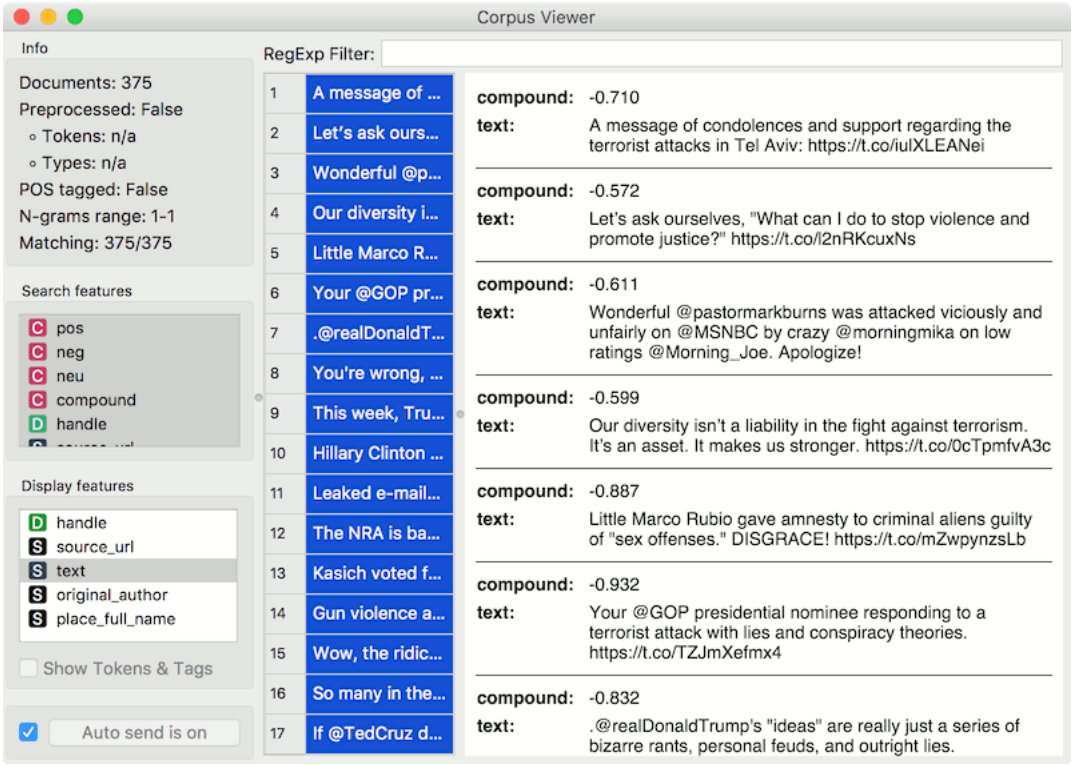
Inputs

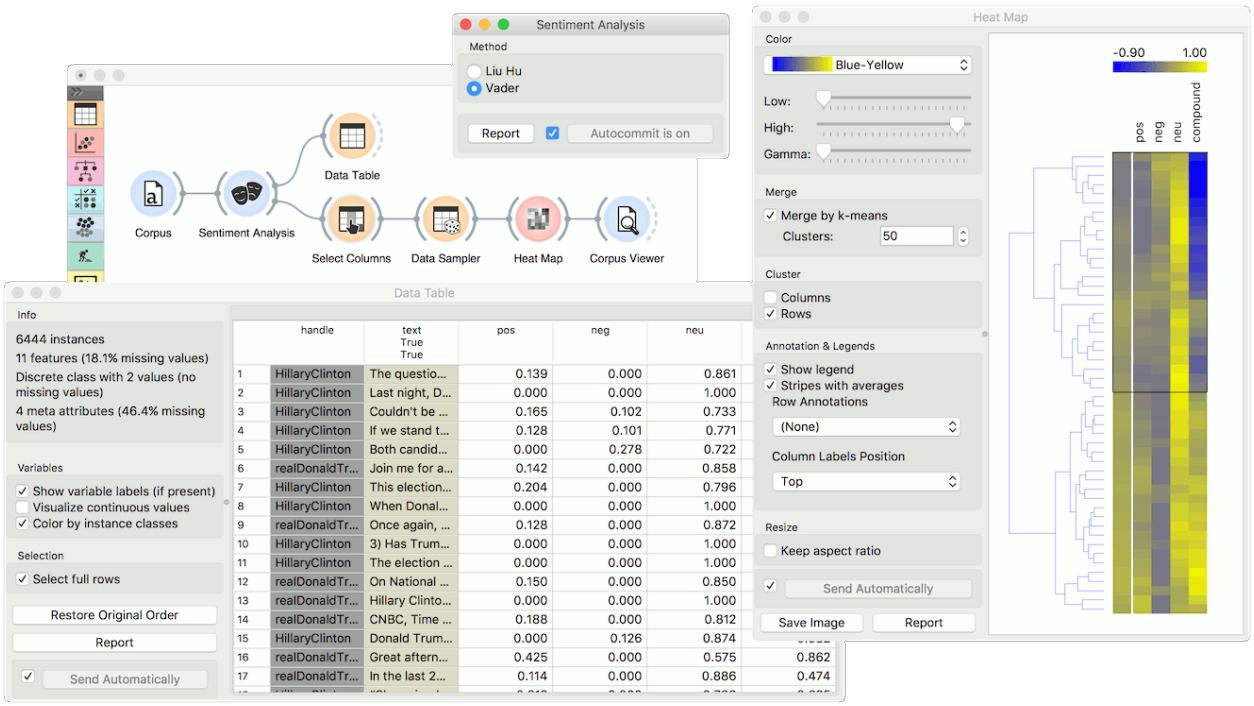
Corpus A collection of tweets (or other documents).

Outputs

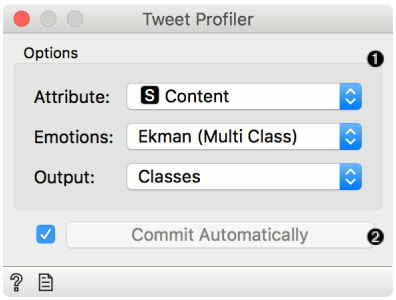
Corpus A corpus with information on the sentiment of each document.


Heat Map
Color
Blue-Yellow
Low:
High:
Gamma:
Merge
Merge by k-means
Clusters:
50
Cluster
Columns
Rows
Annotation & Legends
Show legend
Stripes with averages
Row Annotations
(None)
Column Labels Position
Top
Resize
Keep aspect ratio
Send Automatically
Save Image
Report
-0.90
1.00
pos
neg
neu
compound


Corpus Viewer
Info
Documents: 375
Preprocessed: False
Tokens: n/a
Types: n/a
POS tagged: False
N-grams range: 1-1
Matching: 375/375
Search features
pos
neg
neu
compound
handle
Display features
handle
source_url
text
original_author
place_full_name
Show Tokens & Tags
Auto send is on
RegExp Filter:
1 A message of ...
2 Let's ask ours...
3 Wonderful @p...
4 Our diversity i...
5 Little Marco R...
6 Your @GOP pr...
7 .@realDonaldT...
8 You're wrong, ...
9 This week, Tru...
10 Hillary Clinton ...
11 Leaked e-mail...
12 The NRA is ba...
13 Kasich voted f...
14 Gun violence a...
15 Wow, the ridic...
16 So many in the...
17 If @TedCruz d...
compound: -0.710
text: A message of condolences and support regarding the terrorist attacks in Tel Aviv: https://t.co/iulXLEANei
compound: -0.572
text: Let's ask ourselves, "What can I do to stop violence and promote justice?" https://t.co/l2nRKcuxNs
compound: -0.611
text: Wonderful @pastormarkburns was attacked viciously and unfairly on @MSNBC by crazy @morningmika on low ratings @Morning_Joe. Apologize!
compound: -0.599
text: Our diversity isn't a liability in the fight against terrorism. It's an asset. It makes us stronger. https://t.co/0cTpmfvA3c
compound: -0.887
text: Little Marco Rubio gave amnesty to criminal aliens guilty of "sex offenses." DISGRACE! https://t.co/mZwpynzsLb
compound: -0.932
text: Your @GOP presidential nominee responding to a terrorist attack with lies and conspiracy theories. https://t.co/TZJmXefmx4
compound: -0.832
text: .@realDonaldTrump's "ideas" are really just a series of bizarre rants, personal feuds, and outright lies.



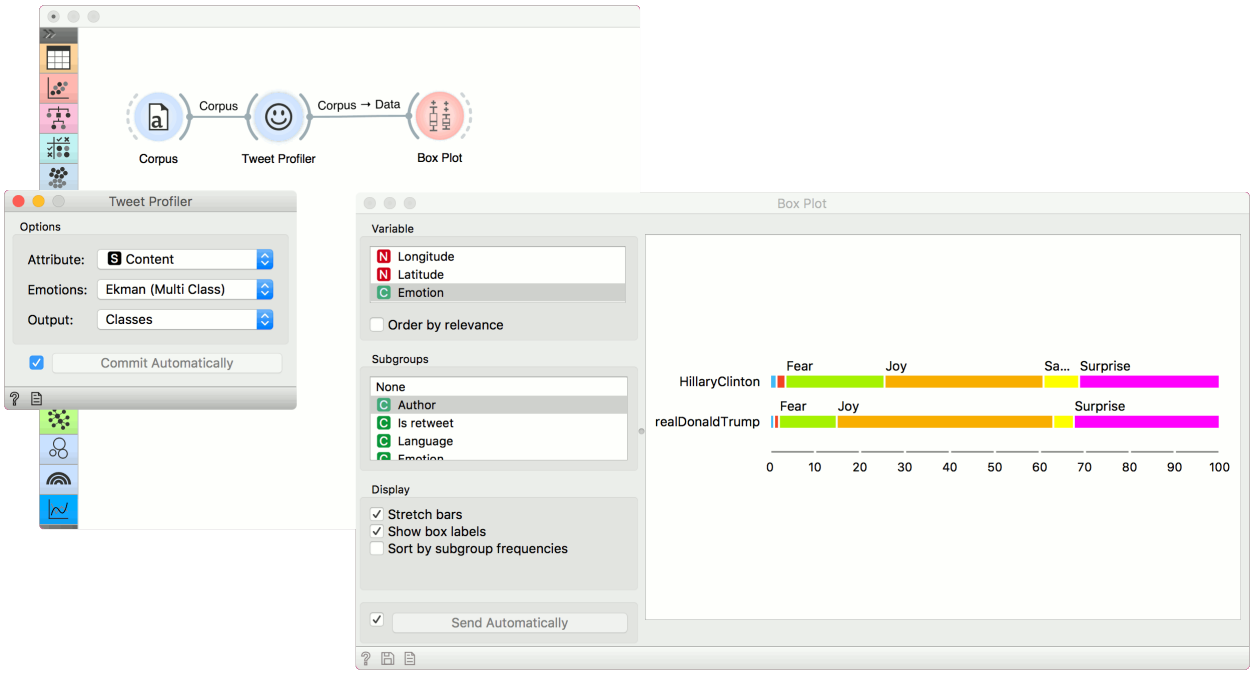
Tweet Profiler retrieves information on sentiment from the server for each given tweet (or document). The widget sends data to the server, where a model computes emotion probabilites and/or scores. The widget support three classifications of emotion, namely Ekman's, Plutchik's and Profile of Mood States (POMS).



1. Options:
   - Attribute to use as content.
   - Emotion classification, either Ekman's, Plutchik's or Profile of Mood States. Multi-class will output one most probable emotion per document, while multi-label will output values in columns per each emotion.
   - The widget can output classes of emotion (categorical), probabilities (numeric), or embeddings (an emotional vector of the document).
2. *Commit Automatically* automatically outputs the result. Alternatively, press *Commit*.

1.12.1 Example

We will use *election-tweets-2016.tab* for this example. Load the data with *Corpus* and connect it to **Tweet Profiler**. We will use *Content* attribute for the analysis, Ekman's classification of emotion with multi-class option and we will output the result as class. We will observe the results in a **Box Plot**. In the widget, we have selected to observe the *Emotion* variable, grouped by *Author*. This way we can see which emotion prevails by which author.



1.12.2 References

Colnerič, Niko and Janez Demšar (2018). Emotion Recognition on Twitter: Comparative Study and Training a Unison Model. In IEEE Transactions on Affective Computing. Available online.

1.13 Topic Modelling

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

Inputs

Corpus A collection of documents.

Outputs

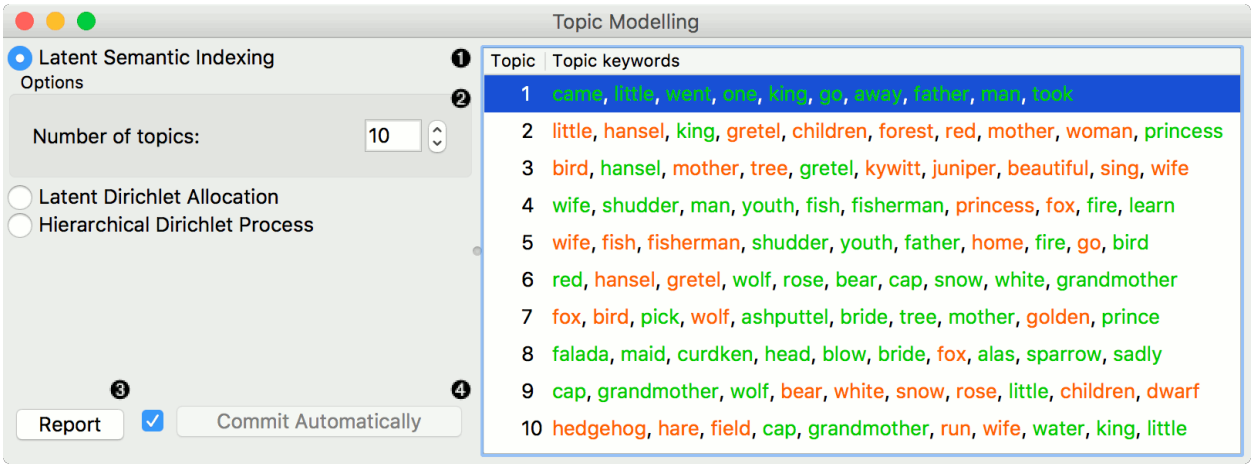
Corpus Corpus with topic weights appended.

Topics Selected topics with word weights.

All Topics Topic weights by tokens.

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

1. **Topic modelling algorithm:**
   - Latent Semantic Indexing
   - Latent Dirichlet Allocation
   - Hierarchical Dirichlet Process
2. **Parameters for the algorithm. LSI and LDA accept only the number of topics modelled, with the default set to 10. HDP, h**



- First level concentration (γ): distribution at the first (corpus) level of Dirichlet Process
- Second level concentration (α): distribution at the second (document) level of Dirichlet Process
- The topic Dirichlet (α): concentration parameter used for the topic draws
- Top level truncation (T): corpus-level truncation (no of topics)
- Second level truncation (K): document-level truncation (no of topics)
- Learning rate (κ): step size
- Slow down parameter (τ)

3. Produce a report.
4. If *Commit Automatically* is on, changes are communicated automatically. Alternatively press *Commit*.

1.13.1 Example

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

LSI provides both positive and negative weights per topic. A positive weight means the word is highly representative of a topic, while a negative weight means the word is highly unrepresentative of a topic (the less it occurs in a text, the more likely the topic). Positive words are colored green and negative words are colored red.

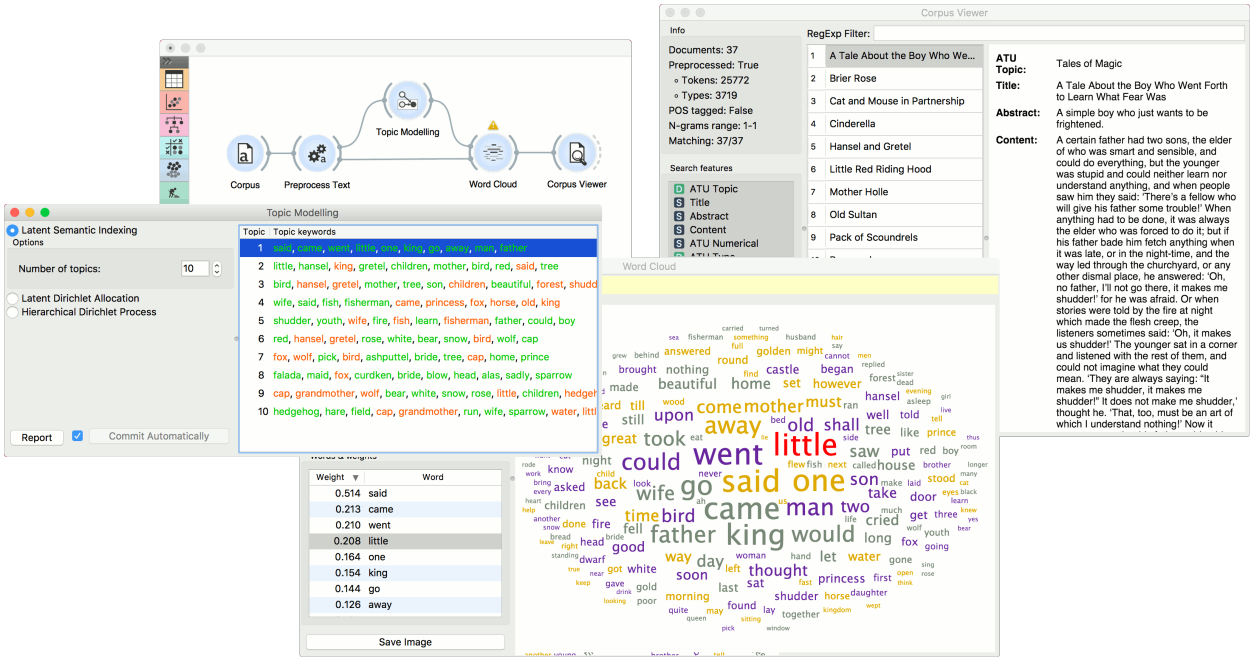
We then select the first topic and display the most frequent words in the topic in *Word Cloud*. We also connected **Preprocess Text** to **Word Cloud** in order to be able to output selected documents. Now we can select a specific word in the word cloud, say *little*. It will be colored red and also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

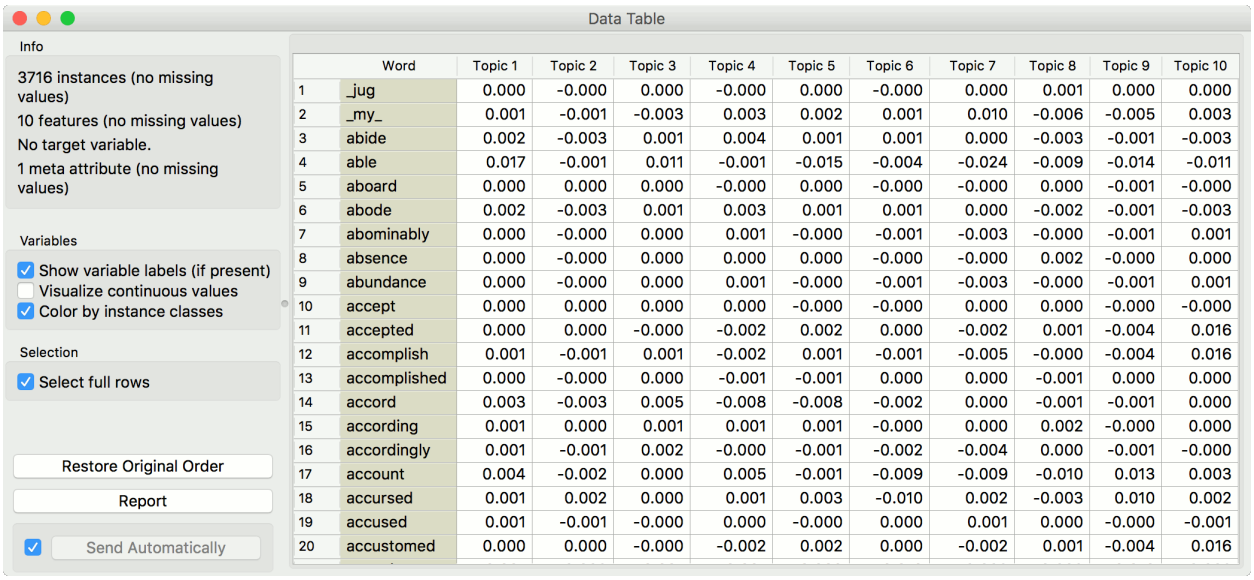
Now we can observe all the documents containing the word *little* in *Corpus Viewer*.

In the second example, we will look at the correlation between topics and words/documents. Connect **Topic Modelling** to **Heat Map**. Ensure the link is set to *All Topics - Data*. **Topic Modelling** will output a matrix of topic weights by words from text (more precisely, tokens).

We can observe the output in a **Data Table**. Tokens are in rows and retrieved topics in colums. Values represent how much a word is represented in a topic.

To visualize this matrix, open **Heat Map**. Select *Merge by k-means* and *Cluster - Rows* to merge similar rows into one and sort them by similarity, which makes the visualization more compact.

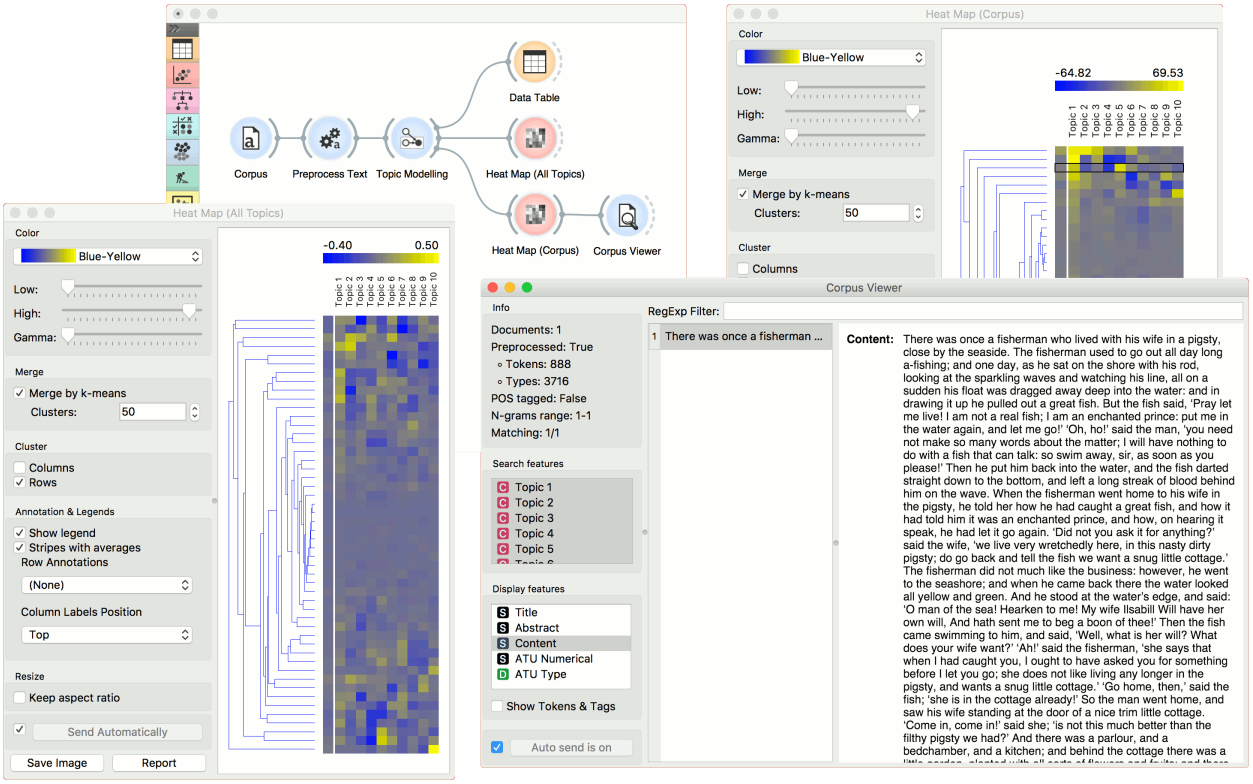

Corpus
Preprocess Text
Topic Modelling
Word Cloud
Corpus Viewer
Topic Modelling
Latent Semantic Indexing
Options
Number of topics: 10
Latent Dirichlet Allocation
Hierarchical Dirichlet Process
Report
Commit Automatically
Topic
Topic keywords
1 said, came, went, little, one, king, go, asked, man, father
2 little, hansel, king, gretel, children, mother, bird, red, said, tree
3 bird, hansel, gretel, mother, tree, son, children, beautiful, forest, shudd
4 wife, said, fish, fisherman, came, princess, fox, horse, old, king
5 shudder, youth, wife, fire, fish, learn, fisherman, father, could, boy
6 red, hansel, gretel, rose, white, bear, snow, bird, wolf, cap
7 fox, wolf, pick, bird, ashputtel, bride, tree, cap, home, prince
8 falada, maid, fox, curdken, bride, blow, head, alas, sadly, sparrow
9 cap, grandmother, wolf, bear, white, snow, rose, little, children, hedgeh
10 hedgehog, hare, field, cap, grandmother, run, wife, sparrow, water, litt
Weight
Word
0.514 said
0.213 came
0.210 went
0.208 little
0.164 one
0.154 king
0.144 go
0.126 away
Save Image
Word Cloud
Corpus Viewer
Info
Documents: 37
Preprocessed: True
Tokens: 25772
Types: 3719
POS tagged: False
N-grams range: 1-1
Matching: 37/37
Search features
ATU Topic
Title
Abstract
Content
ATU Numerical
RegExp Filter:
1 A Tale About the Boy Who We...
2 Brier Rose
3 Cat and Mouse in Partnership
4 Cinderella
5 Hansel and Gretel
6 Little Red Riding Hood
7 Mother Holle
8 Old Sultan
9 Pack of Scoundrels
ATU Topic: Tales of Magic
Title: A Tale About the Boy Who Went Forth to Learn What Fear Was
Abstract: A simple boy who just wants to be frightened.
Content: A certain father had two sons, the elder of who was smart and sensible, and could do everything, but the younger was stupid and could neither learn nor understand anything, and when people saw him they said: 'There's a fellow who will give his father some trouble! When anything had to be done, it was always the elder who was forced to do it; but if his father bade him fetch anything when it was late, or in the night-time, and the way led through the churchyard, or any other dismal place, he answered: 'Oh, no father, I'll not go there, it makes me shudder!' for he was afraid. Or when stories were told by the fire at night which made the flesh creep, the listeners sometimes said: 'Oh, it makes us shudder!' The younger sat in a corner and listened with the rest of them, and could not imagine what they could mean. 'They are always saying: "It makes me shudder, it makes me shudder!" It does not make me shudder,' thought he. 'That, too, must be an art of which I understand nothing!' Now it


Data Table
Info
3716 instances (no missing values)
10 features (no missing values)
No target variable.
1 meta attribute (no missing values)
Variables
Show variable labels (if present)
Visualize continuous values
Color by instance classes
Selection
Select full rows
Restore Original Order
Report
Send Automatically
Word Topic 1 Topic 2 Topic 3 Topic 4 Topic 5 Topic 6 Topic 7 Topic 8 Topic 9 Topic 10
1 _jug 0.000 -0.000 0.000 -0.000 0.000 -0.000 0.000 0.001 0.000 0.000
2 _my_ 0.001 -0.001 -0.003 0.003 0.002 0.001 0.010 -0.006 -0.005 0.003
3 abide 0.002 -0.003 0.001 0.004 0.001 0.001 0.000 -0.003 -0.001 -0.003
4 able 0.017 -0.001 0.011 -0.001 -0.015 -0.004 -0.024 -0.009 -0.014 -0.011
5 aboard 0.000 0.000 0.000 -0.000 0.000 -0.000 -0.000 0.000 -0.001 -0.000
6 abode 0.002 -0.003 0.001 0.003 0.001 0.001 0.000 -0.002 -0.001 -0.003
7 abominably 0.000 -0.000 0.000 0.001 -0.000 -0.001 -0.003 -0.000 -0.001 0.001
8 absence 0.000 -0.000 0.000 0.000 0.000 -0.000 -0.000 0.002 -0.000 0.000
9 abundance 0.000 -0.000 0.000 0.001 -0.000 -0.001 -0.003 -0.000 -0.001 0.001
10 accept 0.000 0.000 0.000 0.000 -0.000 -0.000 0.000 0.000 -0.000 -0.000
11 accepted 0.000 0.000 -0.000 -0.002 0.002 0.000 -0.002 0.001 -0.004 0.016
12 accomplish 0.001 -0.001 0.001 -0.002 0.001 -0.001 -0.005 -0.000 -0.004 0.016
13 accomplished 0.000 -0.000 0.000 -0.001 -0.001 0.000 0.000 -0.001 0.000 0.000
14 accord 0.003 -0.003 0.005 -0.008 -0.008 -0.002 0.000 -0.001 -0.001 0.000
15 according 0.001 0.000 0.001 0.001 0.001 -0.000 0.000 0.002 -0.000 0.000
16 accordingly 0.001 -0.001 0.002 -0.000 -0.001 -0.002 -0.004 0.000 -0.001 -0.000
17 account 0.004 -0.002 0.000 0.005 -0.001 -0.009 -0.009 -0.010 0.013 0.003
18 accursed 0.001 0.002 0.000 0.001 0.003 -0.010 0.002 -0.003 0.010 0.002
19 accused 0.001 -0.001 -0.000 0.000 -0.000 0.000 0.001 0.000 -0.000 -0.001
20 accustomed 0.000 0.000 -0.000 -0.002 0.002 0.000 -0.002 0.001 -0.004 0.016

In the upper part of the visualization, we have words that highly define topics 1-3 and in the lower part those that define topics 5 and 10.

We can similarly observe topic representation across documents. We connect another **Heat Map** to **Topic Modelling** and set link to *Corpus - Data*. We set *Merge* and *Cluster* as above.

In this visualization we see how much is a topic represented in a document. Looks like Topic 1 is represented almost across the entire corpus, while other topics are more specific. To observe a specific set of document, select either a clustering node or a row in the visualization. Then pass the data to *Corpus Viewer*.



1.14 Corpus Viewer

Displays corpus content.

Inputs

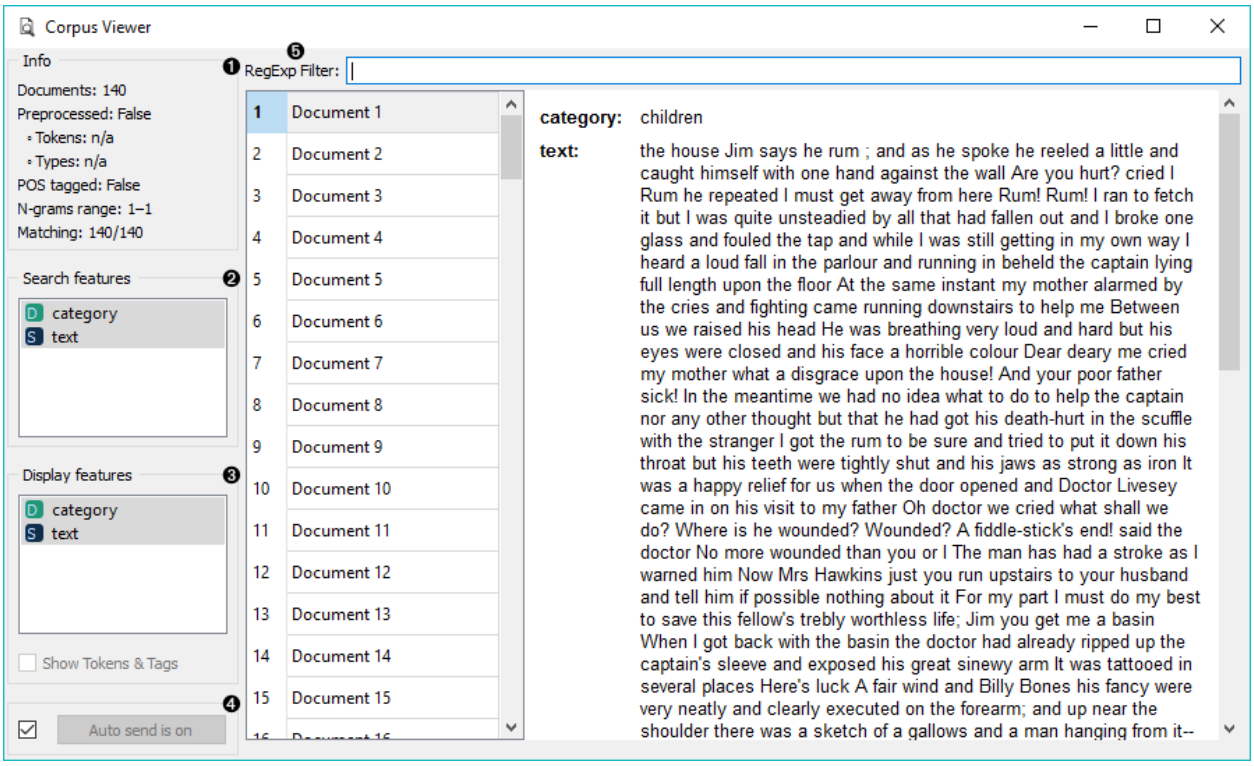
Corpus A collection of documents.

Outputs

Corpus Documents containing the queried word.

Corpus Viewer is meant for viewing text files (instances of *Corpus*). It will always output an instance of corpus. If *RegExp* filtering is used, the widget will output only matching documents.

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



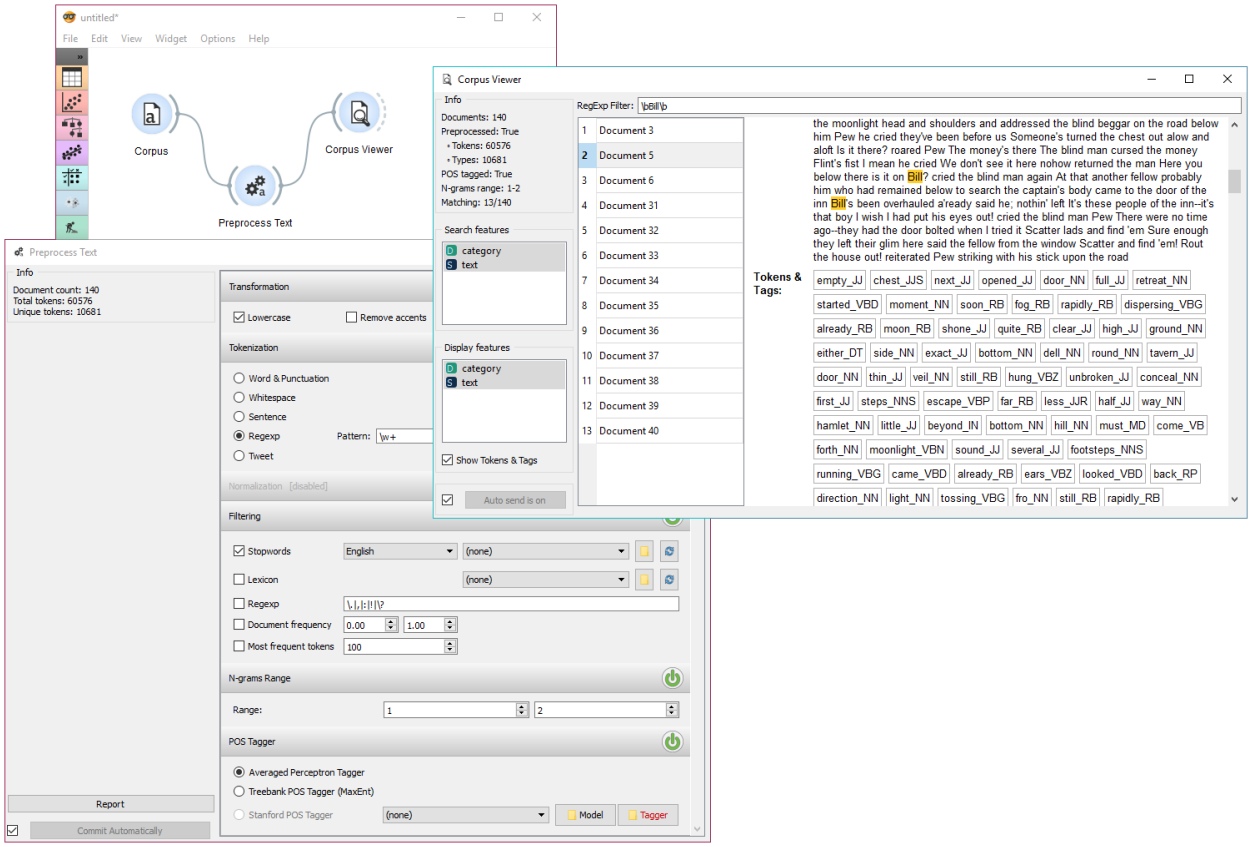
- *POS tagged*: if POS tags are on the input, the result is True, else False.
- *N-grams range*: if N-grams are set in *Preprocess Text*, results are reported, default is 1-1 (one-grams).
- *Matching*: number of documents matching the *RegExp Filter*. All documents are output by default.

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

1.14.1 Example

Corpus Viewer can be used for displaying all or some documents in corpus. In this example, we will first load *book-excerpts.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 *\bBil\b* to find the documents containing only the word Bill. You can output matching or non-matching documents, view them in another *Corpus Viewer* or further analyse them.



1.15 Word Cloud

Generates a word cloud from corpus.

Inputs

Topic Selected topic.

Corpus A collection of documents.

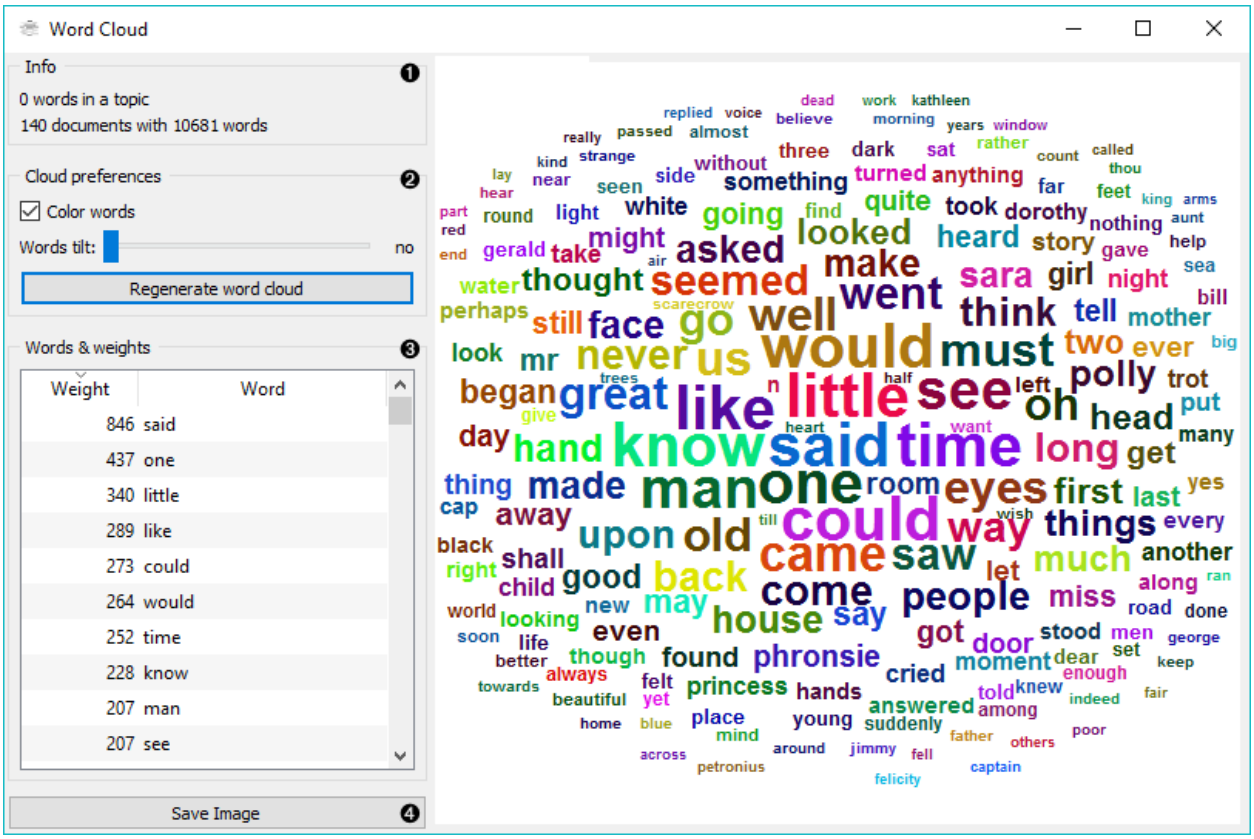
Outputs

Corpus Documents that match the selection.

Word Selected word that can be used as query in *Concordance*.

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

1. **Information on the input.**
   - number of words (tokens) in a topic
   - number of documents and tokens in the corpus
2. **Adjust the plot.**
   - If *Color words* is ticked, words will be assigned a random color. If unchecked, the words will be black.



- *Word tilt* adjust the tilt of words. The current state of tilt is displayed next to the slider ('no' is the default).
- *Regenerate word cloud* plot the cloud anew.

3. Words & weights displays a sorted list of words (tokens) by their frequency in the corpus or topic. Clicking on a word will select that same word in the cloud and output matching documents. Use *Ctrl* to select more than one word. Documents matching ANY of the selected words will be on the output (logical OR).
4. *Save Image* saves the image to your computer in a .svg or .png format.

1.15.1 Example

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

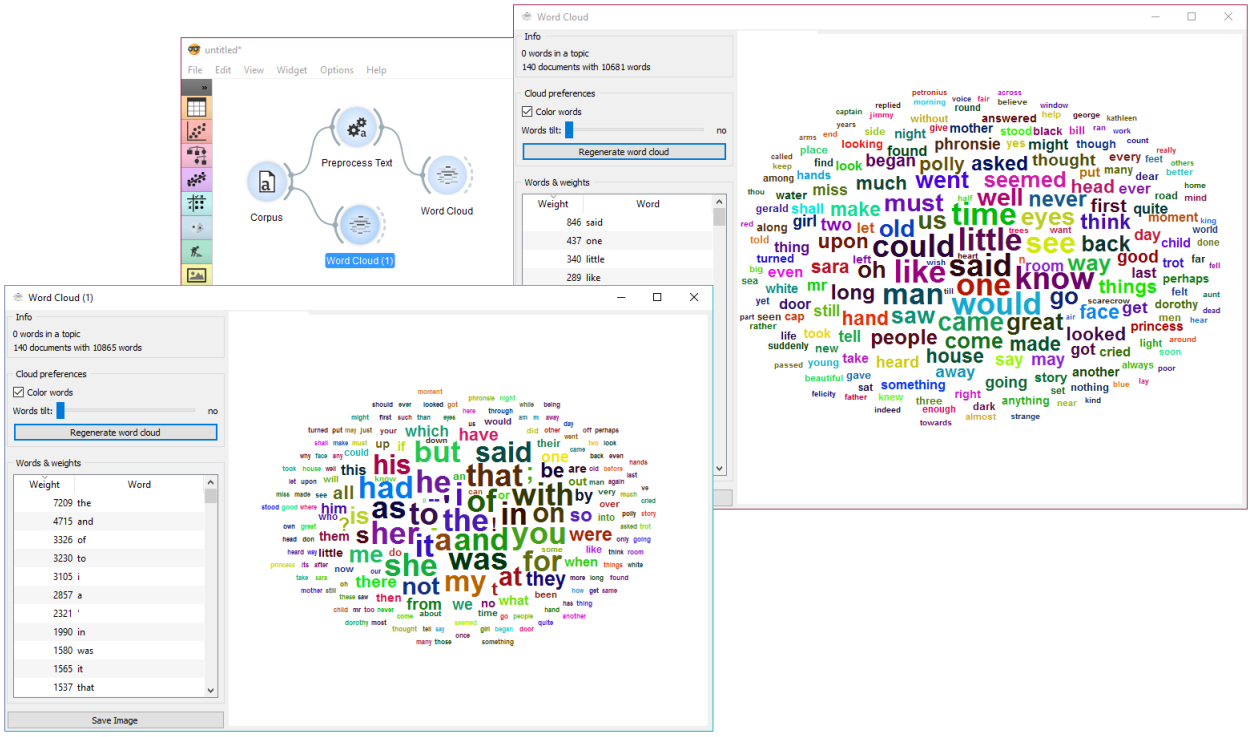
Use *Corpus* to load the data. Connect *Preprocess Text* to it and set your parameters. We've used defaults here, just to see the difference between the default preprocessing in the **Word Cloud** widget and the **Preprocess Text** widget.

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

1.16 Concordance

Display the context of the word.

Inputs



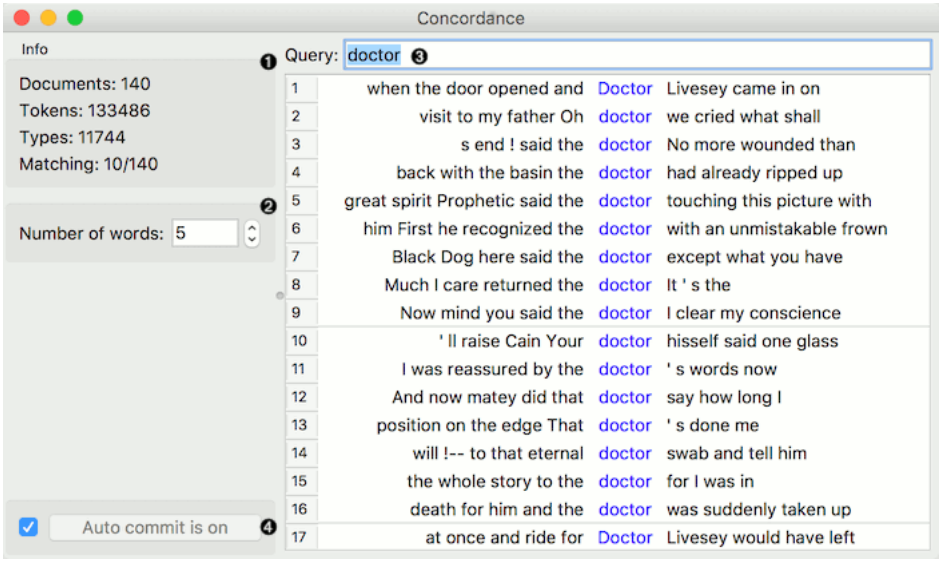
Corpus A collection of documents.

Outputs

Selected Documents Documents containing the queried word.

Concordances A table of concordances.

Concordance finds the queried word in a text and displays the context in which this word is used. Results in a single color come from the same document. The widget can output selected documents for further analysis or a table of concordances for the queried word. Note that the widget finds only exact matches of a word, which means that if you query the word 'do', the word 'doctor' won't appear in the results.

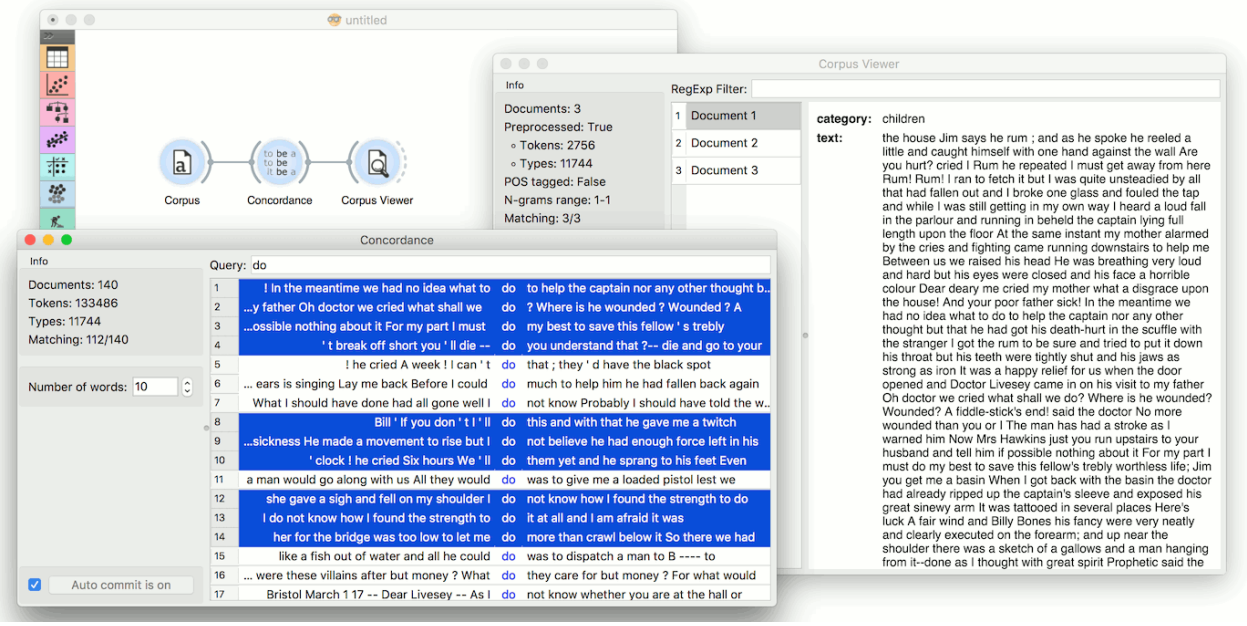


1. ***Information*:**
    - *Documents*: number of documents on the input.
    - *Tokens*: number of tokens on the input.
    - *Types*: number of unique tokens on the input.
    - *Matching*: number of documents containing the queried word.
2. *Number of words*: the number of words displayed on each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

1.16.1 Examples

Concordance can be used for displaying word contexts in a corpus. First, we load *book-excerpts.tab* in *Corpus*. Then we connect **Corpus** to **Concordance** and search for concordances of a word "doctor". The widget displays all documents containing the word "doctor" together with their surrounding (contextual) words.

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

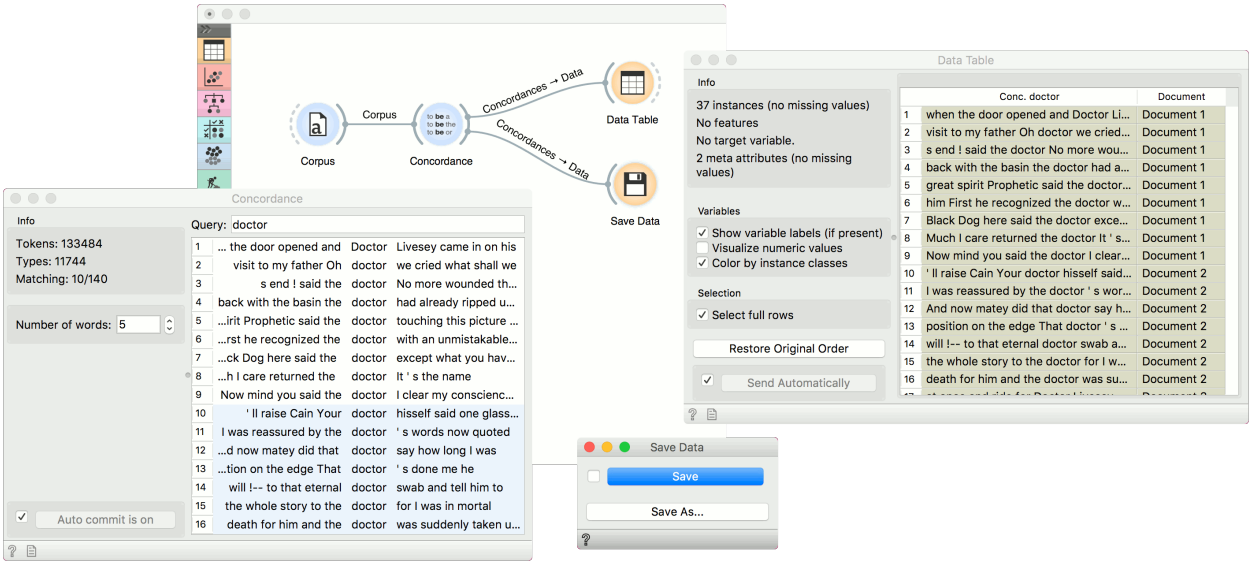


In the second example, we will output concordances instead. We will keep the *book-excerpts.tab* in *Corpus* and the connection to **Concordance**. Our queried word remain "doctor".

This time, we will connect **Data Table** to **Concordance** and select Concordances output instead. In the **Data Table**, we get a list of concordances for the queried word and the corresponding documents. Now, we will save this table with **Save Data** widget, so we can use it in other projects or for further analysis.

1.17 GeoMap

Displays geographic distribution of data.



Inputs

Data Data set.

Outputs

Corpus Documents containing mentions of selected geographical regions.

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

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

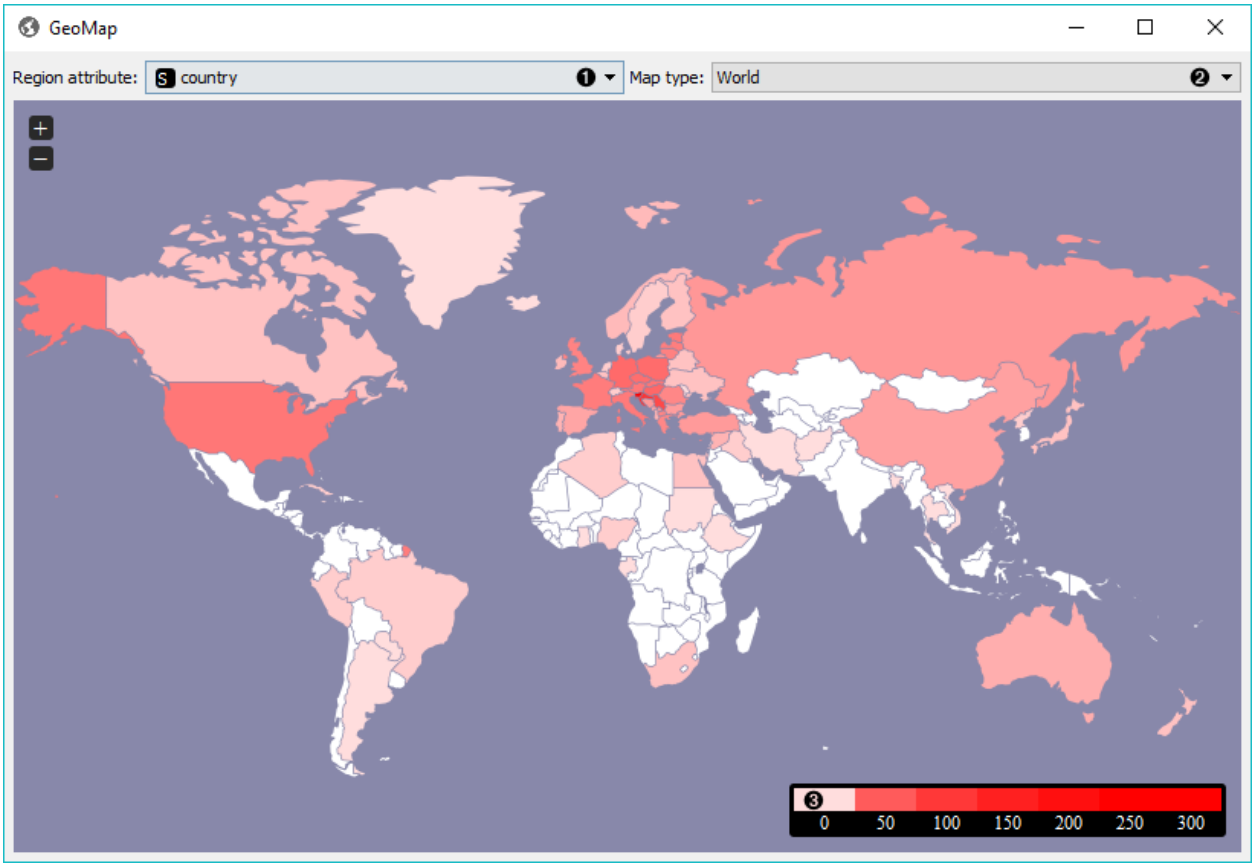
To select documents mentioning a specific country, click on a country and the widget will output matching documents. To select more than one country hold Ctrl/Cmd upon selection.

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

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



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

1.18 Word Enrichment

Word enrichment analysis for selected documents.

Inputs

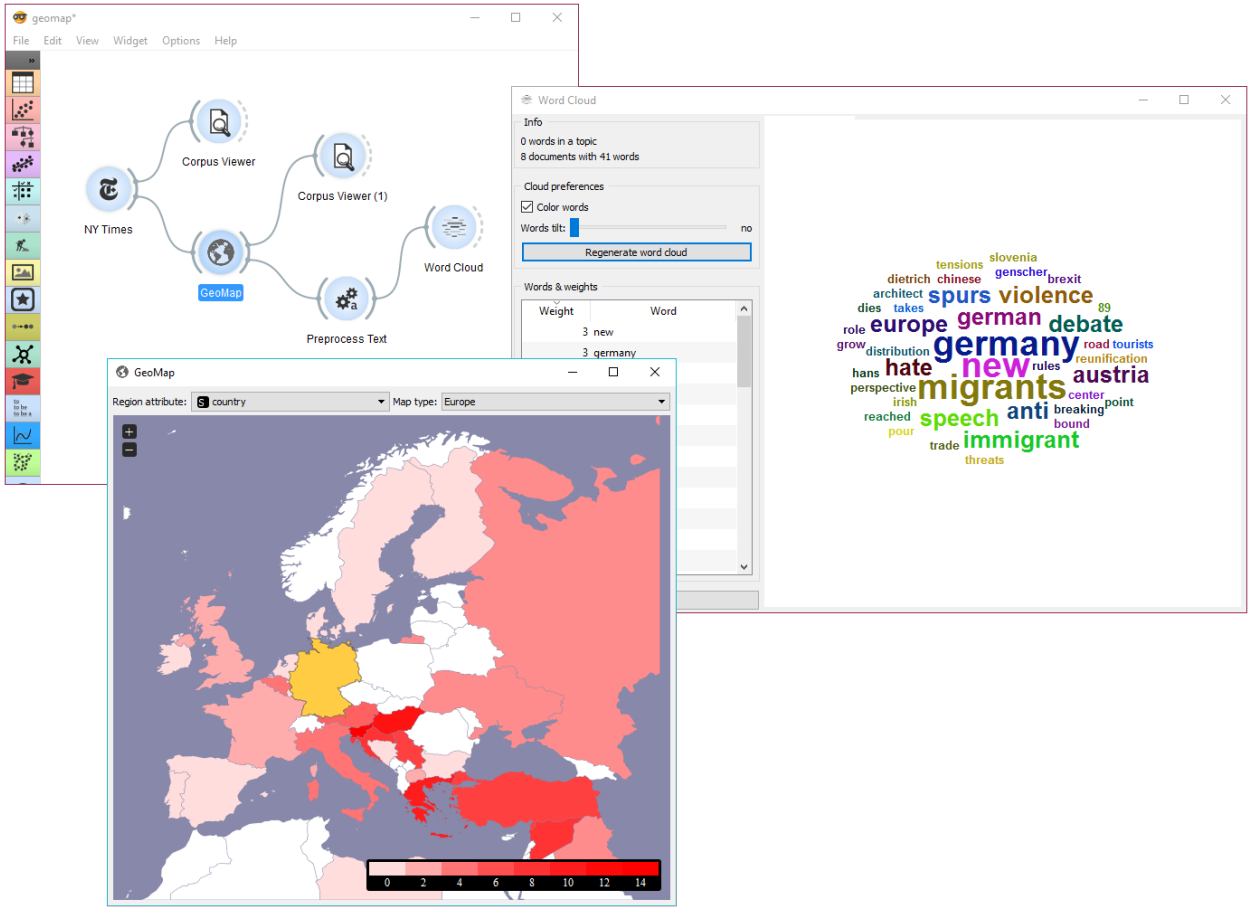
Corpus A collection of documents.

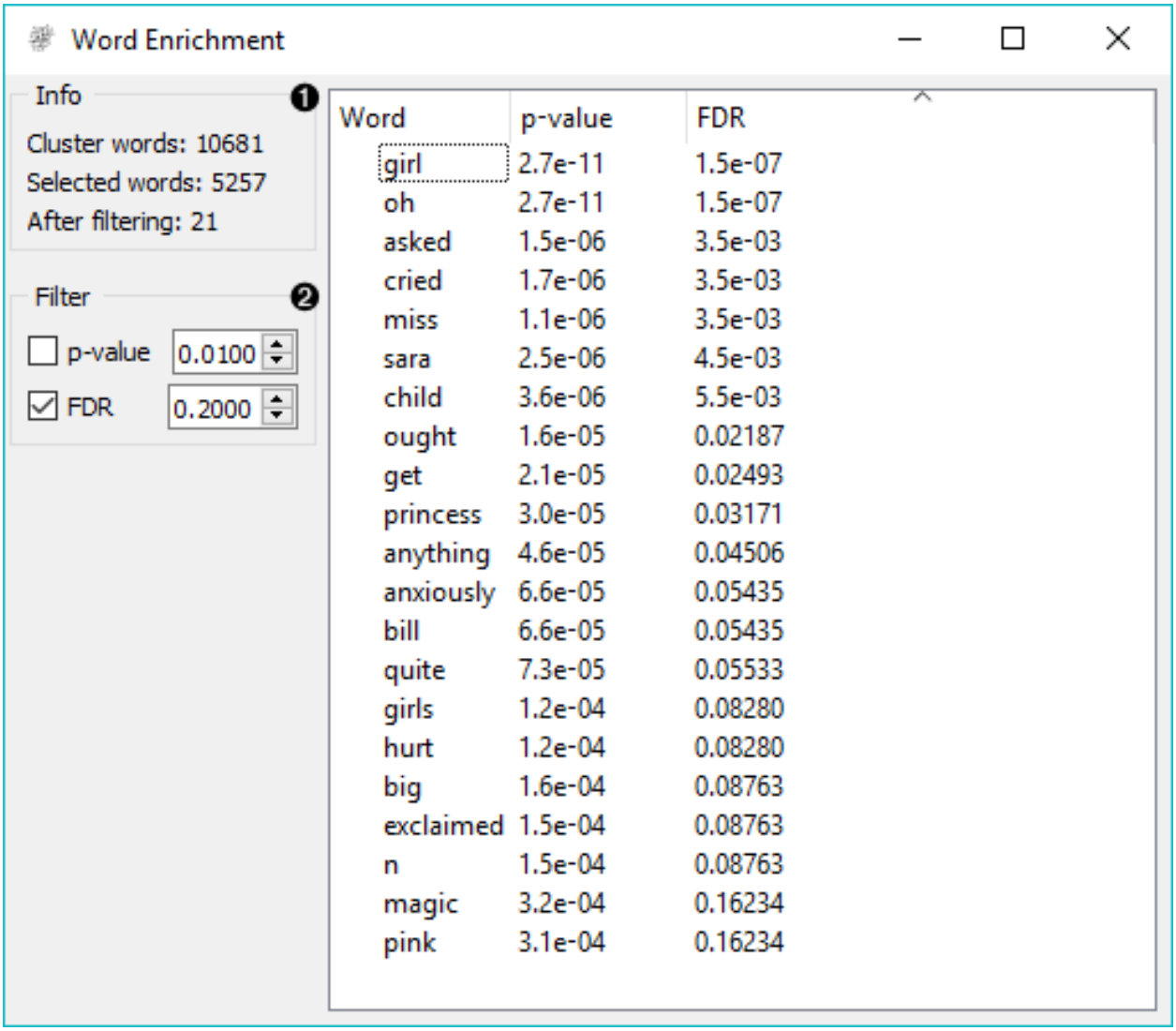
Selected Data Selected instances from corpus.

Outputs None

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:**

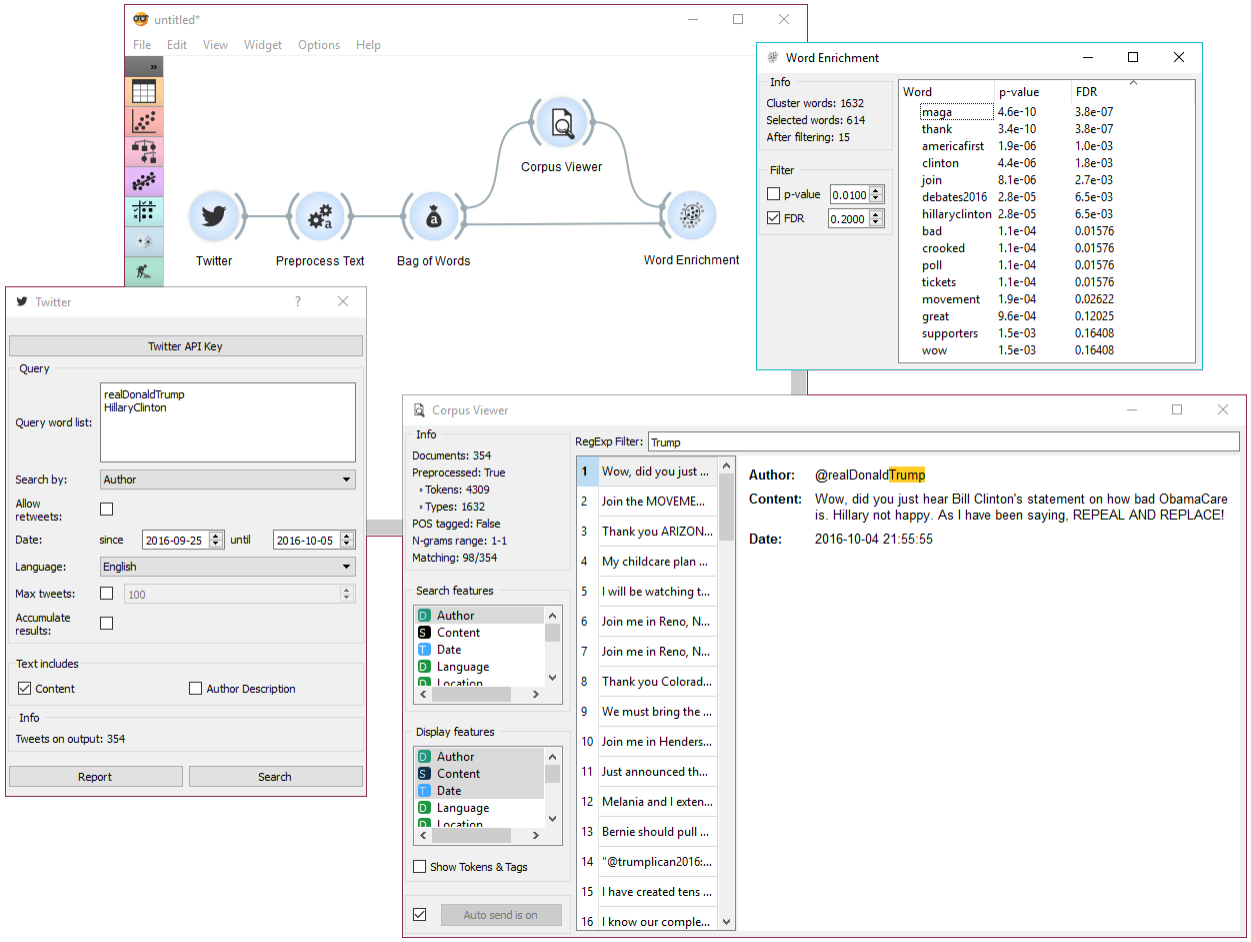

geomap*
File Edit View Widget Options Help
Corpus Viewer
Corpus Viewer (1)
NY Times
GeoMap
Word Cloud
Preprocess Text
Word Cloud
Info
0 words in a topic
8 documents with 41 words
Cloud preferences
Color words
Words tilt:
no
Regenerate word cloud
Words & weights
Weight
Word
3 new
3 germany
tensions slovenia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


Word Enrichment
Info
1
Cluster words: 10681
Selected words: 5257
After filtering: 21
Filter
2
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

- p-value
- false discovery rate (FDR)

1.18.1 Example

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



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

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

1.19 Duplicate Detection

Detect & remove duplicates from a corpus.

Inputs

Distances A distance matrix.

Outputs

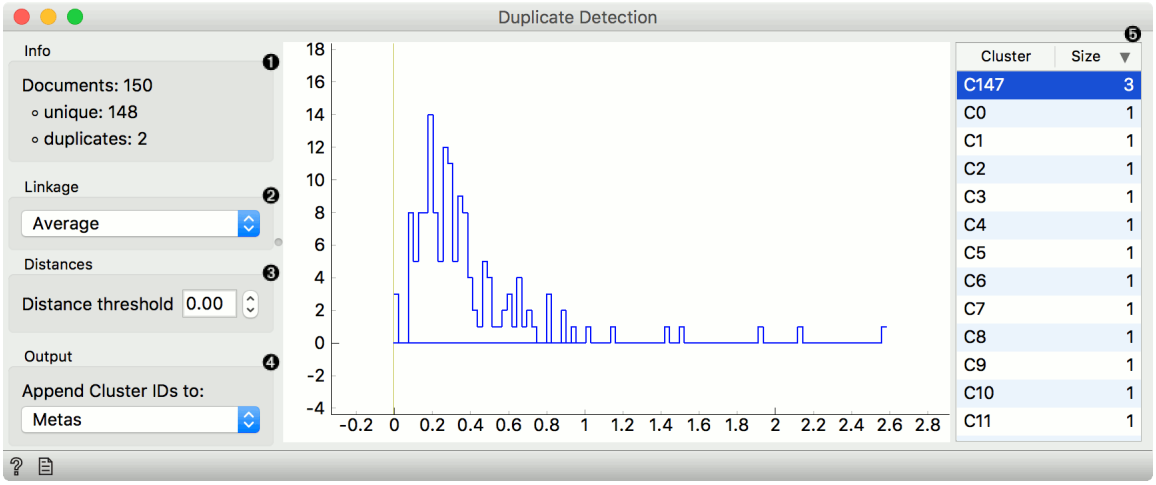
Corpus Without Duplicated Corpus with duplicates removed.

Duplicates Cluster Documents belonging to selected cluster.

Corpus Corpus with appended cluster labels.

Duplicate Detection uses clustering to find duplicates in the corpus. It is great with the *Twitter* widget for removing retweets and other similar documents.

To set the level of similarity, drag the line vertical line left or right in the visualization. The further left the line, the more similar the documents have to be in order to be considered duplicates. You can also set the threshold manually in the control area.



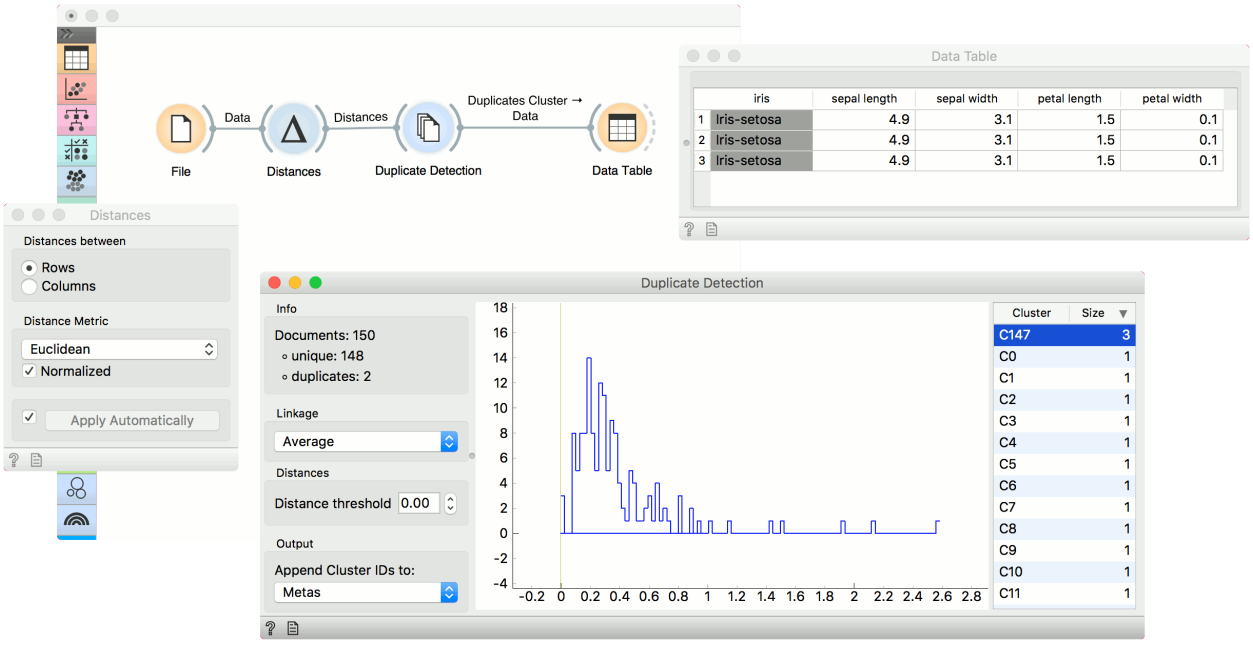
1. Information on unique and duplicate documents.
2. Linkage used for clustering (Single, Average, Complete, Weighted and Ward).
3. Distance threshold sets the similarity cutoff. The lower the value, the more similar the data instances have to be to belong to the same cluster. You can also set the cutoff by dragging the vertical line in the plot.
4. Cluster labels can be appended as attributes, class or metas.
5. List of clusters at the selected threshold. They are sorted by size by default. Click on the cluster to observe its content on the output.

1.19.1 Example

This simple example uses *iris* data to find identical data instances. Load *iris* with the **File** widget and pass it to **Distances**. In **Distances**, use Euclidean distance for computing the distance matrix. Pass distances to **Duplicate Detection**.

It looks like cluster C147 contain three duplicate entries. Let us select it in the widget and observe it in a **Data Table**. Remember to set the output to *Duplicates Cluster*. IThe three data instances are identical. To use the data set without duplicates, use the first output, *Corpus Without Duplicates*.

The same procedure can be used also for corpora. Remember to use the *Bag of Words* between *Corpus* and **Distances**.


File
Data
Distances
Distances
Duplicate Detection
Duplicates Cluster → Data
Data Table
Data Table
iris
sepal length
sepal width
petal length
petal width
1 Iris-setosa 4.9 3.1 1.5 0.1
2 Iris-setosa 4.9 3.1 1.5 0.1
3 Iris-setosa 4.9 3.1 1.5 0.1
Distances
Distances between
Rows
Columns
Distance Metric
Euclidean
Normalized
Apply Automatically
Duplicate Detection
Info
Documents: 150
unique: 148
duplicates: 2
Linkage
Average
Distances
Distance threshold 0.00
Output
Append Cluster IDs to:
Metas
Cluster
Size
C147 3
C0 1
C1 1
C2 1
C3 1
C4 1
C5 1
C6 1
C7 1
C8 1
C9 1
C10 1
C11 1

CHAPTER 2

Scripting

CHAPTER 3

Indices and tables

- genindex
- modindex
- search