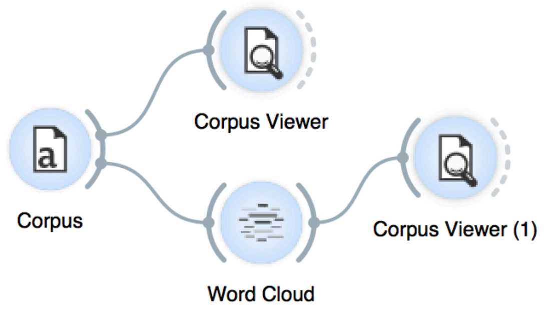
Lesson 1: Corpus

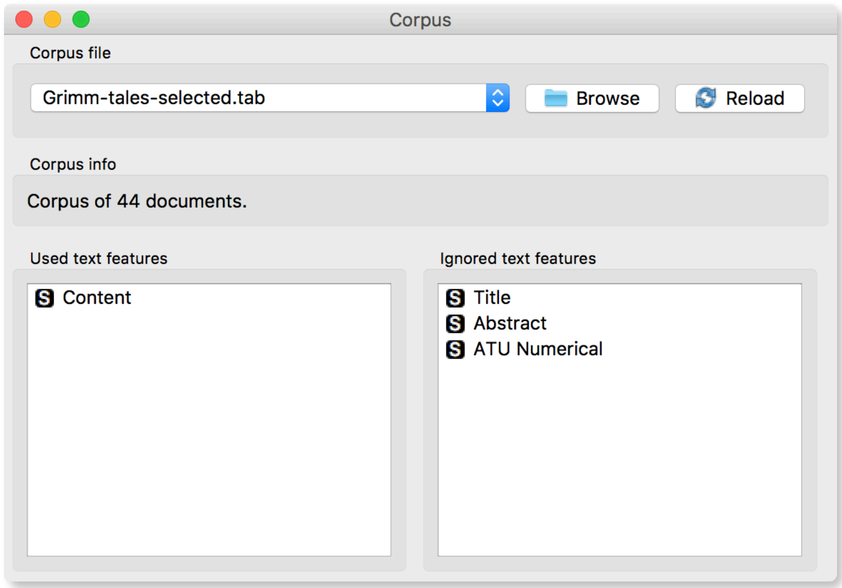
Start by constructing a workflow that consists of a Corpus widget, a Word Cloud widget and two Corpus Viewer widgets:

Corpus is any collection of documents.



Orange3-Text comes with several preloaded data sets. From these ("Browse documentation data sets…") choose *Grimm-tales-selected.tab*, a data set containing Grimm's selected tales.

The particularity of the Corpus widget is that it sets the text feature(s) to apply text mining on. "Used text features" defines the content (text), while other columns contain meta attributes (title, abstract, etc.).



Corpus file: Grimm-tales-selected.tab

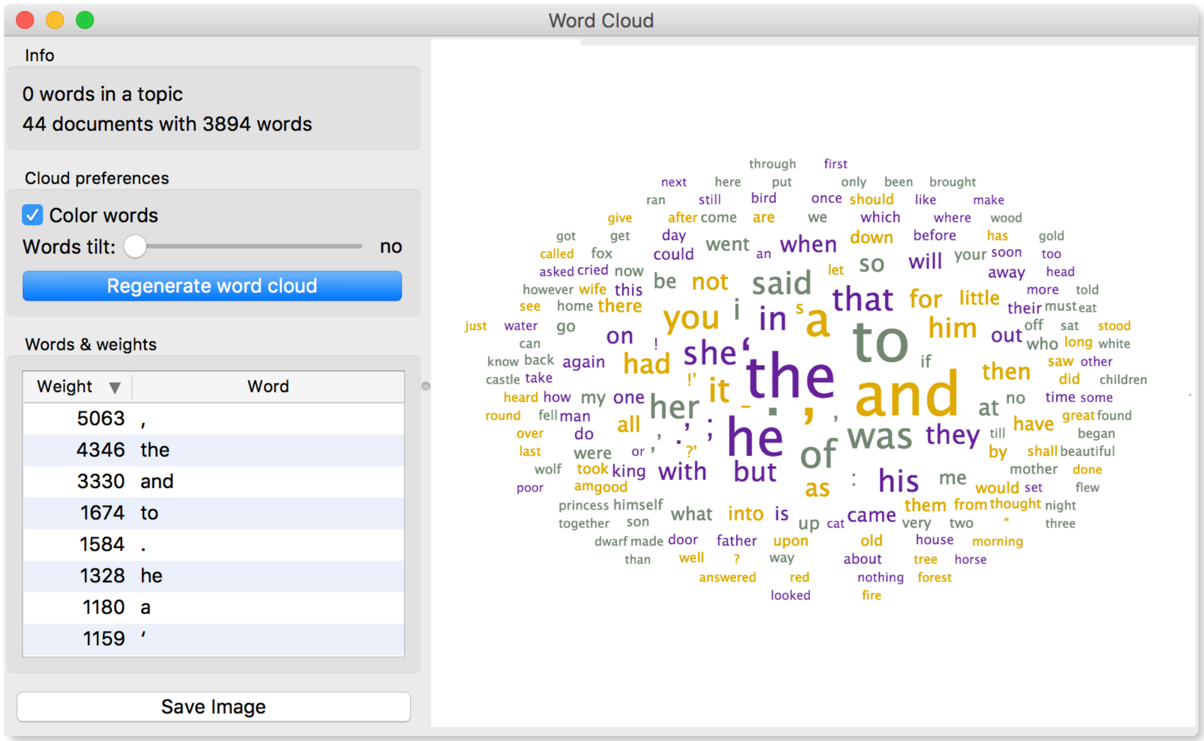
Corpus info

Corpus of 44 documents.

Used text features	Ignored text features
Content	Title
	Abstract
	ATU Numerical

Now open Word Cloud. Word Cloud displays word frequencies, where the more frequent the word, the larger the font. Select a word in the visualization and pass it to Corpus Viewer (1). You can now observe only those documents that contain the selected word in the Corpus Viewer.

But wait a second! This word cloud is a mess! We got a bunch of semantic junk in our visualization. Is there a way to clean this up?



0 words in a topic
44 documents with 3894 words

Color words: ✓
Words tilt: no

Weight ▼	Word
5063	,
4346	the
3330	and
1674	to
1584	.
1328	he
1180	a
1159	‘

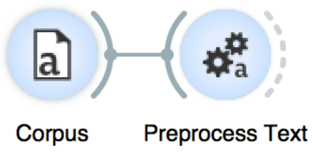
Of course! We need to remove all the bits that carry no information, namely punctuation and stopwords.

Lesson 2: Preprocessing Text

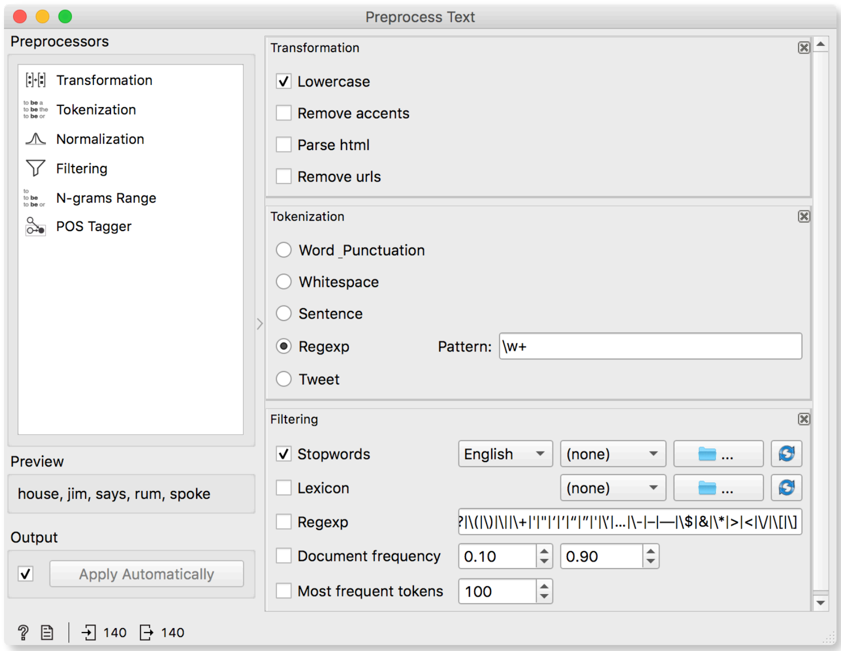
Word Cloud simply displayed all the words and symbols found in the text. But this is often not what we want. We want to extract only meaningful units, such as semantically rich words. This is why we need text preprocessing.

Preprocessing is key to defining what is important in our data. Is "Doctor" the same as "doctor"? Should we consider words such as "and", "the", "when" or omit them? Do we wish to treat "said" and "say" as the same word?

Preprocessing defines the core units of our analysis.



Token is a basic unit of our analysis. It can be a word, a bi-gram, a sentence… With preprocessing we define our tokens for the analysis.


Preprocess Text

Preprocessors
- Transformation
- Tokenization
- Normalization
- Filtering
- N-grams Range
- POS Tagger

Transformation
- [✓] Lowercase
- [ ] Remove accents
- [ ] Parse html
- [ ] Remove urls

Tokenization
- ○ Word_Punctuation
- ○ Whitespace
- ○ Sentence
- ● Regexp — Pattern: \w+
- ○ Tweet

Filtering
- [✓] Stopwords — English, (none)
- [ ] Lexicon — (none)
- [ ] Regexp — ?|\(|\)|\||\+|'|"|‘|’|“|”|'|\'|…|\-|–|—|\$|&|*|>|<|\/|\[|\]
- [ ] Document frequency — 0.10, 0.90
- [ ] Most frequent tokens — 100

Preview

house, jim, says, rum, spoke

Output
- [✓] Apply Automatically

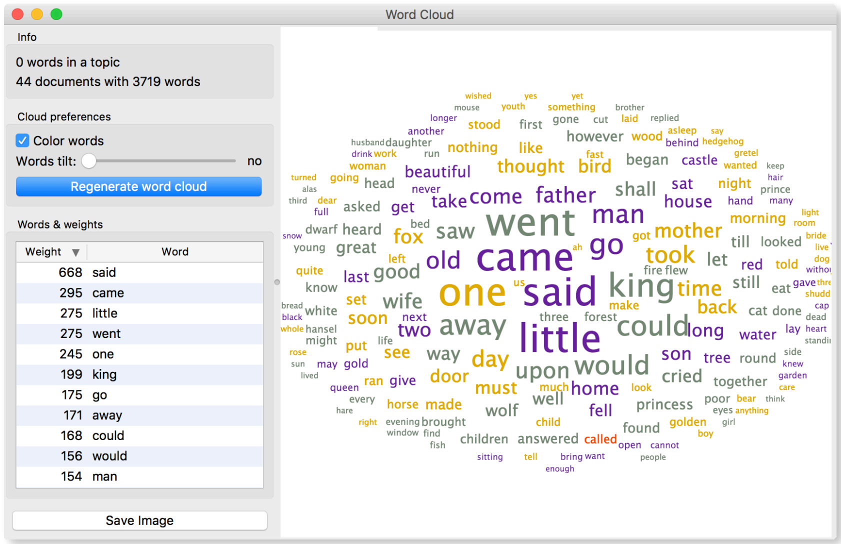
140 140

In the Preprocess Text widget, we decided to transform all words to lowercase, treat each word as a token (and omit punctuation), and to remove the stopwords (such as "in", "and", and "the"). This preprocessing outputs the following tokens:

"This is a sample sentence." → "sample", "sentence"

To see the results of preprocessing, we can display the most frequent tokens in Word Cloud. Word Cloud enables us to identify redundant words and irregularities.

We see the results of our preprocessing in the Word Cloud. Two of the most frequent words are “would” and “could”. If we decide these two words are not important for our analysis, it would be good to omit them. We can do this with custom filtering.



Word Cloud

Info

0 words in a topic
44 documents with 3719 words

Cloud preferences

✓ Color words
Words tilt: no

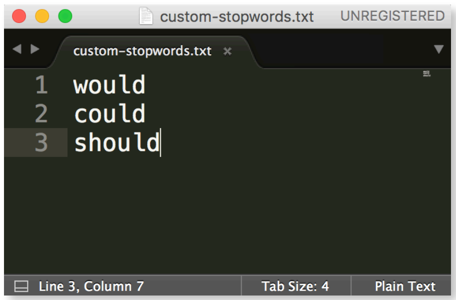
Words & weights

Weight	Word
668	said
295	came
275	little
275	went
245	one
199	king
175	go
171	away
168	could
156	would
154	man

To remove the words that carry no meaning, we have already filtered out some stopwords. But perhaps filtering out generic stopwords is not enough for our analysis.

We can always load our own custom stopword list. Open a plain text editor and create a custom list of stopwords. Write each new word on its own line and save the file.

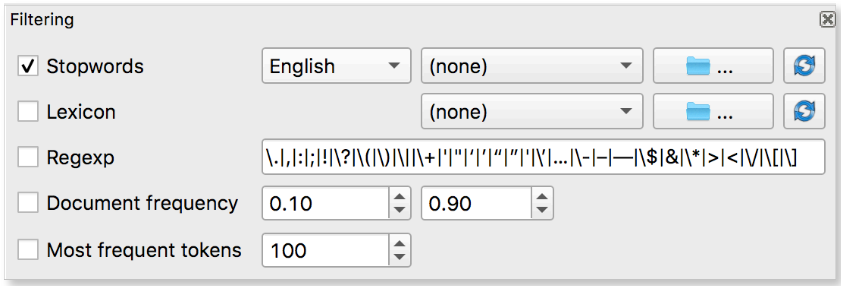
A good plain text editor is Sublime, but you can easily work with Notepad++.



custom-stopwords.txt

```
would
could
should
```

Load the list of custom stopwords in the right-hand dropdown of the Filtering section.



Filtering

✓ Stopwords	English	(none)	
☐ Lexicon		(none)	
☐ Regexp	\.|,|:|;|!|\?|\(|\)|\\||\+|'|"|‘|’|“|”|'|\'|…|\-|–|—|\$|&|*|>|<|\/|\[|\]		
☐ Document frequency	0.10	0.90	
☐ Most frequent tokens	100		

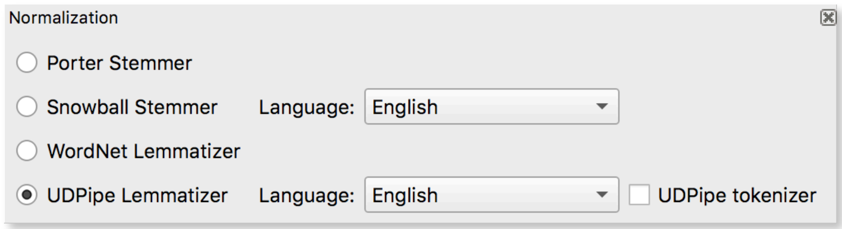
Another preprocessing technique is to filter out words that are too rare and too frequent. Rare words are normally found in only a few documents and frequent words are likely stopwords or very general words. To retain only those words that truly represent the corpus and may distinguish between corpus documents, we use *Document frequency* filter. If we set the values to 0.1 and 0.9; we will retain only those words that appear in more than 10% of the documents and in fewer than 90%.

Preprocessing is really the key to a successful text analysis. We have only mentioned a few techniques, but you can experiment on your own with the following ones:

For POS tag symbols see:

https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

- normalization transforms all words into lemmas or stems (for example *sons* to *son*)



Normalization

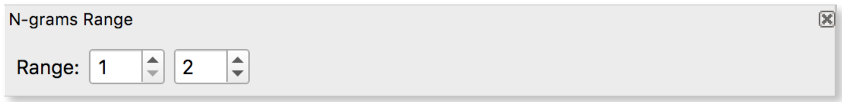
○ Porter Stemmer

○ Snowball Stemmer   Language: English

○ WordNet Lemmatizer

● UDPipe Lemmatizer   Language: English   ☐ UDPipe tokenizer

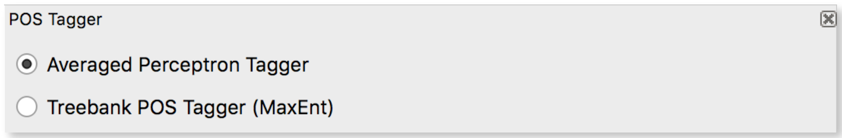
- n-grams are tokens of larger size, bigrams (a pair of consecutive words) and trigrams (word triplets), e.g. "office hours" or "Department of Justice".



N-grams Range

Range: 1 2

- POS tagging tags each token with a corresponding part-of-speech tag (sons → noun, plural, tag = NNS)



POS Tagger

● Averaged Perceptron Tagger

○ Treebank POS Tagger (MaxEnt)

Lesson 3: Context

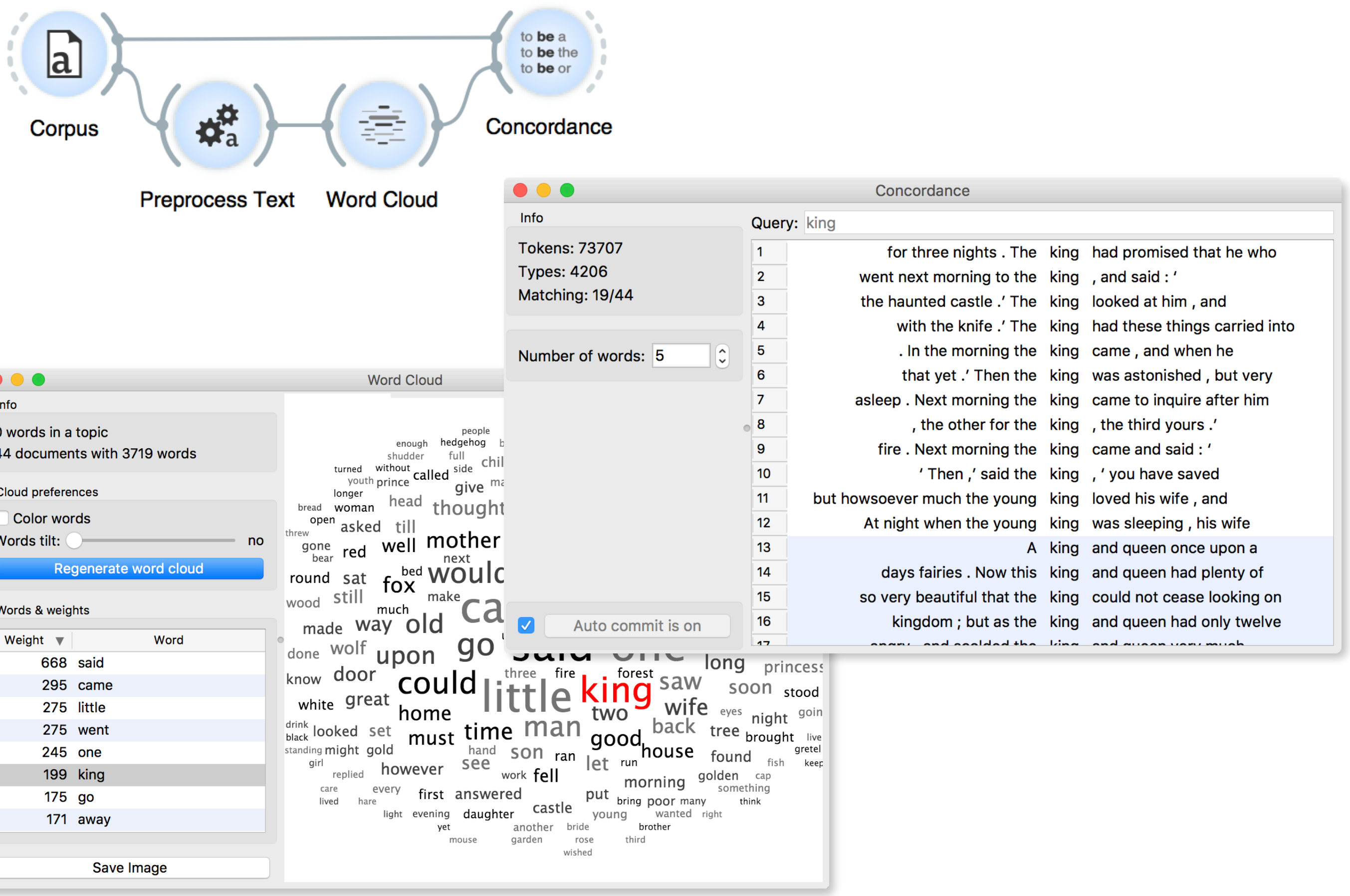
We have prepared our corpus and now it is time to visualize it. We have already seen some of the preprocessing results in a word cloud. Word Cloud shows us word frequencies. The more frequently the word appears in the corpus, the larger it will be in the word cloud.

But we still don't know much about the use of a specific word in a text. For example 'oh' could be a lowercase version of OH (the chemical compound of hydroxide), a simple exclamation 'Oh!' or an abbreviation for the state of Ohio.

To inspect the documents containing a particular word, select the documents in Concordance and pass them to Corpus Viewer for a deeper analysis.

To check the context of a particular word we can use Concordance widget. Concordance shows us the text around our word.

Connect Concordance to Corpus to pass the text to the widget. To browse the word, type it in the query line at the top or provide it with the Word Cloud. Here we have selected the word 'king' in the Word Cloud and observed the context in Concordance.



0 words in a topic
44 documents with 3719 words

Weight	Word
668	said
295	came
275	little
275	went
245	one
199	king
175	go
171	away

Tokens: 73707
Types: 4206
Matching: 19/44

Number of words: 5

Query: king

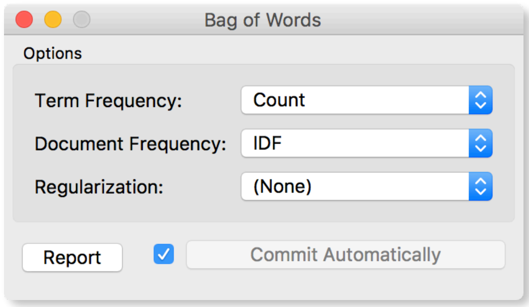
1	for three nights . The	king	had promised that he who
2	went next morning to the	king	, and said : '
3	the haunted castle .' The	king	looked at him , and
4	with the knife .' The	king	had these things carried into
5	. In the morning the	king	came , and when he
6	that yet .' Then the	king	was astonished , but very
7	asleep . Next morning the	king	came to inquire after him
8	, the other for the	king	, the third yours .'
9	fire . Next morning the	king	came and said : '
10	' Then ,' said the	king	, ' you have saved
11	but howsoever much the young	king	loved his wife , and
12	At night when the young	king	was sleeping , his wife
13	A	king	and queen once upon a
14	days fairies . Now this	king	and queen had plenty of
15	so very beautiful that the	king	could not cease looking on
16	kingdom ; but as the	king	and queen had only twelve

Lesson 4: Bag of Words

Now we have a preprocessed text, with proper tokens, but we still cannot really find any patterns in our text. For this, we need numbers and a simple way to convert documents into numeric vectors is to… well, count the words in each text.

Bag of Words creates a table with words in columns and documents in rows. Values are word occurrences in each document. They can be binary, but normally they are counts.

	this	is	an	example	another	apple
"This is an example"	1	1	1	1	0	0
"Another example"	0	0	0	1	1	0
"This is another apple.	1	1	0	0	1	1


Bag of Words

Options

Term Frequency: Count

Document Frequency: IDF

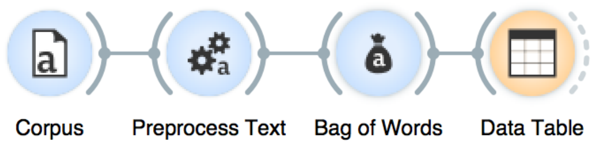
Regularization: (None)

However, text with many common words will have a greater importance than texts with many specific words. To balance the effect of stopwords, TF-IDF approach weights the matrix with total document frequency.

$$tf - idf = tf \times \log \frac{no\ of\ docs}{docs\ containing\ term}$$

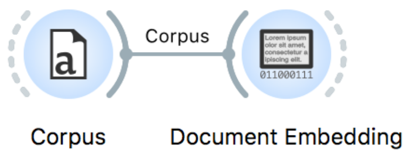
Using TF-IDF, common words will have a low value as they appear across most documents, while significant words will have a high value because they appear frequently in a small number of documents.

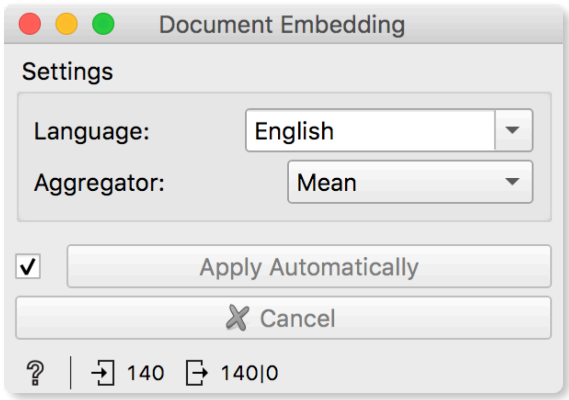
Pass the data through a Bag of Words widget and then again to a Data Table. We get a new column that contains word counts for each document. Now that we have numbers, we can finally perform some magic!



Lesson 5: Document Embedding

Bag of words, however, is not the only way to transform text into numbers. BoW is a great approach, because it is intuitive and interpretable (each feature is a word). But it requires a lot of careful preprocessing and with many words, the document-term matrix can get extremely large. Also, words like *mother* and *mom* would be unrelated in the BoW matrix, which we know is not the case. Wouldn't it be nice to have model that describes *mother* and *mom* with a similar number?





Settings

Language: English

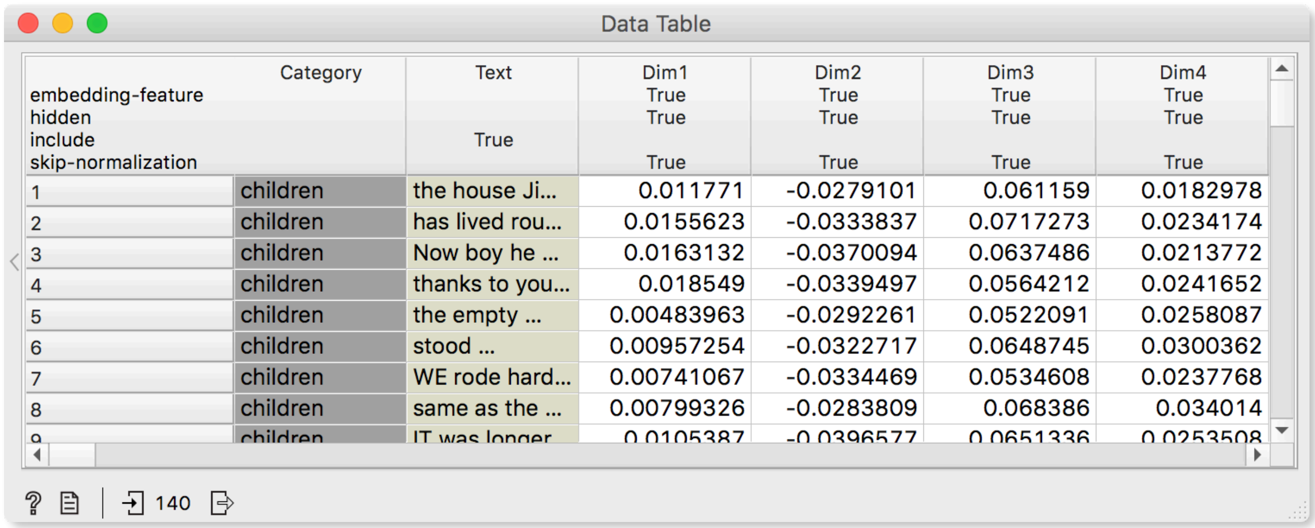
Aggregator: Mean

✓ Apply Automatically

140 | 140|0

Such models are called word embedders and a based on pre-trained deep models that map words in the language space. In such a model, words with similar meaning and words from the same family (car, Toyota, vehicle) would be placed close together. Computing a vector for an individual word based on the model is called embedding.

Orange uses fastText pre-trained models to embed words. Then is averages word vectors to produce a single document vector (one can also use sum, min or max aggregation). The document is now described with 300 features, regardless of the length of the document. However, features cannot be mapped to words — they are abstract representations in the language space.



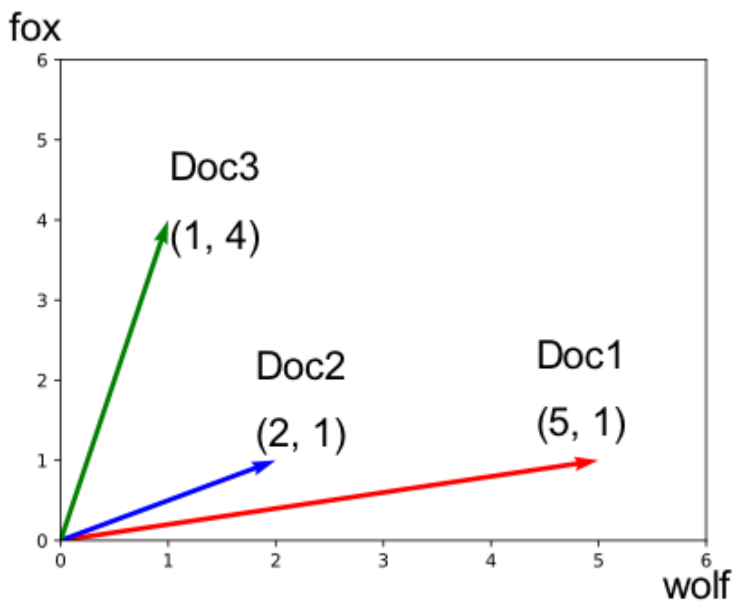
	Category	Text	Dim1	Dim2	Dim3	Dim4
embedding-feature			True	True	True	True
hidden			True	True	True	True
include		True				
skip-normalization			True	True	True	True
1	children	the house Ji...	0.011771	-0.0279101	0.061159	0.0182978
2	children	has lived rou...	0.0155623	-0.0333837	0.0717273	0.0234174
3	children	Now boy he ...	0.0163132	-0.0370094	0.0637486	0.0213772
4	children	thanks to you...	0.018549	-0.0339497	0.0564212	0.0241652
5	children	the empty ...	0.00483963	-0.0292261	0.0522091	0.0258087
6	children	stood ...	0.00957254	-0.0322717	0.0648745	0.0300362
7	children	WE rode hard...	0.00741067	-0.0334469	0.0534608	0.0237768
8	children	same as the ...	0.00799326	-0.0283809	0.068386	0.034014
9	children	IT was longer	0.0105387	-0.0396577	0.0651336	0.0253508

140

Lesson 6: Clustering & Distances

One common task in text mining is finding interesting groups of similar documents. That is, we would like to identify documents that are similar to each other.

We already know how to cluster data instances. We pass the data to Distances, use Euclidean distance, then to Hierarchical Clustering. But the Euclidean distance is not the only option. There are many distance measures and Euclidean doesn't work very well for text. Let us see why.



Using the Euclidean distance, document 2 would be closer to document 3 than to document 1. But documents 1 and 2 talk predominantly about wolves, while document 3 talks about foxes. The measure that captures the similarity of concepts without considering how many words there are in the text is called cosine distance.

Word counts from BoW are vectors, each pointing in a direction defined by text content as seen from the figure. Cosine distance is the angle between these vectors. Once the angle is considered, document 2 would be closer to document 1 than to document 3 (angle between them is smaller). For complex objects, such as texts, cosine distance is a frequent choice.

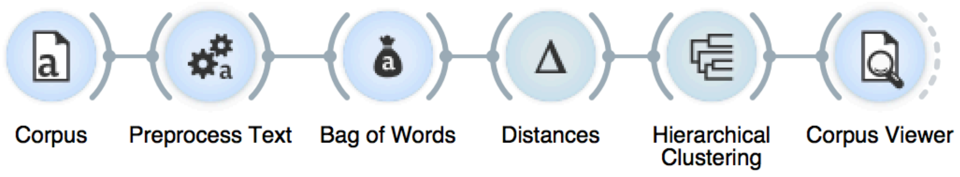
For text, an intuitive approach for measuring similarity would also be the number of words that two documents share. This is simply:

$$J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|}$$

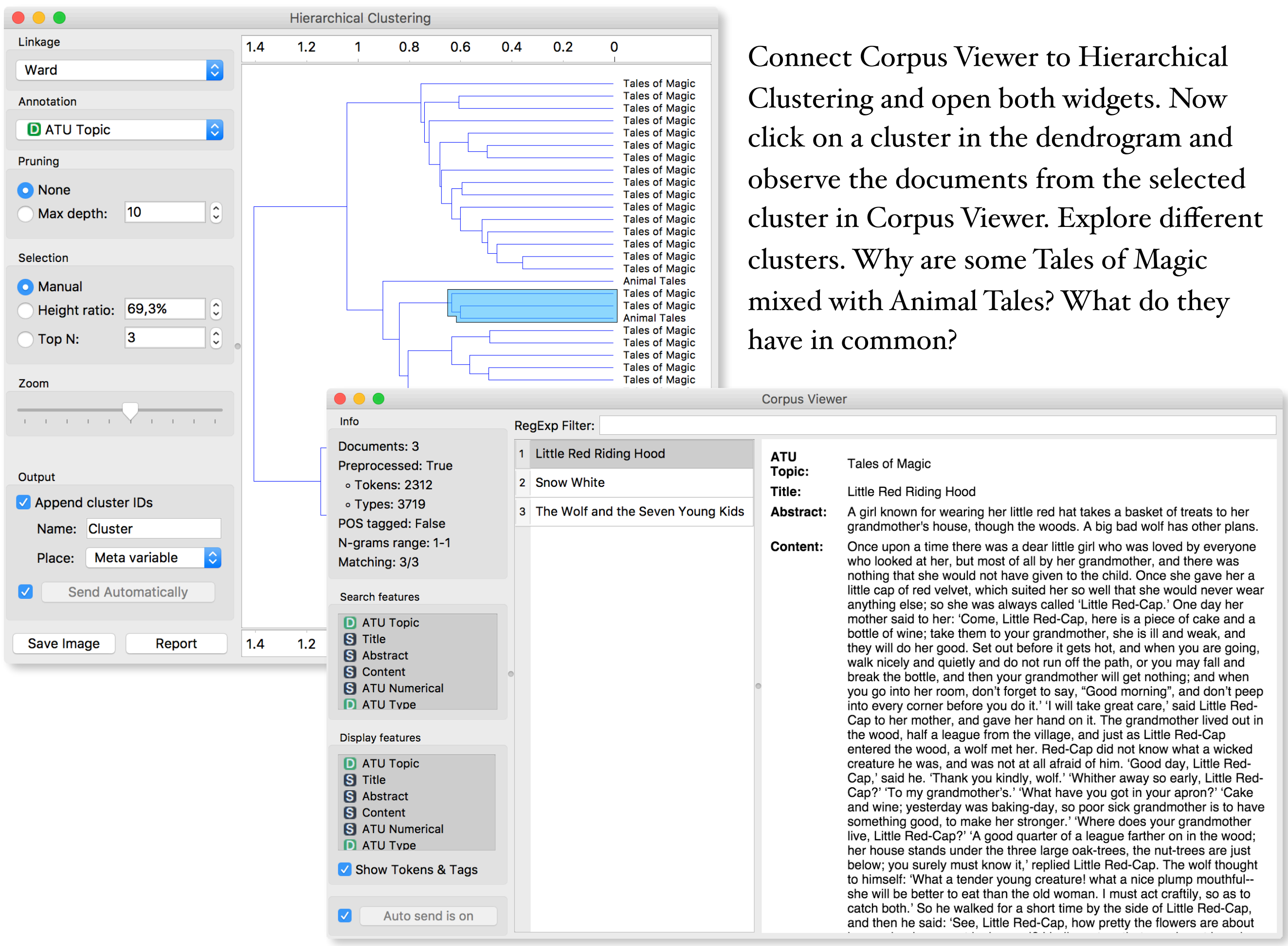
The measure is called Jaccard similarity coefficient or Jaccard index. Note that in this case, we are measuring similarity, not distance. Similarity is the opposite of distance, so to convert Jaccard index to distance, we would subtract it from 1.

You can try the same workflow on a different corpus, say *bookexcerpt.tab*, which contains excerpts from adult and children's books.

Now, let us go back to our Grimm's Tales and construct the following workflow:



Connect Corpus Viewer to Hierarchical Clustering and open both widgets. Now click on a cluster in the dendrogram and observe the documents from the selected cluster in Corpus Viewer. Explore different clusters. Why are some Tales of Magic mixed with Animal Tales? What do they have in common?



Linkage: Ward
Annotation: ATU Topic
Pruning: [✓ None] Max depth: 10
Selection: [✓ Manual] Height ratio: 69,3% Top N: 3
Zoom
Output: ✓ Append cluster IDs
Name: Cluster
Place: Meta variable
✓ Send Automatically

Documents: 3
Preprocessed: True
∘ Tokens: 2312
∘ Types: 3719
POS tagged: False
N-grams range: 1-1
Matching: 3/3

Search features: ATU Topic, Title, Abstract, Content, ATU Numerical, ATU Type

Display features: ATU Topic, Title, Abstract, Content, ATU Numerical, ATU Type

✓ Show Tokens & Tags

✓ Auto send is on

1 Little Red Riding Hood
2 Snow White
3 The Wolf and the Seven Young Kids

ATU Topic: Tales of Magic

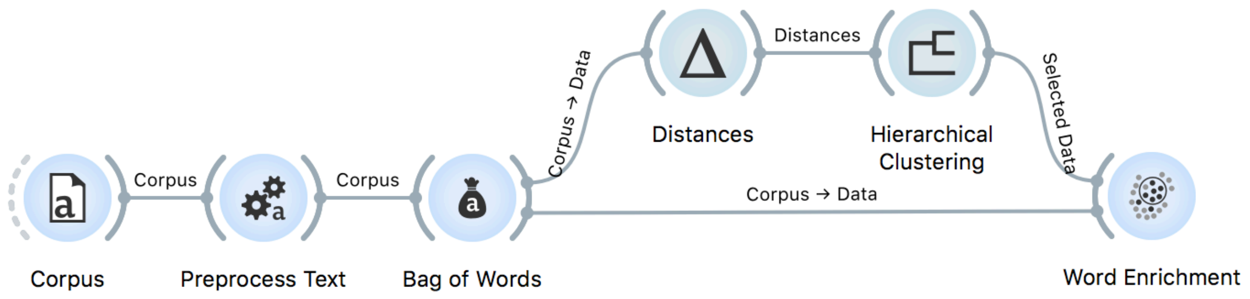
Title: Little Red Riding Hood

Abstract: A girl known for wearing her little red hat takes a basket of treats to her grandmother's house, though the woods. A big bad wolf has other plans.

Content: Once upon a time there was a dear little girl who was loved by everyone who looked at her, but most of all by her grandmother, and there was nothing that she would not have given to the child. Once she gave her a little cap of red velvet, which suited her so well that she would never wear anything else; so she was always called 'Little Red-Cap.' One day her mother said to her: 'Come, Little Red-Cap, here is a piece of cake and a bottle of wine; take them to your grandmother, she is ill and weak, and they will do her good. Set out before it gets hot, and when you are going, walk nicely and quietly and do not run off the path, or you may fall and break the bottle, and then your grandmother will get nothing; and when you go into her room, don't forget to say, "Good morning", and don't peep into every corner before you do it.' 'I will take great care,' said Little Red-Cap to her mother, and gave her hand on it. The grandmother lived out in the wood, half a league from the village, and just as Little Red-Cap entered the wood, a wolf met her. Red-Cap did not know what a wicked creature he was, and was not at all afraid of him. 'Good day, Little Red-Cap,' said he. 'Thank you kindly, wolf.' 'Whither away so early, Little Red-Cap?' 'To my grandmother's.' 'What have you got in your apron?' 'Cake and wine; yesterday was baking-day, so poor sick grandmother is to have something good, to make her stronger.' 'Where does your grandmother live, Little Red-Cap?' 'A good quarter of a league farther on in the wood; her house stands under the three large oak-trees, the nut-trees are just below; you surely must know it,' replied Little Red-Cap. The wolf thought to himself: 'What a tender young creature! what a nice plump mouthful--she will be better to eat than the old woman. I must act craftily, so as to catch both.' So he walked for a short time by the side of Little Red-Cap, and then he said: 'See, Little Red-Cap, how pretty the flowers are about

Lesson 7: Word Enrichment

We have previously explored the clusters with box plot. But for text mining, there is another option to find what is significant in the cluster. The approach is called word enrichment.

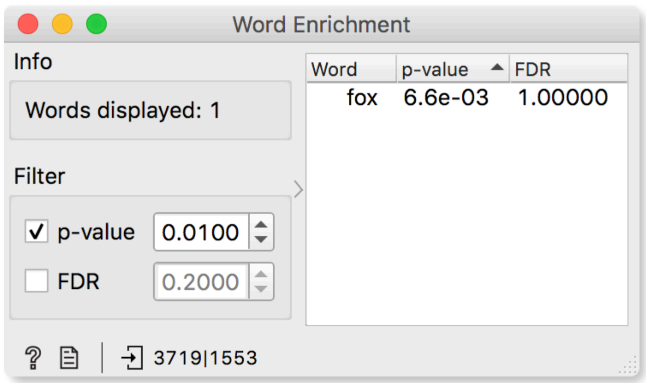


Word Enrichment works on any kind of subset.

Word Enrichment compares a subset of documents against the entire corpus and finds statistically significant words for the selected subset. It uses hypergeometric p-value to find words, that are overrepresented in the subset.

$$p = \frac{\binom{\textit{term in corpus}}{\textit{term in subset}} \times \binom{\textit{other terms}}{\textit{other terms in subset}}}{\binom{\textit{all terms}}{\textit{terms in subset}}}$$

FDR stands for false discovery rate. It is a measure to account for falsly significant words. In a large data matrix (which BoW usually is), some terms will be significant by chance. FDR tried to account for it.



Words displayed: 1

Word	p-value	FDR
fox	6.6e-03	1.00000

☑ p-value 0.0100
☐ FDR 0.2000

3719|1553

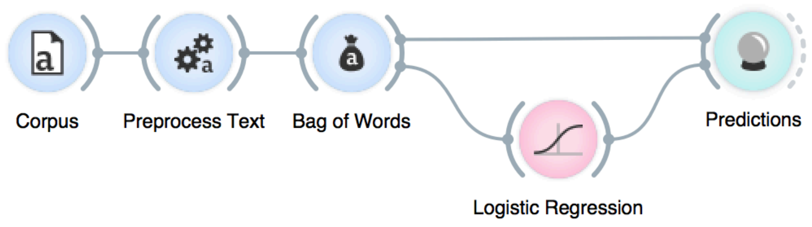
In the above clustering, we selected a cluster with mostly Animal Tales. Unsurprisingly, *fox* the most significant word in the subset. So the next time you see the word *fox* in a tale, you can bet the text is an animal one! :)

Lesson 8: Classification

Earlier we mentioned the Aarne-Thompson type (ATU). This is the index of folk-tale motifs and we have already marked every tale with a high-level (genre) and a mid-level ATU type (subgenre).

Could we perhaps predict the ATU type based on the content of the tale? Let us see.

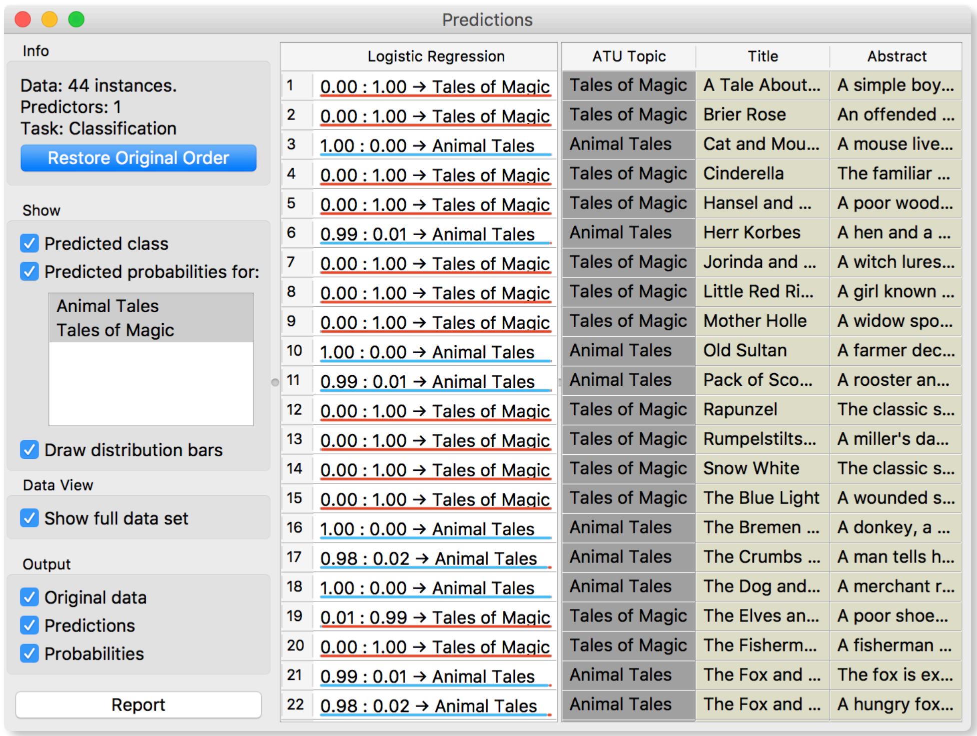
Aarne and Thompson were two folklorists, who invented and perfected the motif-based classification system of folk tales. This system has been in place since 1910 and is commonly used in comparative folkloristics. The final U in ATU stands for Uther, who was the last to update the index in 2004.



First, we need a target variable. This is the feature we are trying to predict, in our case an ATU type. We also need a numerical representation of each document - something we already have from the Bag of Words.

Now we will build a predictive model. A predictive model considers tokens (words) and predicts the target variable (ATU Topic). Every model also needs a learner, which is a method on how to consider the tokens. In our case, this is Logistic Regression.

In Predictions, we can see a column with predicted values from Logistic Regression. Seems like our model got most of the tale types right.



Info

Data: 44 instances.
Predictors: 1
Task: Classification

Show

- ✓ Predicted class
- ✓ Predicted probabilities for:
  - Animal Tales
  - Tales of Magic
- ✓ Draw distribution bars

Data View

- ✓ Show full data set

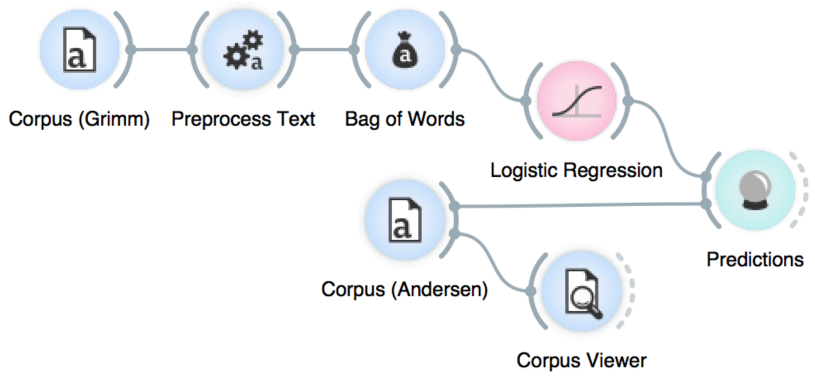
Output

- ✓ Original data
- ✓ Predictions
- ✓ Probabilities

	Logistic Regression	ATU Topic	Title	Abstract
1	0.00 : 1.00 → Tales of Magic	Tales of Magic	A Tale About...	A simple boy...
2	0.00 : 1.00 → Tales of Magic	Tales of Magic	Brier Rose	An offended ...
3	1.00 : 0.00 → Animal Tales	Animal Tales	Cat and Mou...	A mouse live...
4	0.00 : 1.00 → Tales of Magic	Tales of Magic	Cinderella	The familiar ...
5	0.00 : 1.00 → Tales of Magic	Tales of Magic	Hansel and ...	A poor wood...
6	0.99 : 0.01 → Animal Tales	Animal Tales	Herr Korbes	A hen and a ...
7	0.00 : 1.00 → Tales of Magic	Tales of Magic	Jorinda and ...	A witch lures...
8	0.00 : 1.00 → Tales of Magic	Tales of Magic	Little Red Ri...	A girl known ...
9	0.00 : 1.00 → Tales of Magic	Tales of Magic	Mother Holle	A widow spo...
10	1.00 : 0.00 → Animal Tales	Animal Tales	Old Sultan	A farmer dec...
11	0.99 : 0.01 → Animal Tales	Animal Tales	Pack of Sco...	A rooster an...
12	0.00 : 1.00 → Tales of Magic	Tales of Magic	Rapunzel	The classic s...
13	0.00 : 1.00 → Tales of Magic	Tales of Magic	Rumpelstilts...	A miller's da...
14	0.00 : 1.00 → Tales of Magic	Tales of Magic	Snow White	The classic s...
15	0.00 : 1.00 → Tales of Magic	Tales of Magic	The Blue Light	A wounded s...
16	1.00 : 0.00 → Animal Tales	Animal Tales	The Bremen ...	A donkey, a ...
17	0.98 : 0.02 → Animal Tales	Animal Tales	The Crumbs ...	A man tells h...
18	1.00 : 0.00 → Animal Tales	Animal Tales	The Dog and...	A merchant r...
19	0.01 : 0.99 → Tales of Magic	Tales of Magic	The Elves an...	A poor shoe...
20	0.00 : 1.00 → Tales of Magic	Tales of Magic	The Fisherm...	A fisherman ...
21	0.99 : 0.01 → Animal Tales	Animal Tales	The Fox and ...	The fox is ex...
22	0.98 : 0.02 → Animal Tales	Animal Tales	The Fox and ...	A hungry fox...

Lesson 9: Predictions

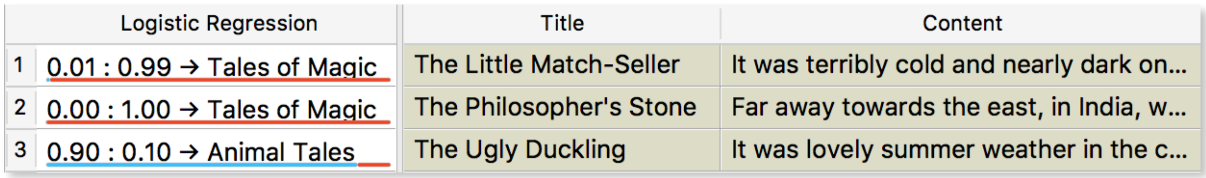
Predicting on new data works just like for regular data.



Open a new Corpus widget and load the *andersen.tab* corpus. Here we have three tales from H. C. Andersen. Inspect them in Corpus Viewer and try to guess the tale type yourself.

Now connect them to Predictions the same way as before - with Logistic Regression passing the constructed model and the new Corpus widget passing the data for prediction. Logistic Regression predicted two tales to be Tales of Magic and one the Animal Tale.

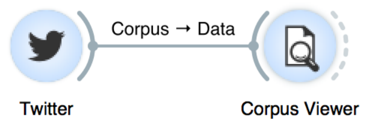
The Ugly Duckling as an animal tale? Sounds quite right!



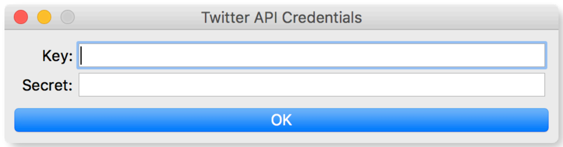
	Logistic Regression	Title	Content
1	0.01 : 0.99 → Tales of Magic	The Little Match-Seller	It was terribly cold and nearly dark on...
2	0.00 : 1.00 → Tales of Magic	The Philosopher's Stone	Far away towards the east, in India, w...
3	0.90 : 0.10 → Animal Tales	The Ugly Duckling	It was lovely summer weather in the c...

Lesson 10: Twitter Data

The Grimm's Tales already come with the program. Text add-on, however, can retrieve data from many other sources: Twitter, Guardian, New York Times, and Wikipedia!



Here's an example on how to use the Twitter widget.

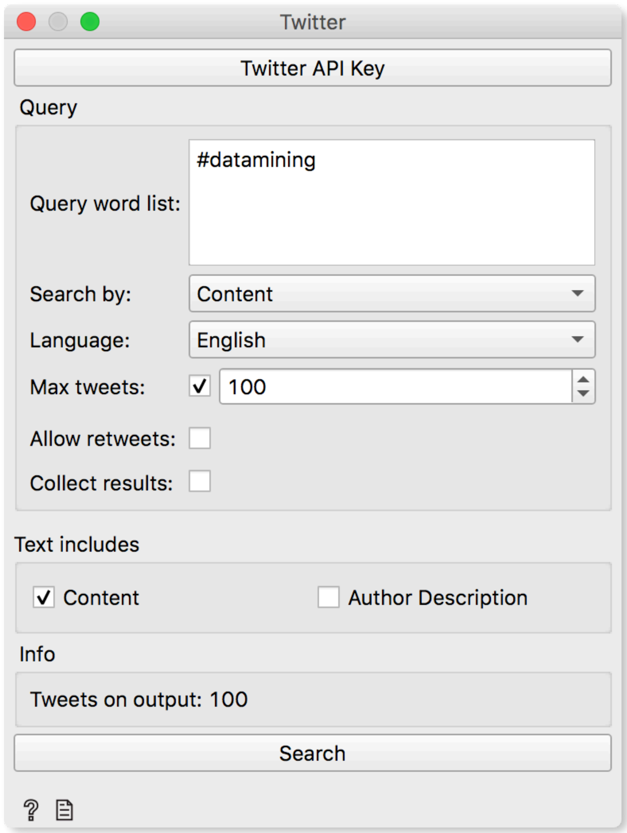


Entry the query, which can be a word, a hashtag, or a mention. You can provide multiple queries, one per line, which will be considered with OR (at least one query appears in the tweet). Set the language to English (or any other language) to limit tweet language.

To use Twitter widget you will need to get a Twitter API key. Go to https://apps.twitter.com/ and create a new app. Once you've created the app, you will get your own API key.

Enter it into the Twitter API Key section and begin using the Twitter widget.

For this example, we will retrieve a hundred English tweets with a hashtag #datamining. We have entered the query in the "Query word list" and set the language to English.



Query word list: #datamining

Search by: Content

Language: English

Max tweets: ✓ 100

Allow retweets: ☐

Collect results: ☐

✓ Content ☐ Author Description

Tweets on output: 100

Now we run "Search" and Twitter widget will send the retrieved tweets immediately to the output. Connect Corpus Viewer to Twitter to observe the retrieved data.

Lesson 11: Twitter Preprocessing

Twitter requires specific preprocessing. Why? Think about this tweet:

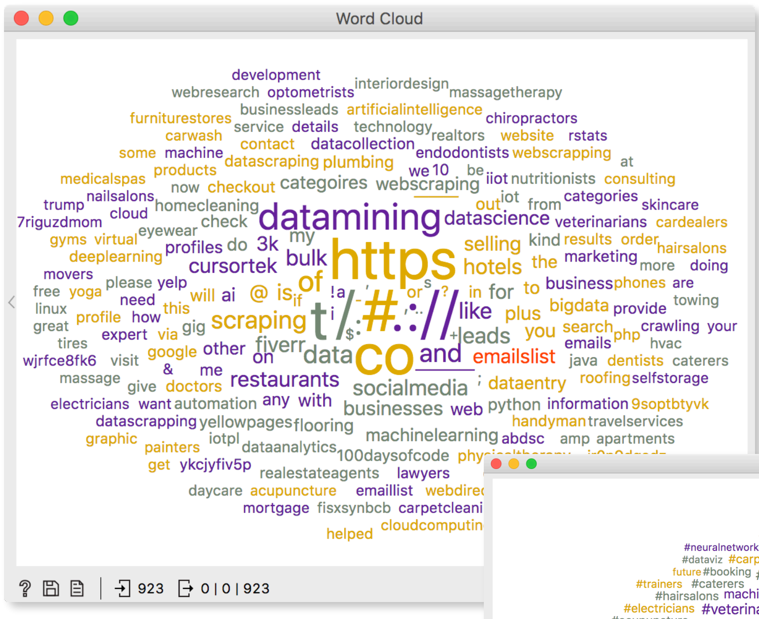
"I am looking forward to today's lesson, @drprofessor. #course 😊*"*

With the standard preprocessing, the tokens would be the following:

i, am, looking, forward, to, today's, lesson, @, drprofessor, #, course, :, -,)

Not exactly what we want. Preprocessing should keep mentions, hashtags and emojis together as one token, not separate it. For this, we use the pre-trained Tweet tokenizer.

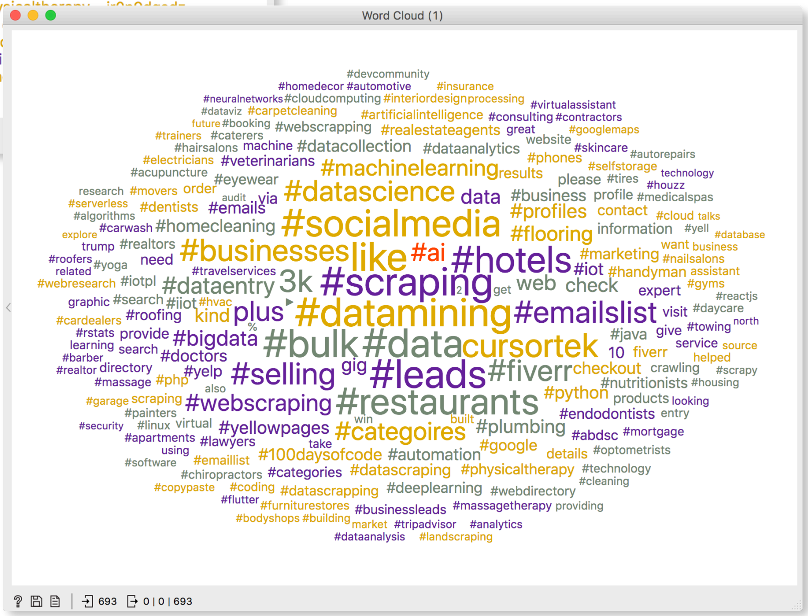
Text preprocessing steps are the following:



1. Remove urls from the text. Twitter's URLs are not informative.

2. Use a special Tweet tokenizer, that was pre-trained on millions of tweets. It keeps hashtags, emojis, and mentions.

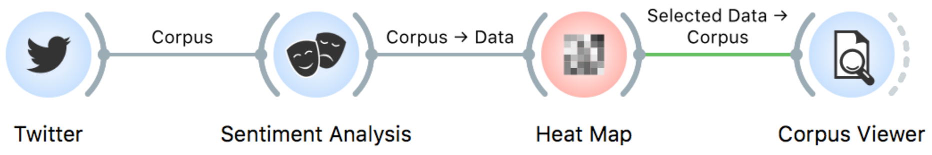
3. Remove punctuation with regex (Tweet tokenizer doesn't do it by default).



Un-preprocessed Word Cloud above and a preprocessed Word Cloud with Tweet tokenizer on the right.

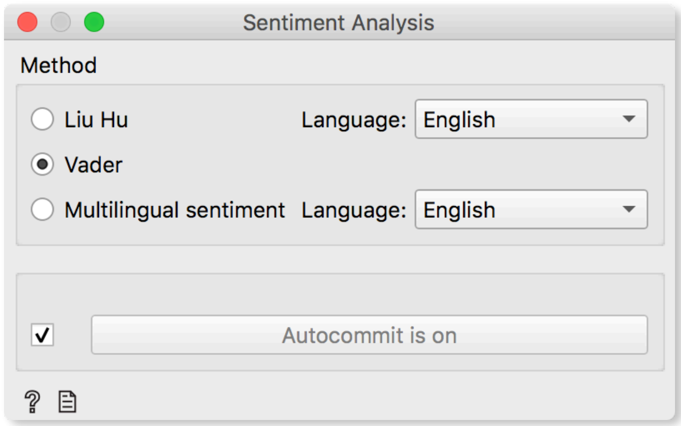
Lesson 11: Sentiment Analysis

Can we discover how people *feel* about data mining? Sure, with sentiment analysis. We will pass the tweets to Sentiment Analysis widget and compute a sentiment score.



Connect Sentiment Analysis to Twitter. Sentiment Analysis uses dictionary-based approaches to discover positive or negative words and provides a total sentiment score. We will use Vader, which is a smarter approach that recognizes phrases ("This is sick, dude."), punctuation (Great!!!!!), and emojis (😲).

More advanced techniques for sentiment analysis are based on models, usually with deep neural networks that learn from a large amount of labelled texts.



Method

- ○ Liu Hu — Language: English
- ● Vader
- ○ Multilingual sentiment — Language: English

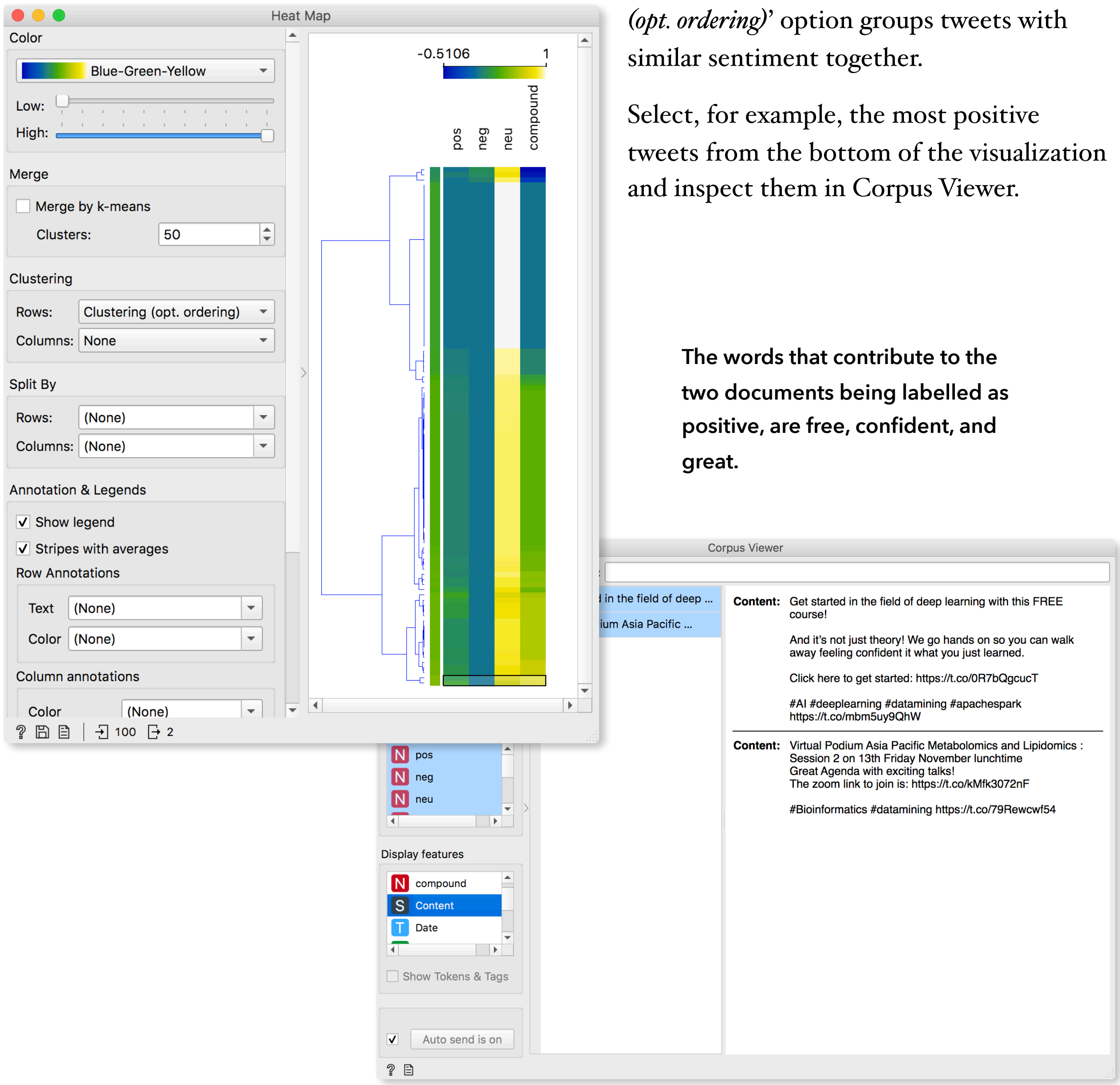
✓ Autocommit is on

Vader outputs 4 sentiment features, namely positive (pos), neutral (neu), negative (neg) and compound scores. Liu & Hu and Multilingual sentiment are both simpler methods based solely on dictionary words. Positive dictionary is used to count positive words and negative dictionar for negative ones. The sum of negative words is subtracted from the sum of positive words to get the final score.

We will observe sentiment strength and polarity in a Heat Map. Heat map shows numeric attributes, where each value is colored according to scale. In our example, data with higher values are yellow and white, while data with lower values are blue. '*Clustering* *(opt. ordering)*' option groups tweets with similar sentiment together.

Select, for example, the most positive tweets from the bottom of the visualization and inspect them in Corpus Viewer.

The words that contribute to the two documents being labelled as positive, are free, confident, and great.



Content: Get started in the field of deep learning with this FREE course!

And it's not just theory! We go hands on so you can walk away feeling confident it what you just learned.

Click here to get started: https://t.co/0R7bQgcucT

#AI #deeplearning #datamining #apachespark https://t.co/mbm5uy9QhW

Content: Virtual Podium Asia Pacific Metabolomics and Lipidomics : Session 2 on 13th Friday November lunchtime
Great Agenda with exciting talks!
The zoom link to join is: https://t.co/kMfk3072nF

#Bioinformatics #datamining https://t.co/79Rewcwf54