Orange3-Argument Documentation

Release 0.1.2

Biolab

Nov 21, 2023

CONTENTS

This work is an open-source Python package that implements a pipeline of processing, analyzing, and visualizing an argument corpus and the attacking relationship inside the corpus.

It also implements the corresponding GUIs on a scientific workflow platform named Orange3, so that users with little knowledge of Python programming can also benefit from it.

CHAPTER

ONE

CONTENTS

1.1 Introduction

This work is designed with a clear mission: to empower researchers in building their own argument mining workflows effortlessly. Leveraging the capabilities of state-of-the-art, pre-trained language models for natural language processing, this tool facilitates the process of processing, analyzing, and understanding arguments from text data.

At its core, this work is committed to transparency and interpretability throughout the analysis process. We believe that clarity and comprehensibility are paramount when working with complex language data. As such, the tool not only automates the task but also ensures that the results are easily interpretable, allowing researchers to gain valuable insights from their data.

Moreover, we have implemented an intuitive, visual programming module that brings the power of argument mining to researchers with limited programming expertise. This feature enables individuals from diverse backgrounds to harness the potential of argument analysis, making it accessible to a broader range of researchers and practitioners.

The package contains three components that can be used to build the workflow:

- Chunker: Split arguments into smaller chunks, learn topics of chunks through topic modeling, measure sentiment and important of chunks within arguments.
- Processor: Merge chunks and meta back to arguments, compute coherence and other potential measurements of arguments.
- Miner: Build attack network of arguments, label supportive and defeated arguments based on the network structure.

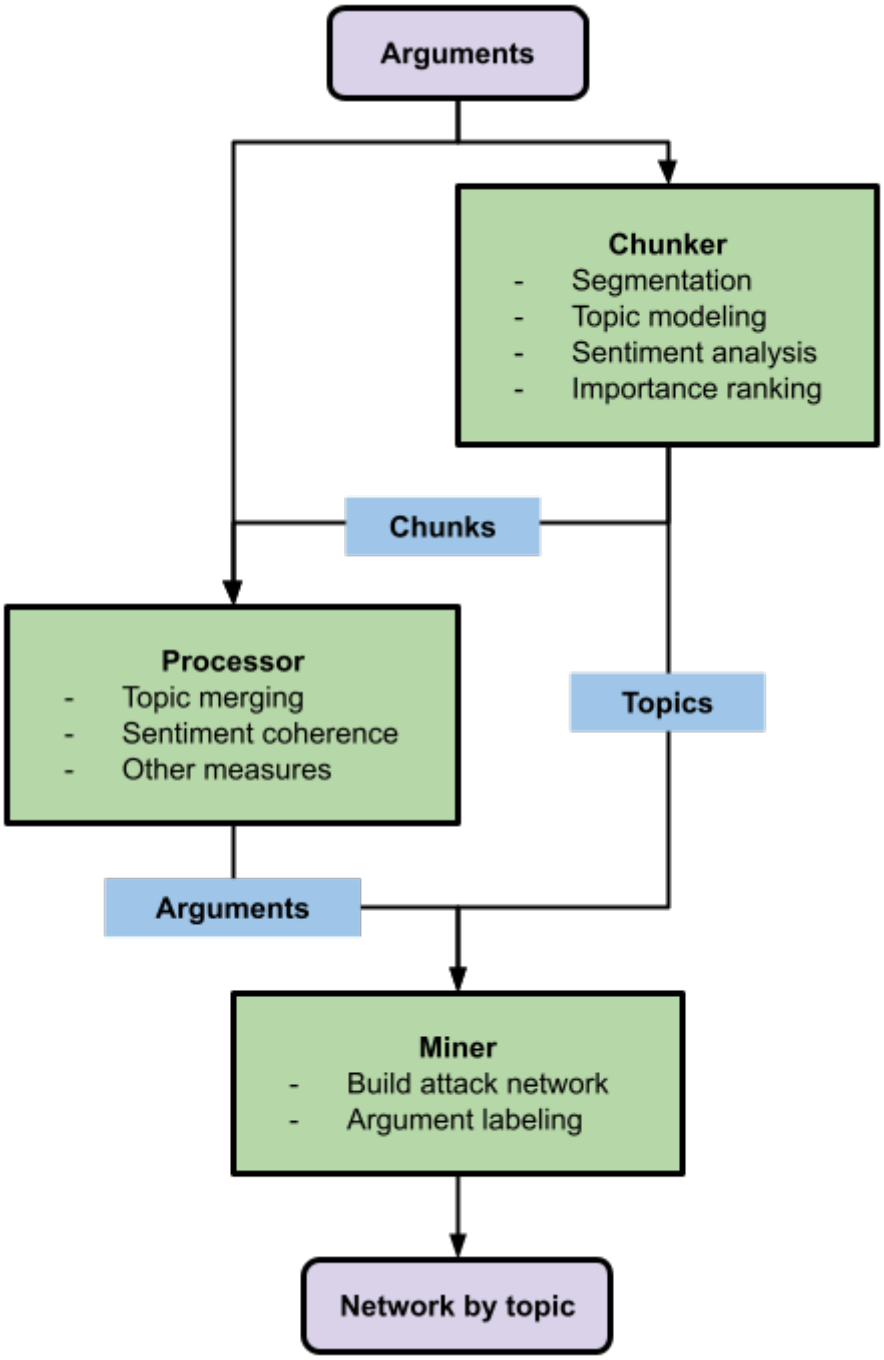
1.2 Installation

1.2.1 Preparation

To install this package, we assume that you have Python installed on your computer. However, if that is not the case, we highly recommend that you first consult the installation guides of Python. You should install Python 3.8 or higher versions to use this package. Additionally, while it's not necessary to be familiar with shell commands, if you're interested, you can explore this helpful list of commonly used shell commands.

Once you have Python installed, open the terminal on your computer:

- **Windows**: If you runs Windows 11 on your computer, press the `Win` key, search for "PowerShell" and then open it. In case of Windows 10, you need to first download it from the Microsoft Store.
- **Linux**: You can press the `Ctrl + Alt + T` key to fire up the terminal.


Arguments
Chunker
- Segmentation
- Topic modeling
- Sentiment analysis
- Importance ranking
Chunks
Processor
- Topic merging
- Sentiment coherence
- Other measures
Topics
Arguments
Miner
- Build attack network
- Argument labeling
Network by topic

- **MacOS**: Click the Launchpad icon in the Doc, or press the `Cmd + Space` type "Terminal" in the search field, then click Terminal.

1.2.2 Installation

To install, we recommend to first navigate to your working directory by running this command:

```
cd /path/to/your/working/directory
```

We recommend to install our package in a new virtual environment to avoid dependency conflicts, and we recommand to use *venv* to do this:

```
python -m venv venv
```

To activate the virtual environment just created, on Windows, run:

```
venv\Scripts\activate
```

And on Linux and MacOS, run:

```
source venv/bin/activate
```

Then, to install this package, run:

```
pip install orangearg
```

1.3 Example: Review Labeling by Topic

In this notebook, we will use a subset of the Amazon Product Review data to demonstrate the usage of this work for labeling arguments. The problem to be addressed here is determining the credibility (reliable/unreliable) of reviews that evaluate a specific aspect of the product (i.e. size of shoes) and being able to provide reasoning for the results. The dataset can be found here.

[21]:
```
from orangearg.argument.miner import reader, chunker, processor, miner

fpath = "./example_dataset.json"
```

1.3.1 Read the input file

[2]:
```
df_arguments = reader.read_json_file(fpath=fpath)
df_arguments = df_arguments.dropna().reset_index(drop=True)  # remove rows with na
```

The results of reading the data file are as follows. It can be seen that this dataset contains two aspects of information, namely the text of the reviews (`reviewText`) and the rating evaluations provided by the purchasers (`overall`, ranging from 1 to 5 stars).

[3]:
```
df_arguments
```

[3]:
```
                                            reviewText  overall
0    I always get a half size up in my tennis shoes...        3
1    Put them on and walked 3 hours with no problem...        5
2                                             excelente        5
3    The shoes fit well in the arch area. They are ...        4
4    Tried them on in a store before buying online ...        5
..                                                 ...      ...
365  Favorite Nike shoe ever! The flex sole is exce...        5
366       I wear these everyday to work, the gym, etc.        5
367    Love these shoes! Great fit, very light weight.        5
368  Super comfortable and fit my small feet perfec...        5
369                                 Love these shoes!        5

[370 rows x 2 columns]
```

1.3.2 Split arguments into chunks

By analyzing, reviews will first be segmented into smaller chunks, which are clauses that express complete meanings. The reason for doing this is to identify from which different perspectives reviews provide their evaluations, in preparation for the subsequent review labeling process.

[4]:
```
arguments = df_arguments["reviewText"]
arg_scores = df_arguments["overall"]

# Split reviews into chunks
chunk_arg_ids, chunks = chunker.get_chunk(docs=arguments)

# Compute polarity score of chunks
chunk_p_scores = chunker.get_chunk_polarity_score(chunks=chunks)

# Compute topics of chunks
chunk_topics, chunk_embeds, df_topics = chunker.get_chunk_topic(chunks=chunks)

# Comput importance of chunks inside the arguments
chunk_ranks = chunker.get_chunk_rank(arg_ids=chunk_arg_ids, embeds=chunk_embeds)

# Collect everything together as a dataframe
df_chunks = chunker.get_chunk_table(
    arg_ids=chunk_arg_ids,
    chunks=chunks,
    p_scores=chunk_p_scores,
    topics=chunk_topics,
    ranks=chunk_ranks
)
```

Some explanations of `df_chunks`:

- `argument_id`: the index of the argument the chunk coming from.
- `polarity_score`: the sentiment polarity score of a chunk, in range of [-1, 1], where 0 signifies neutrality, positive values indicate positivity, and negative values denote negativity.
- `topic`: the index of a topic in the `df_topics` table below.

- `rank`: importance of a chunk within the argument it comes from, in range of [0, 1]. This is computed as the pagerank of chunks based on the similarity network of chunks. Therefore, the sum of ranks from chunks belonging to the same argument is equal to 1.

[5]:
```
df_chunks
```

[5]:
```
      argument_id                                              chunk  \
0               0   I always get a half size up in my tennis shoes .
1               0  For some reason these feel to big in the heel ...
2               1                      walked 3 hours with no problem
3               1                                 Put them on and !
4               1                                      Love them !
...           ...                                               ...
1192          368               I can wear the shoe all day long and
1193          368  they are easy to clean compared to other shoes...
1194          368  They are light colored so any dirt will be see...
1195          368  Would definitely buy another pair in a differe...
1196          369                                 Love these shoes !

      polarity_score  topic      rank
0          -0.166667      4  0.500000
1          -0.050000     10  0.500000
2           0.000000      7  0.249374
3           0.000000      2  0.255228
4           0.625000      3  0.250344
...              ...    ...       ...
1192       -0.050000     15  0.125961
1193        0.225000      0  0.128238
1194        0.342857     23  0.128449
1195        0.000000     13  0.125681
1196        0.625000     22  1.000000

[1197 rows x 5 columns]
```

And explanations of `df_topics`:

- `topic`: the index of a topic
- `count`: the number of chunks in a topic
- `keywords`: the top keywords of a topic
- `name`: a short name of the topic

[6]:
```
df_topics.head()
```

[6]:
```
   topic  count                          name  \
0      0    147          0_shoes_the_these_for
1      1     99  1_fit_perfect_true_perfectly
2      2     87            2_for_them_work_use
3      3     79           3_love_them_they_are
4      4     74        4_size_ordered_half_big

                                        keywords
0  (shoes, the, these, for, shoe, comfortable, ar...
1  (fit, perfect, true, perfectly, fits, expected...
2  (for, them, work, use, wear, training, in, gym...
```

(continues on next page)

(continued from previous page)

```
3  (love, them, they, are, these, cute, really, p...
4  (size, ordered, half, big, large, order, an, a...
```

1.3.3 Merge chunks back to arguments

By merging the chunks back into reviews and performing the corresponding computations, we will obtain relevant information at the review level, including the topics covered in each review, the sentiment of the review, and its consistency with the overall score. This information will be further used for labeling reviews under different topics.

[7]:
```
# Compute topics of arguments
arg_topics = processor.get_argument_topics(arg_ids=chunk_arg_ids, topics=chunk_topics)

# Compute sentiment of arguments
arg_sentiments = processor.get_argument_sentiment(arg_ids=chunk_arg_ids, ranks=chunk_
↪ranks, p_scores=chunk_p_scores)

# Compute the coherence between the sentiments and the overall of arguments
arg_coherences = processor.get_argument_coherence(scores=arg_scores, sentiments=arg_
↪sentiments)

# Collect everything together as a datafrae
df_arguments_processed = processor.update_argument_table(
    df_arguments=df_arguments,
    topics=arg_topics,
    sentiments=arg_sentiments,
    coherences=arg_coherences
)
```

Some columns are added to the original `df_arguments` dataframe, which are:

- `topics`: the topics that an argument has mentioned.
- `sentiment`: the sentiment score of an argument, in range of [0, 1], the higher the more positive.
- `coherence`: the coherence between the `sentiment` and `overall`, in range of [0, 1], the higher the more coherent.

[8]:
```
df_arguments_processed.head()
```

[8]:
```
                                        reviewText  overall  \
0  I always get a half size up in my tennis shoes...        3
1  Put them on and walked 3 hours with no problem...        5
2                                        excelente        5
3  The shoes fit well in the arch area. They are ...        4
4  Tried them on in a store before buying online ...        5

            topics  sentiment  coherence
0          (4, 10)   0.445833   0.992692
1     (7, 2, 3, 9)   0.627243   0.706545
2             (6,)   0.500000   0.535261
3  (0, 10, 10, 21)   0.524397   0.880521
4  (1, 0, 5, 0, 6)   0.712758   0.813614
```

1.3.4 Review labeling

In this step, by looking at reviews under a specific topic, an attacking network of reviews are built, where nodes are reviews and edges are the attacks in between. Reviews are labeled based on that.

These are the rules of generating the network:

- Edges exist only between reviews with different `overall`.
- Edges start from a review with higher coherence to lower coherence.
- Weight of edges are computed as difference of coherence of the vertices.
- A node is labeled as `supportive` (meaning reliable in our case),
  - if no other nodes attack it, or
  - if all attackers of this node are attacked by some other nodes.
- A node is labeled as `defeated` (meaning unreliable in our case), if it is not `supportive`.

[9]:
```
from IPython.display import display, HTML

# Select reviews of the last topic
last_topic = df_topics.iloc[-1]["topic"]
print(f"The last topic is topic nr. {last_topic}:")
display(HTML(df_topics[df_topics["topic"] == last_topic].to_html()))
```

```
The last topic is topic nr. 24:
```

```
<IPython.core.display.HTML object>
```

Seems that the arguments under this topic are about judgements of the returning experience of this product.

[10]:
```
# select the arguments under the last topic
arg_selection = miner.select_by_topic(data=df_arguments_processed, topic=last_topic)
arg_selection = arg_selection.rename(columns={
    "reviewText": "argument",
    "overall": "score"
})  # rename columns for the following steps
arg_selection
```

[10]:

	argument	score	\
0	I wore these shoe one time, from the airport i...	1	
1	I usually wear a size 8 and they fit fine. The...	1	
2	Great shoe! Outside arch is kind of high, but ...	5	
3	I bought these for gym training - weight class...	2	
4	Oops! I returned these because I ordered wrong...	1	
5	I loved these shoes...that is until after abou...	1	
6	I returned them...found a Ryka pair I liked be...	3	
7	I got the impression it's cushiony and comfy b...	3	
8	Ordered 9(m) received 9 Wide for the second ti...	1	
9	Returning these. the pictures on here make the...	1	
10	Tried one in the store and bought it online bu...	2	
11	I returned these as they were not true to size...	2	
12	I bought a pair of these in my size, but they ...	3	
13	Unfortunately, this Flex Supreme does NOT have...	1	
14	After using this shoes seven times for regular...	1	

(continues on next page)

(continued from previous page)

```
                            topics  sentiment  coherence  argument_id
0              (16, 14, 23, 24, 24)   0.500000   0.535261           46
1              (1, 4, 10, 24, 4, 6)   0.496439   0.540030           77
2   (21, 2, 19, 14, 4, 4, 4, 1, 24)   0.659098   0.747863           78
3      (13, 2, 7, 7, 18, 7, 24, 10)   0.516497   0.837317           83
4                          (14, 24)   0.343750   0.744226          114
5                       (13, 7, 24)   0.599393   0.407310          118
6                             (24,)   0.775000   0.827735          121
7      (0, 2, 24, 11, 9, 18, 6, 24)   0.565749   0.989251          154
8                       (4, 4, 24)   0.491652   0.546454          205
9            (24, 23, 23, 23, 3, 15)   0.520525   0.507953          254
10                  (24, 24, 23, 1)   0.557920   0.788963          263
11                         (24, 4)   0.509821   0.844705          266
12                 (4, 4, 7, 0, 24)   0.440069   0.991061          288
13                  (9, 10, 24, 24)   0.494785   0.542248          304
14                  (7, 14, 20, 24)   0.346269   0.741000          305
```

[11]:
```
# Compute edges of the attacking network
edges = miner.get_edges(data=arg_selection)
weights = miner.get_edge_weights(data=arg_selection, edges=edges)
df_edges = miner.get_edge_table(edges=edges, weights=weights)
```

Edges are defined between reviews with different overall scores. Also, edges are directed and weighted, where `source` and `target` are indices of reviews in `arg_selection`.

[12]:
```
df_edges
```

[12]:
```
    source  target  weight
0        2       0    0.21
1        3       0    0.30
2        6       0    0.29
3        7       0    0.45
4       10       0    0.25
..     ...     ...     ...
66      12      11    0.15
67      11      13    0.30
68      11      14    0.10
69      12      13    0.45
70      12      14    0.25

[71 rows x 3 columns]
```

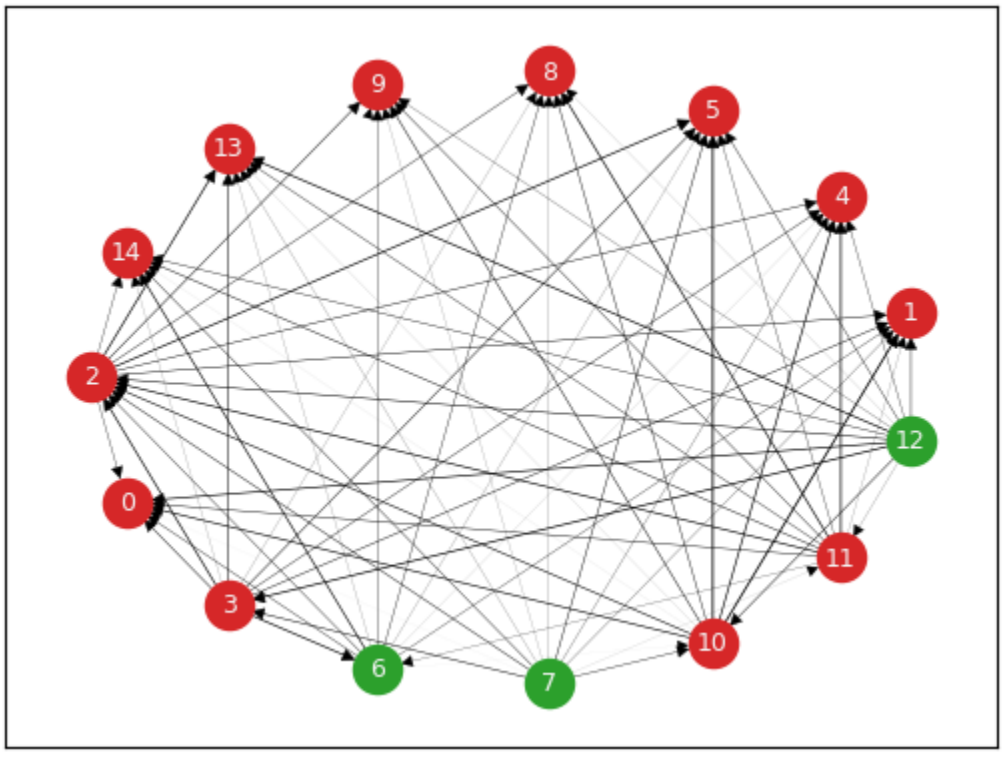
[13]:
```
# Compute labels of reviews in the selection
labels = miner.get_node_labels(
    indices=arg_selection.index.tolist(),
    sources=df_edges["source"].tolist(),
    targets=df_edges["target"].tolist()
)
arg_selection["labels"] = labels
arg_selection
```

[13]:

	argument	score	topics	sentiment	coherence	argument_id	labels
0	I wore these shoe one time, from the airport i...	1	(16, 14, 23, 24, 24)	0.500000	0.535261	46	defeated
1	I usually wear a size 8 and they fit fine. The...	1	(1, 4, 10, 24, 4, 6)	0.496439	0.540030	77	defeated
2	Great shoe! Outside arch is kind of high, but ...	5	(21, 2, 19, 14, 4, 4, 4, 1, 24)	0.659098	0.747863	78	defeated
3	I bought these for gym training - weight class...	2	(13, 2, 7, 7, 18, 7, 24, 10)	0.516497	0.837317	83	defeated
4	Oops! I returned these because I ordered wrong...	1	(14, 24)	0.343750	0.744226	114	defeated
5	I loved these shoes...that is until after abou...	1	(13, 7, 24)	0.599393	0.407310	118	defeated
6	I returned them...found a Ryka pair I liked be...	3	(24,)	0.775000	0.827735	121	supportive
7	I got the impression it's cushiony and comfy b...	3	(0, 2, 24, 11, 9, 18, 6, 24)	0.565749	0.989251	154	supportive
8	Ordered 9(m) received 9 Wide for the second ti...	1	(4, 4, 24)	0.491652	0.546454	205	defeated
9	Returning these. the pictures on here make the...	1	(24, 23, 23, 23, 3, 15)	0.520525	0.507953	254	defeated
10	Tried one in the store and bought it online bu...	2	(24, 24, 23, 1)	0.557920	0.788963	263	defeated
11	I returned these as they were not true to size...	2	(24, 4)	0.509821	0.844705	266	defeated
12	I bought a pair of these in my size, but they ...	3	(4, 4, 7, 0, 24)	0.440069	0.991061	288	supportive
13	Unfortunately, this Flex Supreme does NOT have...	1	(9, 10, 24, 24)	0.494785	0.542248	304	defeated
14	After using this shoes seven times for regular...	1	(7, 14, 20, 24)	0.346269	0.741000	305	defeated

The attacking network of the reviews are visualized as below for better understanding the output.

[14]:
```
import networkx as nx
import matplotlib.pyplot as plt

DG = nx.DiGraph()
DG.add_edges_from(edges)

# graph layout
pos = nx.shell_layout(DG)

# draw nodes
reliable_indices = arg_selection[arg_selection["labels"] == "supportive"].index.tolist()
unreliable_indices = arg_selection[arg_selection["labels"] == "defeated"].index.tolist()
nx.draw_networkx_nodes(DG, pos, nodelist=reliable_indices, node_color="tab:green")
nx.draw_networkx_nodes(DG, pos, nodelist=unreliable_indices, node_color="tab:red")

# draw edges
nx.draw_networkx_edges(DG, pos, width=df_edges["weight"])

# draw labels
labels = {i: i for i in arg_selection.index}
nx.draw_networkx_labels(DG, pos, labels, font_size=9, font_color="whitesmoke")

plt.show()
```



It can be seen from the above plot that review #6, #7, and #12 are considered reliable, while the others are not. Look back to the `arg_selection` table, it seems that those reviews indeed show very high level of consistence, except #6, which is attacked by #3 and #11. But since they are also attacked by some other reviews, #6 is safe.

The weights of edges seems to make sense. One example here is that the attack $11 \rightarrow 6$ is much weaker than $3 \rightarrow 6$,

because #3 describes the totally opposite returning experience than #6 and #11 (can't return in #3 vs. return succesfully in $6 and #11).

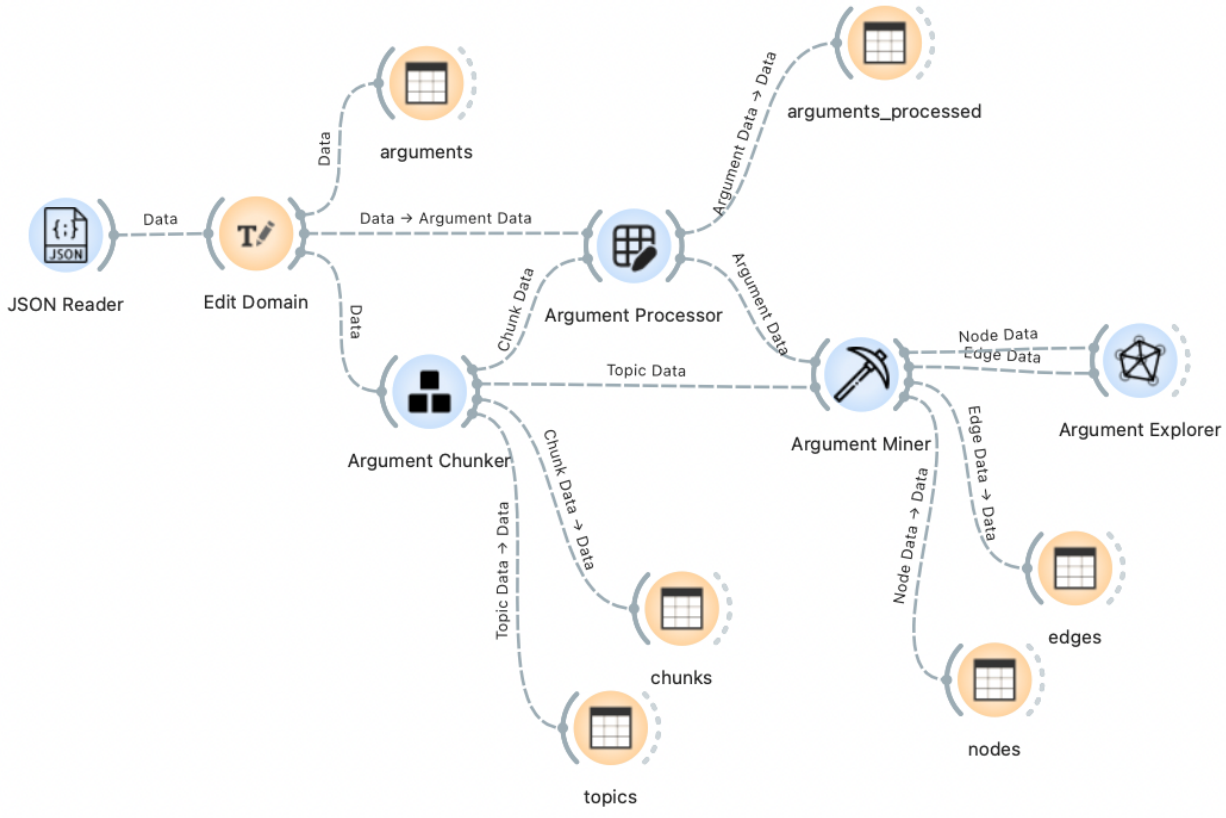
1.4 Use as Widgets on Orange3

We have developed a series of widgets on Orange3, bundling all the essential functionalities for this task. These widgets not only facilitate the analysis but also offer additional visual exploration tools for a more intuitive understanding of the results and insights discovery. Researchers can also benefit from the flexibility of Orange3's built-in functionalities and components to tailor workflows to their specific research needs.

1.4.1 How to use this package on Orange3

It's highly recommended that you first read the documents of Orange3, especially the visual programming session, to understand the basics of building scientific workflows with Orange3. Especially, Orange3 provides a substantial number of built-in widgets, which are quite useful.

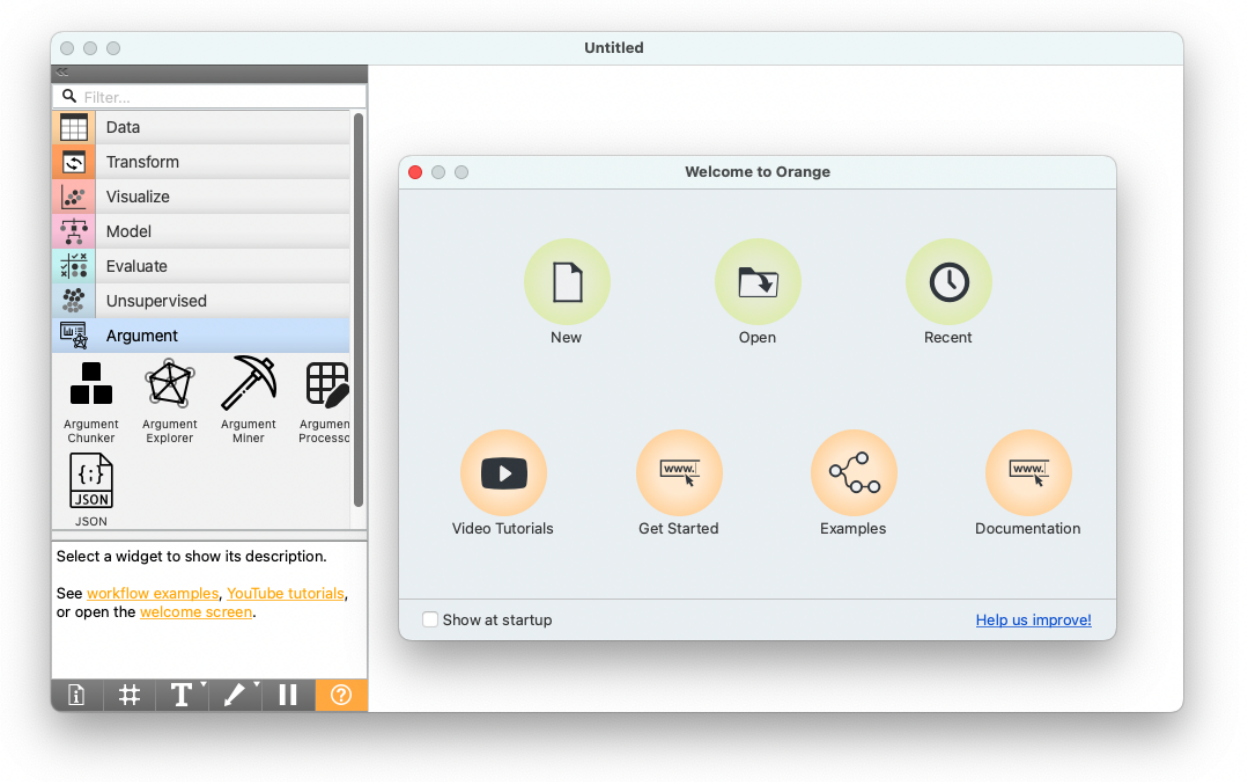
For demonstration purpose, an example workflow is provided in the GitHub repository to showcase how to utilize this library effectively within Orange3.



To run the workflow on your own computer, you need to first install our package, which includes all the dependencies. Then, to start Orange3 GUIs, run the following command in your terminal:

```
python -m Orange.canvas
```

Executing this command will launch the Orange3 GUI, known as the 'canvas.' If your setup is correct, you should observe the following interface, where the 'Argument' add-on is visible on the left panel of widgets. After reaching this point, you can proceed by opening the workflow file and running it sequentially from left to right. Start by double-clicking on the 'JSON Reader' widget to load the example dataset file located in the same folder as the workflow file.



Note: Loading pre-trained language models and performing topic modeling with the *Argument Chunker* widget may take some time, which might make the program appear unresponsive. Kindly exercise patience and wait for a moment.

1.4.2 User manual of the widgets

JSON File Reader

Read a local JSON file and output its data as a table.

Signals

Inputs

- (None)

Outputs:

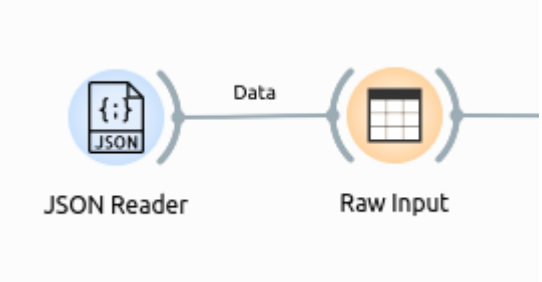
- `Data`: Output data table

Description

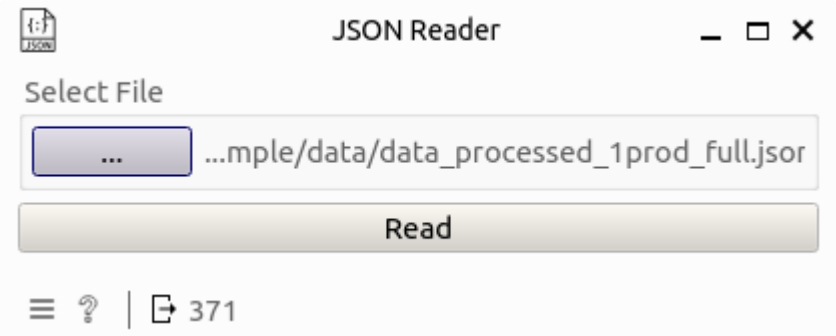
JSON File Reader provides a user interface for selecting and reading a local JSON file. It processes the JSON content, converts it to a table, and outputs the resulting data as a Table type output, which can be used in an Orange workflow for further analysis and visualization.

Example

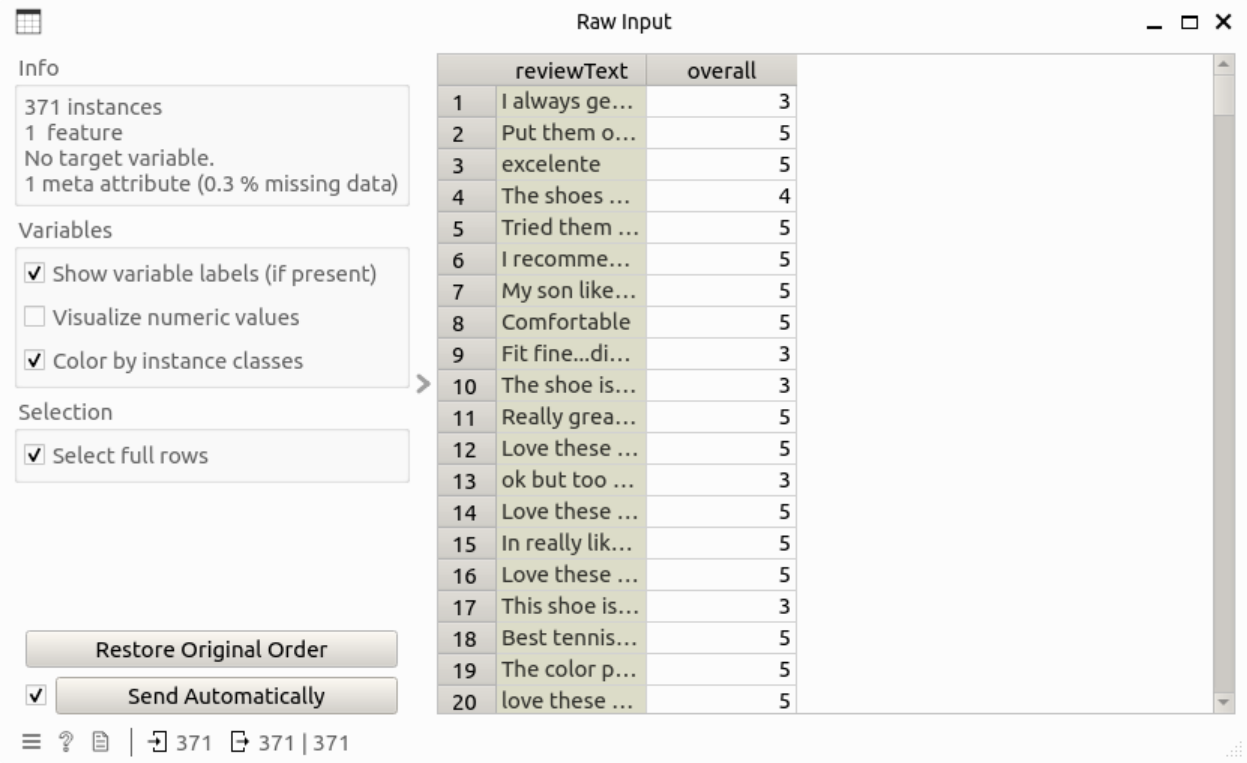
Here is an example workflow of using the JSON file reader widget to read a json file.



Double-clicking the widget opens a sub-interface where users can use the `...` button to select an input file using the system file browser.



Clicking the `Read` button will get the following table as output



Argument Chunker

Segment text-based arguments, enable users to explore the thematic structure of the arguments and their underlying topics.

Signals

Inputs

- `Data`: Data table that contains the argument-level information. This table must contain two columns: *argument* for argument text and *score* that is the corresponding overview score.

Outputs:

- `Chunk Data`: Data table that contains information about argument chunks, including columns: chunk, argument_id, topic, rank, and polarity_score.
- `Topic Data`: Data table that contains information about topics of chunks, including columns: name, Representation, Representative_Docs, keywords, keyword_scores, topic, and count.

Description

Argument Chunker implements the following functions:

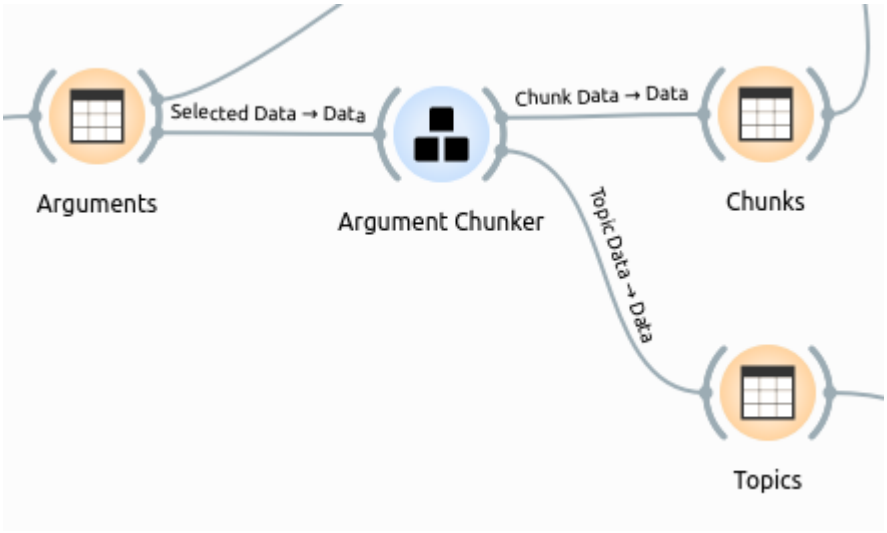
- Chunking: Argument texts are first splitted into sentences, which will then be further parsed into chunks. Dependency parsing is chosen as the default parsing method here. A chunk corpus is generated as the result of this step, including chunk text and the corresponding argument id.
- Topic modeling: Topic modeling is performed on the chunk corpus to learn the themes there. This process is implemented based on a BERT-based topic modeling approach in Python named BERTopic. To summarize this step in brief: chunks are first embedded as high-dimensional vectors through a pre-selected sentence-transformer model; then a dimensionality reduction algorithm is applied to reduce the dimension of the vectors for efficient computation; afterwards, chunks are clustered based on the corresponding vectors, with control of clustering outliers; and finally topics are generated on top of the clustering results.
- Sentiment analysis: Each chunk will be calculated the sentiment (polarity) scores, while the definition of sentiment polarity and an example can be found here.
- Chunk ranking: Chunks are ranked on the argument level, this means each chunk will be given a score of importance within the argument it belongs. This ranking is calculated through PageRank of chunks on their similarity network.

Control

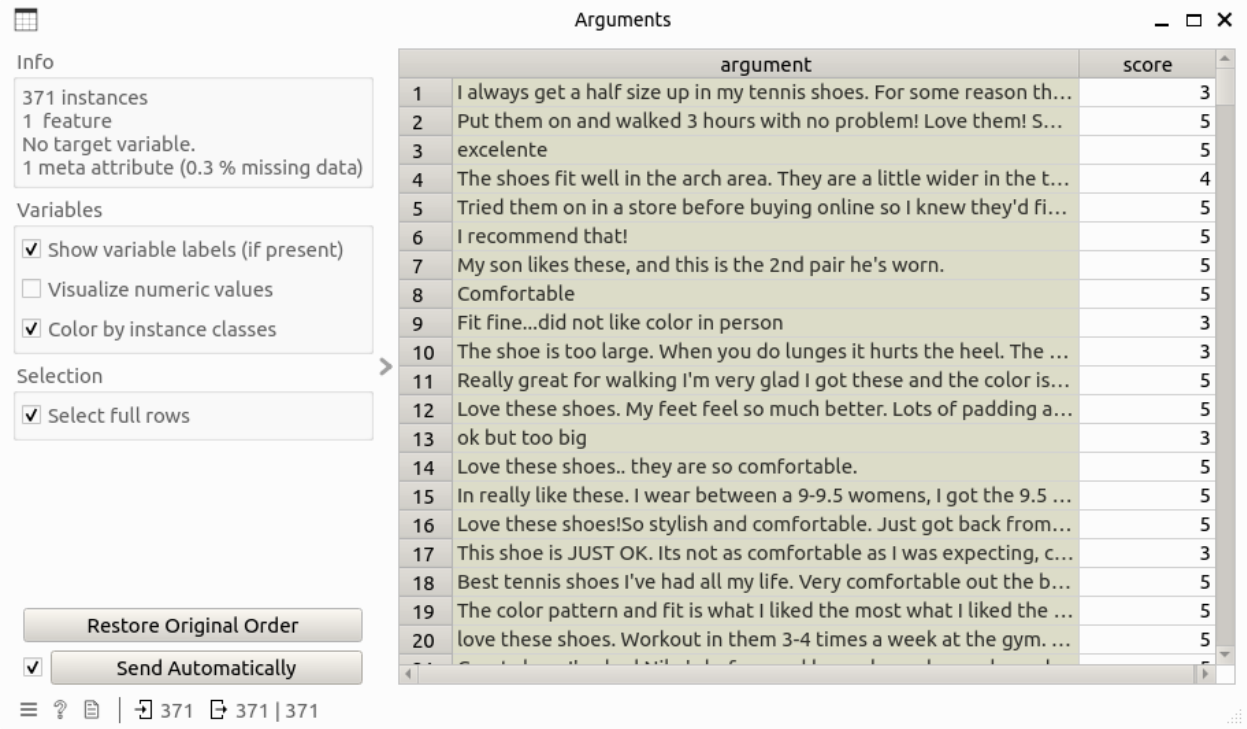
(None)

Example

The following workflow shows how the argument chunker widget works:

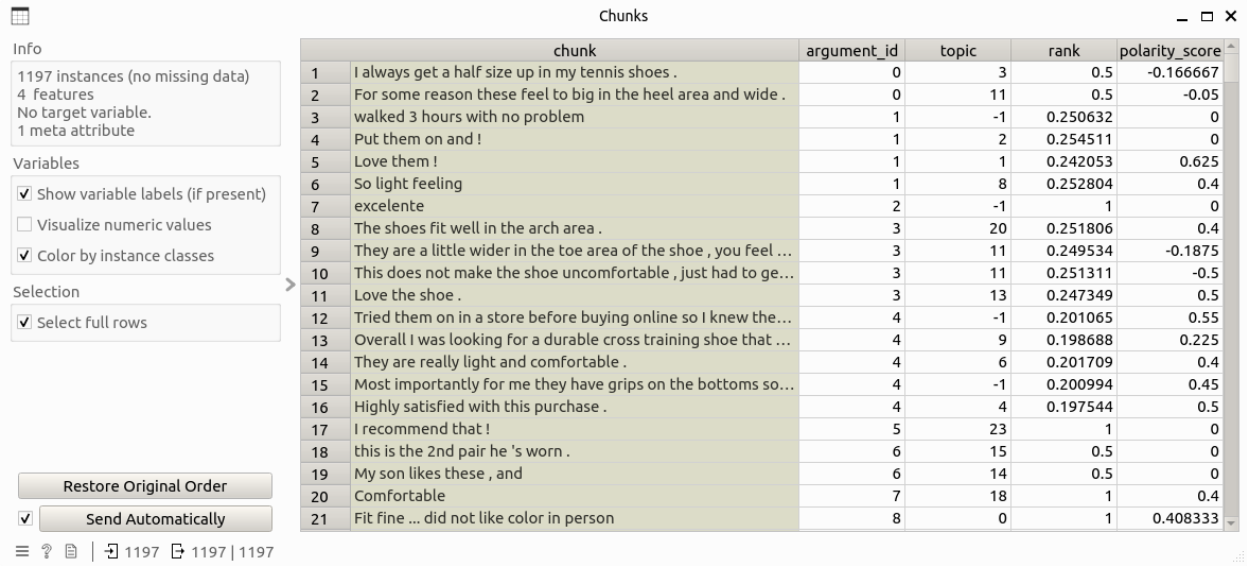


where the input *Arguments* ` table looks like this:



Double-clicking on the widget will show the following subinterface:

After clicking the *chunk*` button and wait for a while, and input table will be processed and two output tables are generated like this.



	name	Representation	Representative_Docs	keywords	keyword_scores	topic	count
1	-1_cushion_expensiv…	['cushion', 'expensiv…	['This is the optimal sn…	['cushion', '…	[0.266282842834…	-1	153
2	0_true_fits_expecte…	['true', 'fits', 'expect…	['Perfect fit , very com…	['true', 'fits'…	[0.463264322971…	0	109
3	1_favorite_absolute_…	['favorite', 'absolute'…	['Absolute favorite .', '…	['favorite', '…	[0.658264124667…	1	71
4	2_row_everyday_wo…	['row', 'everyday', 'w…	["they 're great for we…	['row', 'eve…	[0.401105458768…	2	66
5	3_half_ordered_size…	['half', 'ordered', 'siz…	["tried the same shoe …	['half', 'ord…	[0.524886259397…	3	62
6	4_purchase_satisfied…	['purchase', 'satisfie…	['Very satisfied with t…	['purchase',…	[0.790654259980…	4	46
7	5_slippers_tread_sli…	['slippers', 'tread', 'sl…	['the only reason I trie…	['slippers', '…	[0.407748419841…	5	46
8	6_attractive_sturdy_…	['attractive', 'sturdy'…	['they are nice a room…	['attractive…	[0.495497843044…	6	46
9	7_mesh_job_felt_su…	['mesh', 'job', 'felt', '…	['feel like another laye…	['mesh', 'jo…	[0.632049917525…	7	44
10	8_lightweight_mini…	['lightweight', 'mini…	['Extremely light weig…	['lightweig…	[0.735696561212…	8	38
11	9_camp_cross_boot_…	['camp', 'cross', 'boo…	["I 'm very picky with s…	['camp', 'cr…	[0.459769281157…	9	35
12	10_hurt_pain_hip_bli…	['hurt', 'pain', 'hip', 'b…	['The second day ; ho…	['hurt', 'pai…	[0.596057872915…	10	34
13	11_wide_does_wider…	['wide', 'does', 'wide…	["I know some people …	['wide', 'do…	[0.471508493594…	11	45
14	12_bright_gray_oran…	['bright', 'gray', 'ora…	["so it is n't as gray as …	['bright', 'g…	[0.738897377176…	12	29
15	13_compliments_air…	['compliments', 'airp…	['I have gotten so man…	['complime…	[0.700739769760…	13	31
16	14_loves_daughter_…	['loves', 'daughter', '…	['My daughter loves t…	['loves', 'da…	[0.997307882484…	14	28
17	15_second_pair_pair…	['second', 'pair', 'pair…	['This is my second or …	['second', '…	[0.618717910757…	15	35
18	16_runner_miles_we…	['runner', 'miles', 'we…	['I do treadmill , stair…	['runner', '…	[0.726615678997…	16	26
19	17_excellent_aweso…	['excellent', 'aweso…	['Excellent … ! ! !', 'EXC…	['excellent'…	[1.123185471721…	17	29
20	18_right_flexible_fir…	['right', 'flexible', 'fir…	['comfortable', "It 's n…	['right', 'fle…	[0.696150041629…	18	30
21	19_owned_nike_nike…	['owned', 'nike', 'nik…	["I ordered the Nike W…	['owned', 'n…	[0.584149667250…	19	36

Argument Processor

Calculate argument-level metrics and measures.

Signals

Inputs

- `Argument Data`: Data table that contains the argument-level information. This table must contain two columns: *argument* for argument text and *score* that is the corresponding overview score.
- `Chunk Data`: Data table that contains information about argument chunks, including columns: chunk, argument_id, topic, rank, and polarity_score.

Outputs:

- `Argument Data`: Data table that contains additional information of arguments to the input data table, including columns: argument, score, topics, readability, sentiment, and coherence.

Description

Argument Processor implements the following functions:

- Topic merging: For each argument, its topic is defined as the combination of the topics of chunks that belongs to this one.
- Argument readability computing: The Flesh-Kincaid reading score is computed for each arugment, check this link for more information.

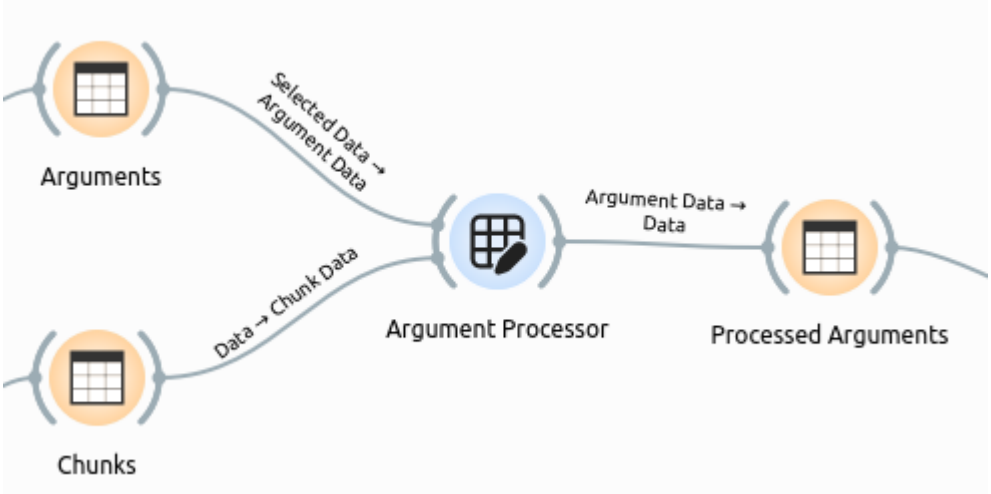
- Argument Coherence computing: In this step, the coherence between the sentiment and overall score of arguments are calculated, where the sentiment score of argument is calculated as the sum of sentiment scores of corresponding chunks, weighted by chunk ranks.

Control

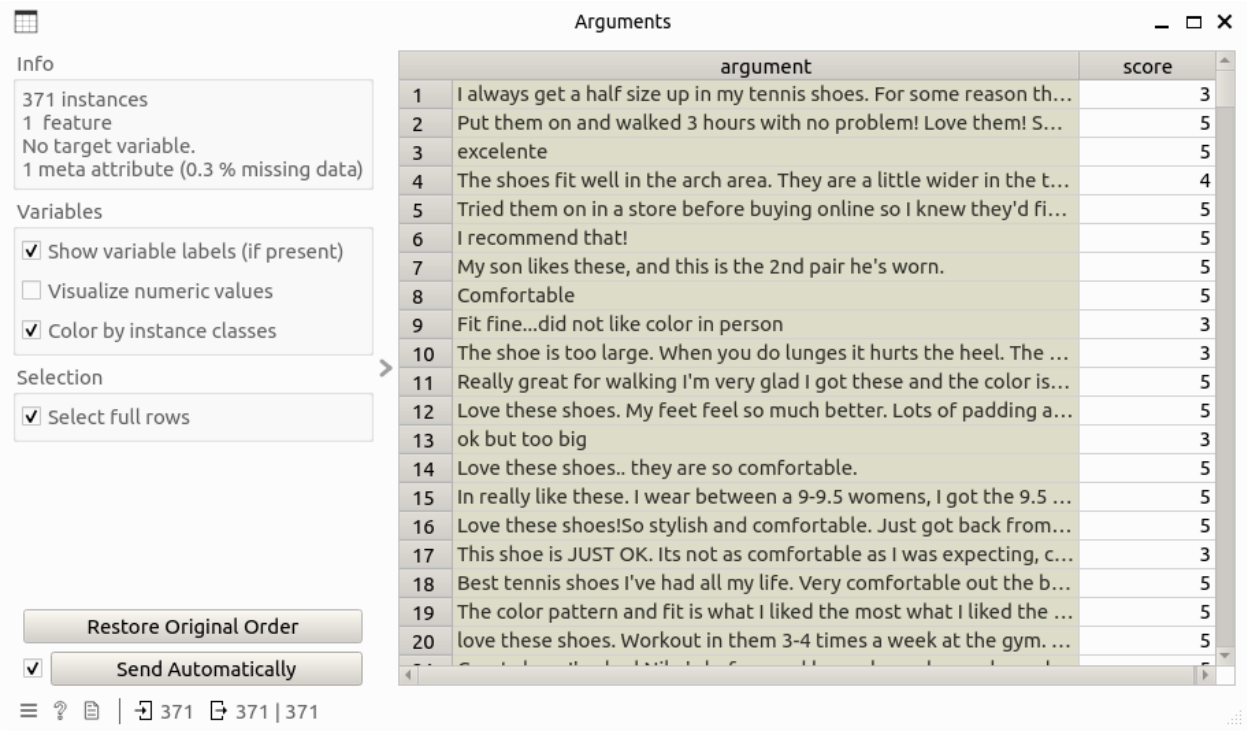
(None)

Example

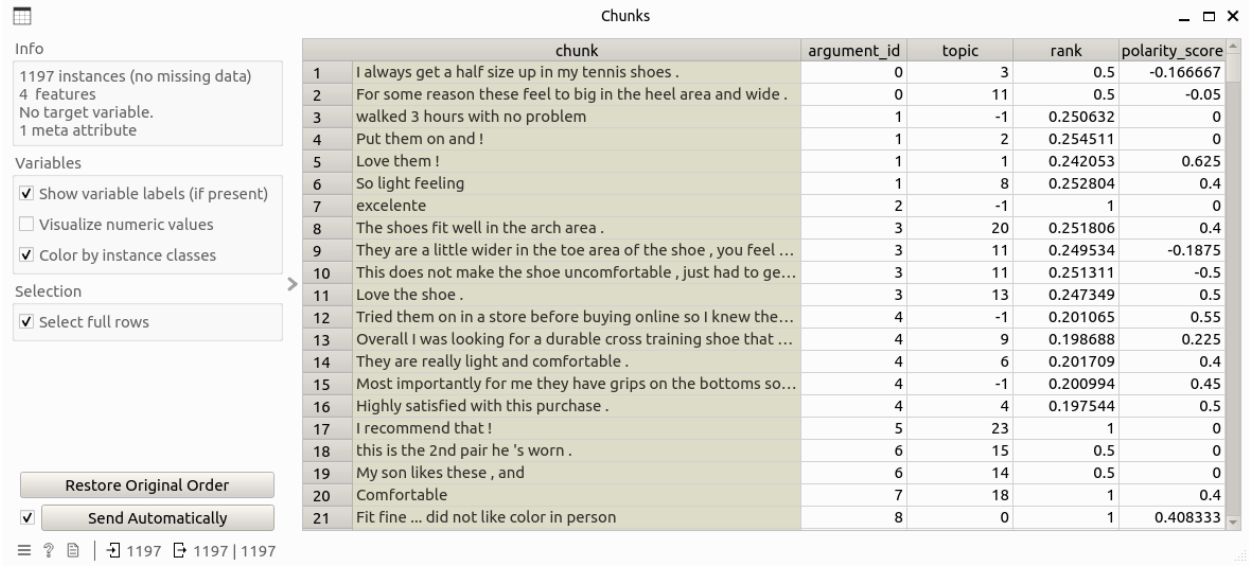
Here is an example workflow that shows how the argument processor widget works:



where the input *Arguments* and *Chunks* table look like this:

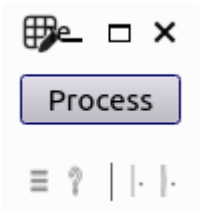


	argument	score
1	I always get a half size up in my tennis shoes. For some reason th…	3
2	Put them on and walked 3 hours with no problem! Love them! S…	5
3	excelente	5
4	The shoes fit well in the arch area. They are a little wider in the t…	4
5	Tried them on in a store before buying online so I knew they'd fi…	5
6	I recommend that!	5
7	My son likes these, and this is the 2nd pair he's worn.	5
8	Comfortable	5
9	Fit fine...did not like color in person	3
10	The shoe is too large. When you do lunges it hurts the heel. The …	3
11	Really great for walking I'm very glad I got these and the color is…	5
12	Love these shoes. My feet feel so much better. Lots of padding a…	5
13	ok but too big	3
14	Love these shoes.. they are so comfortable.	5
15	In really like these. I wear between a 9-9.5 womens, I got the 9.5 …	5
16	Love these shoes!So stylish and comfortable. Just got back from…	5
17	This shoe is JUST OK. Its not as comfortable as I was expecting, c…	3
18	Best tennis shoes I've had all my life. Very comfortable out the b…	5
19	The color pattern and fit is what I liked the most what I liked the …	5
20	love these shoes. Workout in them 3-4 times a week at the gym. …	5

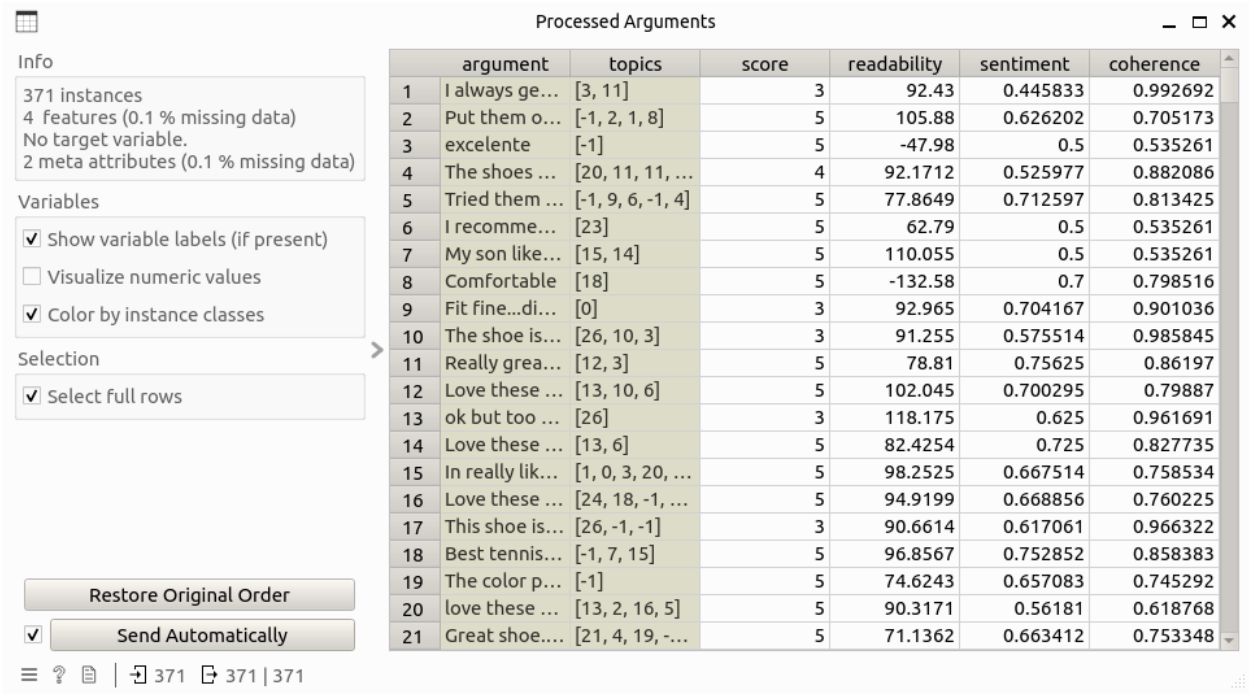


	chunk	argument_id	topic	rank	polarity_score
1	I always get a half size up in my tennis shoes .	0	3	0.5	-0.166667
2	For some reason these feel to big in the heel area and wide .	0	11	0.5	-0.05
3	walked 3 hours with no problem	1	-1	0.250632	0
4	Put them on and !	1	2	0.254511	0
5	Love them !	1	1	0.242053	0.625
6	So light feeling	1	8	0.252804	0.4
7	excelente	2	-1	1	0
8	The shoes fit well in the arch area .	3	20	0.251806	0.4
9	They are a little wider in the toe area of the shoe , you feel …	3	11	0.249534	-0.1875
10	This does not make the shoe uncomfortable , just had to ge…	3	11	0.251311	-0.5
11	Love the shoe .	3	13	0.247349	0.5
12	Tried them on in a store before buying online so I knew the…	4	-1	0.201065	0.55
13	Overall I was looking for a durable cross training shoe that …	4	9	0.198688	0.225
14	They are really light and comfortable .	4	6	0.201709	0.4
15	Most importantly for me they have grips on the bottoms so…	4	-1	0.200994	0.45
16	Highly satisfied with this purchase .	4	4	0.197544	0.5
17	I recommend that !	5	23	1	0
18	this is the 2nd pair he 's worn .	6	15	0.5	0
19	My son likes these , and	6	14	0.5	0
20	Comfortable	7	18	1	0.4
21	Fit fine ... did not like color in person	8	0	1	0.408333

Double-clicking the widget opens the subinterface like this:



By clicking on the `Process` button and wait for a while, the result data table will be computed like this:



	argument	topics	score	readability	sentiment	coherence
1	I always ge…	[3, 11]	3	92.43	0.445833	0.992692
2	Put them o…	[-1, 2, 1, 8]	5	105.88	0.626202	0.705173
3	excelente	[-1]	5	-47.98	0.5	0.535261
4	The shoes …	[20, 11, 11, …	4	92.1712	0.525977	0.882086
5	Tried them …	[-1, 9, 6, -1, 4]	5	77.8649	0.712597	0.813425
6	I recomme…	[23]	5	62.79	0.5	0.535261
7	My son like…	[15, 14]	5	110.055	0.5	0.535261
8	Comfortable	[18]	5	-132.58	0.7	0.798516
9	Fit fine...di…	[0]	3	92.965	0.704167	0.901036
10	The shoe is…	[26, 10, 3]	3	91.255	0.575514	0.985845
11	Really grea…	[12, 3]	5	78.81	0.75625	0.86197
12	Love these …	[13, 10, 6]	5	102.045	0.700295	0.79887
13	ok but too …	[26]	3	118.175	0.625	0.961691
14	Love these …	[13, 6]	5	82.4254	0.725	0.827735
15	In really lik…	[1, 0, 3, 20, …	5	98.2525	0.667514	0.758534
16	Love these …	[24, 18, -1, …	5	94.9199	0.668856	0.760225
17	This shoe is…	[26, -1, -1]	3	90.6614	0.617061	0.966322
18	Best tennis…	[-1, 7, 15]	5	96.8567	0.752852	0.858383
19	The color p…	[-1]	5	74.6243	0.657083	0.745292
20	love these …	[13, 2, 16, 5]	5	90.3171	0.56181	0.618768
21	Great shoe.…	[21, 4, 19, -…	5	71.1362	0.663412	0.753348

Argument Miner

Generate attacking relationship information of arguments from argument corpus.

Signals

Inputs

- `Argument Data`: Data table that contains additional information of arguments to the input data table, including columns: argument, score, topics, readability, sentiment, and coherence.

Outputs:

- `Edge Data`: Data table that contains edge information of the argument attacking network, including columns: source, target, weight.
- `Node Data`: Data table that contains node information of the argument attacking network, including one additional column than the input argument data table that is label.

Description

Argument Miner has the following functions:

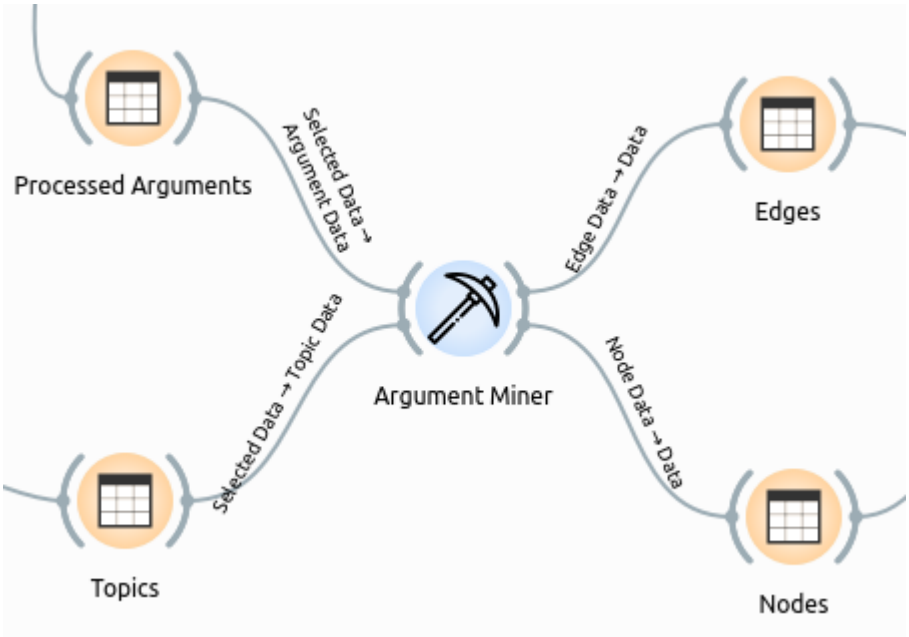
- Attacking network mining: Based on the input table, an argument attacking network is learned for a given topic, where nodes are arguments that cover the given topic, and edges represent a kind of disagreeing relation between arguments. Weights of edges are computed as the coherence gap of the corresponding two nodes, while direction is determined as from high to low coherent node.
- Node labeling: Based on the learned structure of the attacking network, nodes (arguments) are classified and labeled as either *supportive* ` or *defeated*, which can be simply understood as reliable or non-reliable. There are three roles of labeling the nodes:
  - If a node is not being attacked by any other nodes, this node is labeled as supportive.
  - If all attackers of a node are being attacked by some other nodes, this node is labeled as supportive.
  - If a node is not supportive, it is labeled as defeated.

Control

- `Select topic`: a combo box that allows user to choose a topic to generate the attacking network.

Example

Here is an example workflow that shows how the argument miner widget works:



where the input *Processed Arguments* and *Topics* ` tables are as follows:

Processed Arguments

Info

371 instances
4 features (0.1 % missing data)
No target variable.
2 meta attributes (0.1 % missing data)

Variables

- [x] Show variable labels (if present)
- [ ] Visualize numeric values
- [x] Color by instance classes

Selection

- [x] Select full rows

Restore Original Order

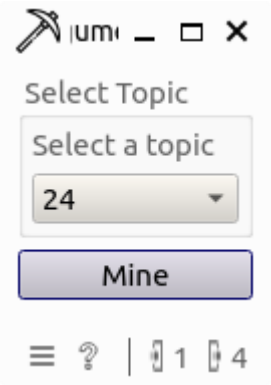
- [x] Send Automatically

371 | 371 | 371

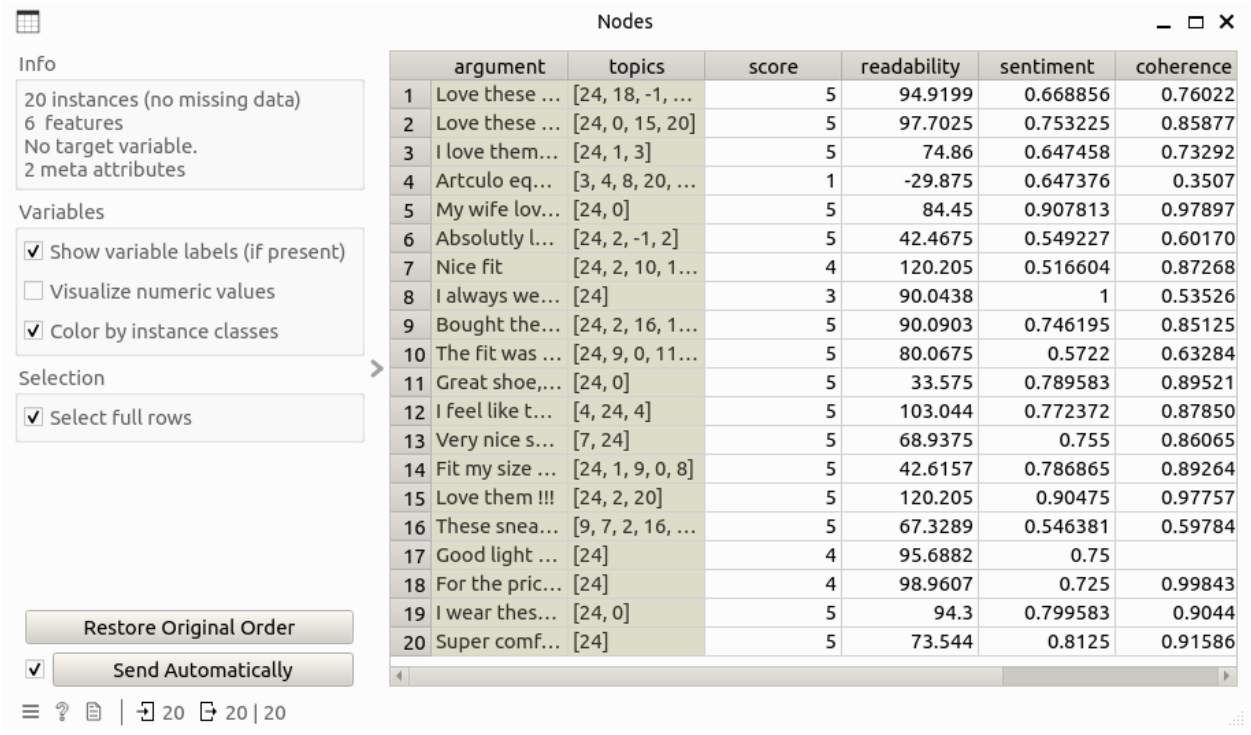
	argument	topics	score	readability	sentiment	coherence
1	I always ge…	[3, 11]	3	92.43	0.445833	0.992692
2	Put them o…	[-1, 2, 1, 8]	5	105.88	0.626202	0.705173
3	excelente	[-1]	5	-47.98	0.5	0.535261
4	The shoes …	[20, 11, 11, …	4	92.1712	0.525977	0.882086
5	Tried them …	[-1, 9, 6, -1, 4]	5	77.8649	0.712597	0.813425
6	I recomme…	[23]	5	62.79	0.5	0.535261
7	My son like…	[15, 14]	5	110.055	0.5	0.535261
8	Comfortable	[18]	5	-132.58	0.7	0.798516
9	Fit fine...di…	[0]	3	92.965	0.704167	0.901036
10	The shoe is…	[26, 10, 3]	3	91.255	0.575514	0.985845
11	Really grea…	[12, 3]	5	78.81	0.75625	0.86197
12	Love these …	[13, 10, 6]	5	102.045	0.700295	0.79887
13	ok but too …	[26]	3	118.175	0.625	0.961691
14	Love these …	[13, 6]	5	82.4254	0.725	0.827735
15	In really lik…	[1, 0, 3, 20, …	5	98.2525	0.667514	0.758534
16	Love these …	[24, 18, -1, …	5	94.9199	0.668856	0.760225
17	This shoe is…	[26, -1, -1]	3	90.6614	0.617061	0.966322
18	Best tennis…	[-1, 7, 15]	5	96.8567	0.752852	0.858383
19	The color p…	[-1]	5	74.6243	0.657083	0.745292
20	love these …	[13, 2, 16, 5]	5	90.3171	0.56181	0.618768
21	Great shoe.…	[21, 4, 19, -…	5	71.1362	0.663412	0.753348

	name	Representation	Representative_Docs	keywords	keyword_scores	topic	count
1	-1_cushion_expensiv…	['cushion', 'expensiv…	['This is the optimal sn…	['cushion', '…	[0.266282842834…	-1	153
2	0_true_fits_expecte…	['true', 'fits', 'expect…	['Perfect fit , very com…	['true', 'fits'…	[0.463264322971…	0	109
3	1_favorite_absolute_…	['favorite', 'absolute'…	['Absolute favorite .', '…	['favorite', '…	[0.658264124667…	1	71
4	2_row_everyday_wo…	['row', 'everyday', 'w…	["they 're great for we…	['row', 'eve…	[0.401105458768…	2	66
5	3_half_ordered_size…	['half', 'ordered', 'siz…	["tried the same shoe …	['half', 'ord…	[0.524886259397…	3	62
6	4_purchase_satisfied…	['purchase', 'satisfie…	['Very satisfied with t…	['purchase',…	[0.790654259980…	4	46
7	5_slippers_tread_sli…	['slippers', 'tread', 'sl…	['the only reason I trie…	['slippers', '…	[0.407748419841…	5	46
8	6_attractive_sturdy_…	['attractive', 'sturdy'…	['they are nice a room…	['attractive…	[0.495497843044…	6	46
9	7_mesh_job_felt_su…	['mesh', 'job', 'felt', '…	['feel like another laye…	['mesh', 'jo…	[0.632049917525…	7	44
10	8_lightweight_mini…	['lightweight', 'mini…	['Extremely light weig…	['lightweig…	[0.735696561212…	8	38
11	9_camp_cross_boot_…	['camp', 'cross', 'boo…	["I 'm very picky with s…	['camp', 'cr…	[0.459769281157…	9	35
12	10_hurt_pain_hip_bli…	['hurt', 'pain', 'hip', 'b…	['The second day ; ho…	['hurt', 'pai…	[0.596057872915…	10	34
13	11_wide_does_wider…	['wide', 'does', 'wide…	["I know some people …	['wide', 'do…	[0.471508493594…	11	45
14	12_bright_gray_oran…	['bright', 'gray', 'ora…	["so it is n't as gray as …	['bright', 'g…	[0.738897377176…	12	29
15	13_compliments_air…	['compliments', 'airp…	['I have gotten so man…	['complime…	[0.700739769760…	13	31
16	14_loves_daughter_…	['loves', 'daughter', '…	['My daughter loves t…	['loves', 'da…	[0.997307882484…	14	28
17	15_second_pair_pair…	['second', 'pair', 'pair…	['This is my second or …	['second', '…	[0.618717910757…	15	35
18	16_runner_miles_we…	['runner', 'miles', 'we…	['I do treadmill , stair…	['runner', '…	[0.726615678997…	16	26
19	17_excellent_aweso…	['excellent', 'aweso…	['Excellent … ! ! !', 'EXC…	['excellent'…	[1.123185471721…	17	29
20	18_right_flexible_fir…	['right', 'flexible', 'fir…	['comfortable', "It 's n…	['right', 'fle…	[0.696150041629…	18	30
21	19_owned_nike_nike…	['owned', 'nike', 'nik…	["I ordered the Nike W…	['owned', 'n…	[0.584149667250…	19	36

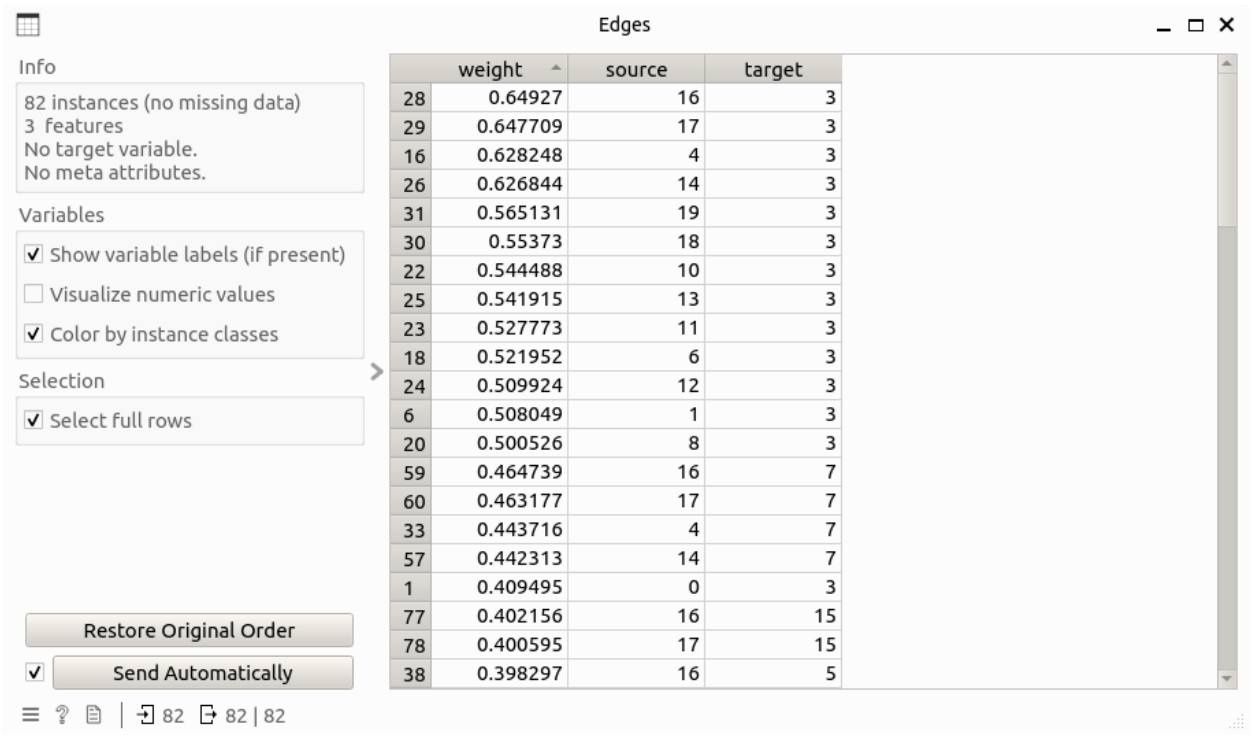
Double-clicking the widget opens the subinterface of the widget like this:



By selecting the target topic (24 in this example) and clicking the `mine` button, the result *Nodes* ` and *Edges* ` tables are generated as follows:

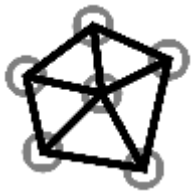


	argument	topics	score	readability	sentiment	coherence
1	Love these …	[24, 18, -1, …	5	94.9199	0.668856	0.76022
2	Love these …	[24, 0, 15, 20]	5	97.7025	0.753225	0.85877
3	I love them…	[24, 1, 3]	5	74.86	0.647458	0.73292
4	Artculo eq…	[3, 4, 8, 20, …	1	-29.875	0.647376	0.3507
5	My wife lov…	[24, 0]	5	84.45	0.907813	0.97897
6	Absolutly l…	[24, 2, -1, 2]	5	42.4675	0.549227	0.60170
7	Nice fit	[24, 2, 10, 1…	4	120.205	0.516604	0.87268
8	I always we…	[24]	3	90.0438	1	0.53526
9	Bought the…	[24, 2, 16, 1…	5	90.0903	0.746195	0.85125
10	The fit was …	[24, 9, 0, 11…	5	80.0675	0.5722	0.63284
11	Great shoe,…	[24, 0]	5	33.575	0.789583	0.89521
12	I feel like t…	[4, 24, 4]	5	103.044	0.772372	0.87850
13	Very nice s…	[7, 24]	5	68.9375	0.755	0.86065
14	Fit my size …	[24, 1, 9, 0, 8]	5	42.6157	0.786865	0.89264
15	Love them !!!	[24, 2, 20]	5	120.205	0.90475	0.97757
16	These snea…	[9, 7, 2, 16, …	5	67.3289	0.546381	0.59784
17	Good light …	[24]	4	95.6882	0.75	
18	For the pric…	[24]	4	98.9607	0.725	0.99843
19	I wear thes…	[24, 0]	5	94.3	0.799583	0.9044
20	Super comf…	[24]	5	73.544	0.8125	0.91586



	weight	source	target
28	0.64927	16	3
29	0.647709	17	3
16	0.628248	4	3
26	0.626844	14	3
31	0.565131	19	3
30	0.55373	18	3
22	0.544488	10	3
25	0.541915	13	3
23	0.527773	11	3
18	0.521952	6	3
24	0.509924	12	3
6	0.508049	1	3
20	0.500526	8	3
59	0.464739	16	7
60	0.463177	17	7
33	0.443716	4	7
57	0.442313	14	7
1	0.409495	0	3
77	0.402156	16	15
78	0.400595	17	15
38	0.398297	16	5

Argument Explorer



Network visualization of argument attacking relationships.

Inputs

- `Edge Data`: Data table that contains edge information of the argument attacking network, including columns: source, target, weight.
- `Node Data`: Data table that contains node information of the argument attacking network.

Outputs

- `Selected Data`: Data table that contains information of the selected nodes.

Description

Argument Explorer has the following function:

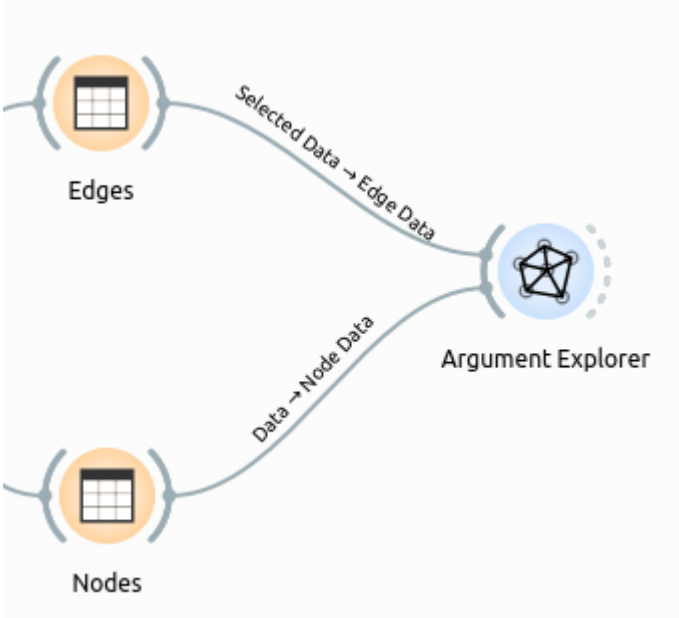
- Network visualization: The argument attacking network is visualized with node coler representing their labels (green for supportive and red for defeated) and edge width for showing weights.
- Node selection: This widget allows to select node(s) and this will update the output table that contains the information of selected nodes. Also, when a node is selected, all the edges relevant to that node will be highlighed by hiding the unrelevant edges.
- Layouting: A set of network layout can be chosen, that include *spring*, *multipartite*, *kamada kawai*, and *spectral*.
- Navigation: This widget supports a series of navigating functions for better observing the network, that include *zooming*, *panning*, and *centralizing*. Also, by hovering over a node, the relevant meta information of that node will be shown in the popping-up tooltips.

Control

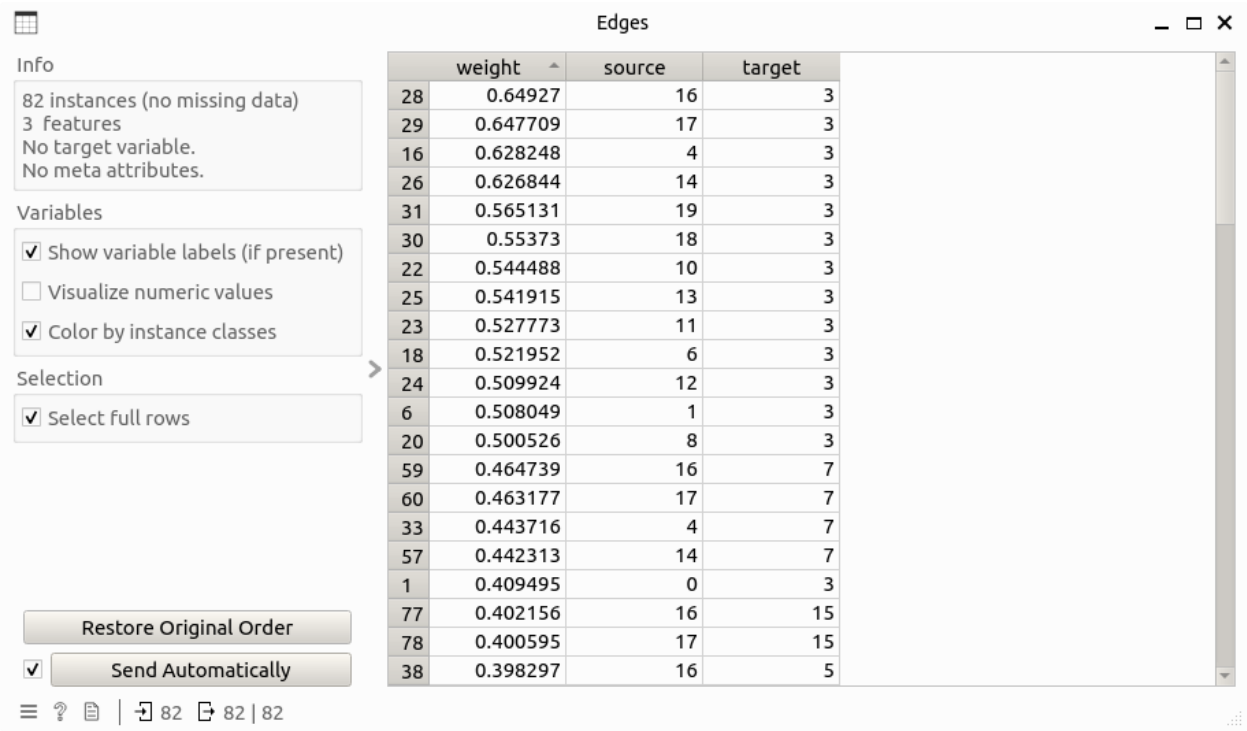
- `Graph layout`: Layout used for positing nodes and edges in the network.
- `Node sparsity`: Spatial closeness of nodes, in range of [1, 10]
- `Zoom/Select`: Navigation tools for better observing the network.
- `Send Automatically`: if the checkbox is enabled, the information of selected nodes will be automatically sent to the output data.

Example

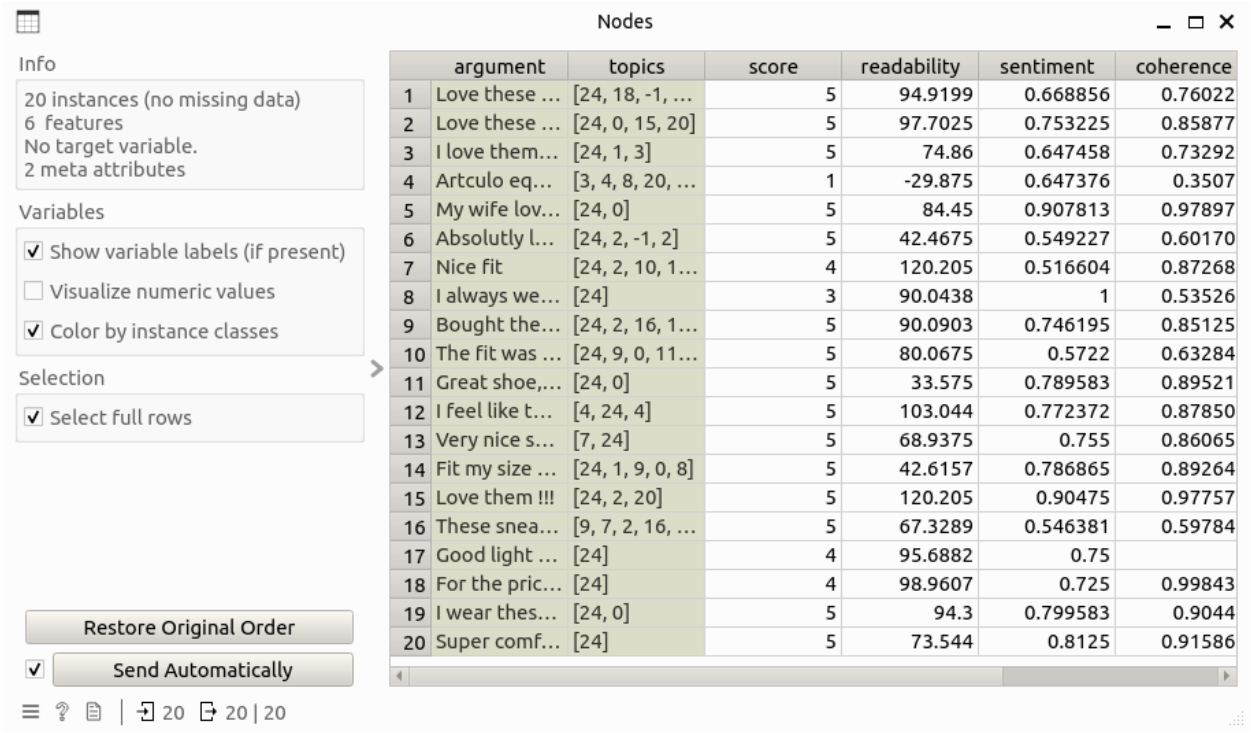
Here is an example workflow that shows how the widget works:



where the input **Edges** and **Nodes** table look like this:

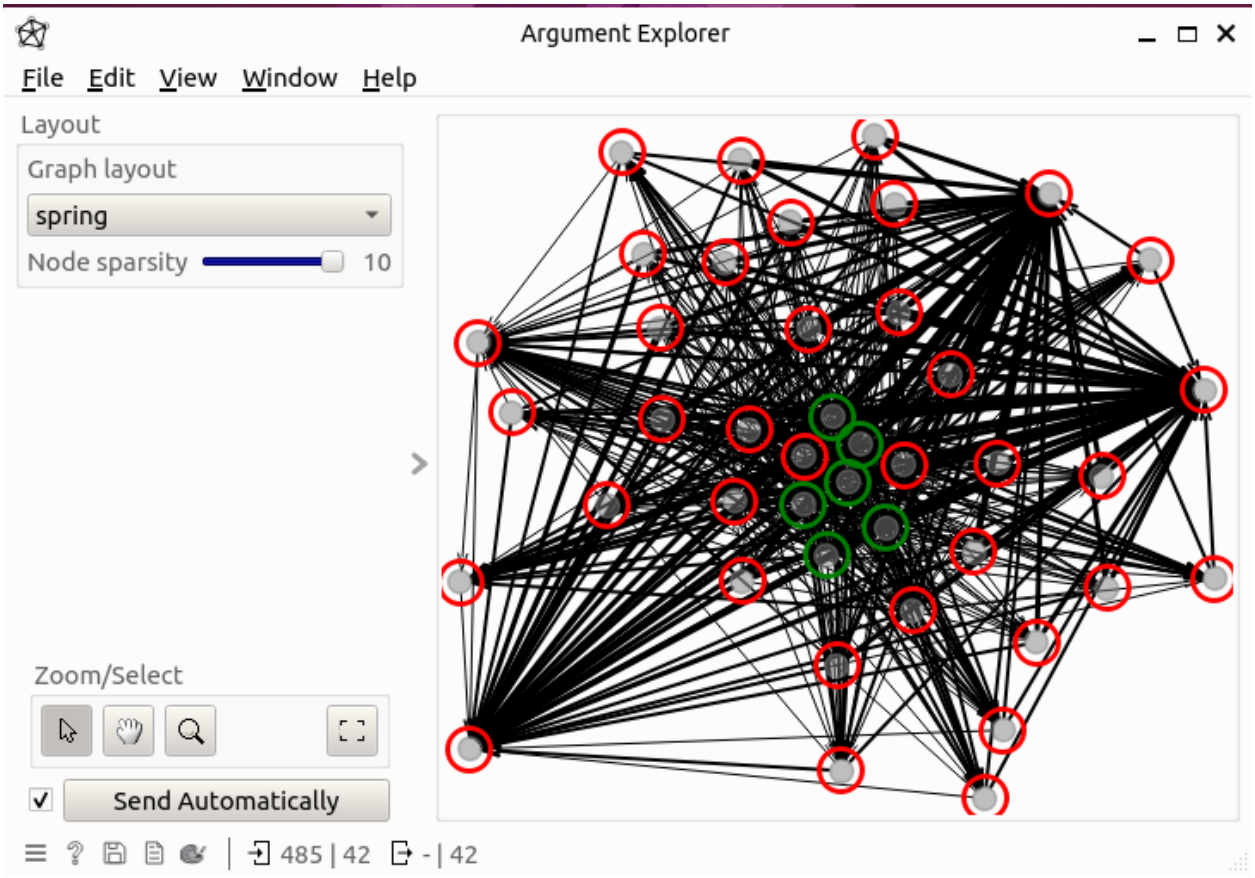


	weight	source	target
28	0.64927	16	3
29	0.647709	17	3
16	0.628248	4	3
26	0.626844	14	3
31	0.565131	19	3
30	0.55373	18	3
22	0.544488	10	3
25	0.541915	13	3
23	0.527773	11	3
18	0.521952	6	3
24	0.509924	12	3
6	0.508049	1	3
20	0.500526	8	3
59	0.464739	16	7
60	0.463177	17	7
33	0.443716	4	7
57	0.442313	14	7
1	0.409495	0	3
77	0.402156	16	15
78	0.400595	17	15
38	0.398297	16	5



	argument	topics	score	readability	sentiment	coherence
1	Love these …	[24, 18, -1, …	5	94.9199	0.668856	0.76022
2	Love these …	[24, 0, 15, 20]	5	97.7025	0.753225	0.85877
3	I love them…	[24, 1, 3]	5	74.86	0.647458	0.73292
4	Artculo eq…	[3, 4, 8, 20, …	1	-29.875	0.647376	0.3507
5	My wife lov…	[24, 0]	5	84.45	0.907813	0.97897
6	Absolutly l…	[24, 2, -1, 2]	5	42.4675	0.549227	0.60170
7	Nice fit	[24, 2, 10, 1…	4	120.205	0.516604	0.87268
8	I always we…	[24]	3	90.0438	1	0.53526
9	Bought the…	[24, 2, 16, 1…	5	90.0903	0.746195	0.85125
10	The fit was …	[24, 9, 0, 11…	5	80.0675	0.5722	0.63284
11	Great shoe,…	[24, 0]	5	33.575	0.789583	0.89521
12	I feel like t…	[4, 24, 4]	5	103.044	0.772372	0.87850
13	Very nice s…	[7, 24]	5	68.9375	0.755	0.86065
14	Fit my size …	[24, 1, 9, 0, 8]	5	42.6157	0.786865	0.89264
15	Love them !!!	[24, 2, 20]	5	120.205	0.90475	0.97757
16	These snea…	[9, 7, 2, 16, …	5	67.3289	0.546381	0.59784
17	Good light …	[24]	4	95.6882	0.75	
18	For the pric…	[24]	4	98.9607	0.725	0.99843
19	I wear thes…	[24, 0]	5	94.3	0.799583	0.9044
20	Super comf…	[24]	5	73.544	0.8125	0.91586

The result network can be observed directly from the widget subinterface:



1.5 API Reference

This page contains auto-generated API reference documentation[1].

1.5.1 orangearg

Subpackages

orangearg.argument

Subpackages

orangearg.argument.miner

Submodules

orangearg.argument.miner.chunker

Argument chunker module

Module Contents

Classes

TopicModel	Topic modeling class.

Functions

load_nlp_pipe(model_name)	Download the required nlp pipe if not exist
get_chunk(→ Tuple[List[int], List[str]])	Split documents of a given corpus into chunks.
get_chunk_polarity_score(chunks)	Compute polarity score of each chunk in the given list.
get_chunk_topic(chunks)	Get topic information and embedding vectors of chunks via topic modeling.
get_chunk_rank(arg_ids, embeds)	In each argument, comput rank of chunks within.
get_chunk_table(arg_ids, chunks, p_scores, topics, ranks)	Given all the measures of chunks, generate and return the chunk table as a pandas dataframe, with pre-defined column names.

orangearg.argument.miner.chunker.**load_nlp_pipe**(*model_name: str*)

Download the required nlp pipe if not exist

Parameters

model_name (*str*) – name of the nlp pipe, a full list of models can be found from https://spacy.io/usage/models.

[1] Created with sphinx-autoapi

Returns
The spacy nlp model.

orangearg.argument.miner.chunker.**get_chunk**(*docs: List[str]*) → Tuple[List[int], List[str]]

Split documents of a given corpus into chunks.

A chunk can be considered as a meaningful clause, which can be part of a sentence. For instance, the sentence "I like the color of this car but it's too expensive." will be splitted as two chunks, which are "I like the color of this car" and "but it's too expensive". A dependency parser is implemented for doing this job.

Parameters
docs (*List[str]*) – The input corpus.

Returns
ids of the arguments that the chunks belongs to. List[str]: chunk text.

Return type
List[int]

orangearg.argument.miner.chunker.**get_chunk_polarity_score**(*chunks: List[str]*)

Compute polarity score of each chunk in the given list.

The polarity score is a float within the range [-1.0, 1.0], where 0 means neutral, + means positive, and - means negative.

Parameters
chunks (*List[str]*) – chunk list

Returns
polarity scores of the given chunks

Return type
List[float]

orangearg.argument.miner.chunker.**get_chunk_topic**(*chunks: List[str]*)

Get topic information and embedding vectors of chunks via topic modeling.

Parameters
chunks (*List[str]*) – chunk list.

Returns
topic ids of chunks. np.ndarray: embedding vectors of chunks. pd.DataFrame: Table of topic information.

Return type
List[int]

orangearg.argument.miner.chunker.**get_chunk_rank**(*arg_ids: List[int]*, *embeds: numpy.ndarray*)

In each argument, comput rank of chunks within.

Rank can be understood as importance of chunks. This function computes the relative importance of chunks within arguments they belong to. This is done by applying the Pagerank algorithm, where similarity is computed as the cosine similarity of chunk embedding vectors.

Parameters

- **arg_ids** (*List[int]*) – ids of arguments that chunks belongs to.
- **embeds** (*np.ndarray*) – embedding vectors of chunks.

Returns
rank of chunks

Return type
List[float]

orangearg.argument.miner.chunker.**get_chunk_table**(*arg_ids: List[int]*, *chunks: List[str]*, *p_scores: List[float]*, *topics: List[int]*, *ranks: List[float]*)

Given all the measures of chunks, generate and return the chunk table as a pandas dataframe, with pre-defined column names.

Parameters

- **arg_ids** (*List[int]*) – ids of arguments that chunks belong to
- **chunks** (*List[str]*) – chunk text
- **p_scores** (*List[float]*) – polarity score of chunks
- **topics** (*List[int]*) – topic id of chunks
- **ranks** (*List[float]*) – rank of chunks

Returns
chunk table

Return type
pd.DataFrame

class orangearg.argument.miner.chunker.**TopicModel**

Topic modeling class.

Functions are implemented based on the BERTopic model. For now, the topic model is setup with a set of default parameters of the sub-models. However, it should be possible that the user can config it further. This will be a next step.

_rd_model (

obj:'UMAP'): instance of UMAP algorithm as the dimensionality reduction sub-model.

model (

obj:'BERTopic'): the topic model that applied the sub-models predefined.

init_model(*transformer: str = 'all-mpnet-base-v1'*, *n_components: int = 5*, *min_cluster_size: int = 10*)

Initialize the topic model by indicating a number of arguments.

Parameters

- **transformer** (*str, optional*) – Name of the sentence embedding model. Defaults to "all-mpnet-base-v1". A list of pretrained models can be found here: https://www.sbert.net/docs/pretrained_models.html.
- **n_components** (*int, optional*) – Number of dimensions after reduction. Defaults to 5.
- **min_cluster_size** (*int, optional*) – Minimum size of clusters for the clustering algorithm. Defaults to 5.

fit_transform_reduced(*docs: List[str]*) → List[int]

Further reduce outliers from the result of the fit_transform function.

Note that BERTopic is a clustering approach, which means that it doesn not work if there is nothing to be clustered. And keep in mind that the input corpus should contain at least 1000 documents to get meaningful results. Refer to this thread: https://github.com/MaartenGr/BERTopic/issues/59#issuecomment-775718747.

Parameters
docs (*List[str]*) – The input corpus.

Returns
Topics of the input docs.

Return type
List[int]

get_topic_table() → pandas.DataFrame

Get the table of topic information and return it as a pandas dataframe.

Returns
The topic table.

Return type
pd.DataFrame

get_doc_embeds() → numpy.ndarray

Get the embeddings of the docs.

Returns
Embeddings of the docs, in size of (n_doc, n_components).

Return type
np.ndarray

orangearg.argument.miner.miner

Argument mining module

Module Contents

Functions

select_by_topic(→ pandas.DataFrame)	Select arguments mentioning the given topic.
get_edges(→ List[Tuple[int]])	Get edges from argument dataframe.
get_edge_weights(→ List[float])	Get edge weights.
get_edge_table(→ pandas.DataFrame)	Get the edge dataframe.
get_node_labels(→ List[str])	Get labels of arguments given the attacking network.
get_node_table(→ pandas.DataFrame)	Get the node dataframe.

orangearg.argument.miner.miner.**select_by_topic**(*data: pandas.DataFrame*, *topic: int*) → pandas.DataFrame

Select arguments mentioning the given topic.

Parameters

- **data** (*pd.DataFrame*) – The argument dataframe that must contain the 'topics' column.
- **topic** (*int*) – The given topic to select.

Raises
ValueError – if the 'topics' value of an argument is stored as something else other than a tuple (e.g. a list).

Returns
Part of the original argument dataframe that only contains arguments mentioning the given topic.

Return type
pd.DataFrame

orangearg.argument.miner.miner.**get_edges**(*data: pandas.DataFrame*) → List[Tuple[int]]

Get edges from argument dataframe.

Edges (attacks) only exist if the two arguments have different overall scores. Edges are tuple of source and target, which are indices of the corresponding argument in the input dataframe.

Parameters
data (*pd.DataFrame*) – The argument dataframe that must have the 'score' column.

Returns
The edge list.

Return type
List[Tuple[int]]

orangearg.argument.miner.miner.**get_edge_weights**(*data: pandas.DataFrame*, *edges: List[Tuple[int]]*) → List[float]

Get edge weights.

Edge weights are computed as the difference between the coherence of the source and that of the target.

Parameters

- **data** (*pd.DataFrame*) – The argument dataframe that must have the 'coherence' column.
- **edges** (*List[Tuple[int]]*) – The edge list.

Returns
The list of edge weights.

Return type
List[float]

orangearg.argument.miner.miner.**get_edge_table**(*edges: List[Tuple[int]]*, *weights: List[float]*) → pandas.DataFrame

Get the edge dataframe.

There will be three columns in the output dataframe, which are 'source', 'target', and 'weight'. Together, they describe weighted directed edges from source to target argument. Note that there will be no negative weights in the output dataframe, instead, all values will be replace with their absolute values. For edges with negative weights, we swap their source and target.

Parameters

- **edges** (*List[Tuple[int]]*) – The edge list, which are tuples of source and target argument ids.
- **weights** (*List[float]*) – The list of edge weights.

Raises
ValueError – if size of the input lists doesn't match.

Returns
The result edge dataframe.

Return type
pd.DataFrame

orangearg.argument.miner.miner.**get_node_labels**(*indices: List[int]*, *sources: List[int]*, *targets: List[int]*) → List[str]

Get labels of arguments given the attacking network.

Arguments are separated into two classes, 'supportive' and 'defeated', which generally means reliable and unreliable. The rule of detecting the labels is as follows: if an argument is attacked by another argument who is not attacked by any argument, then this argument is labeled as 'defeated'; otherwise, it's labeled as 'supportive'. That means, if an argument appears in *targets*, where its corresponding source doesn't, this argument will be labeled as 'defeated', and otherwise 'supportive'.

Parameters

- **indices** (*List[int]*) – The node index list
- **sources** (*List[int]*) – The source list of the attacking network.
- **targets** (*List[int]*) – The target list of the attacking network.

Returns
The label list.

Return type
List[str]

orangearg.argument.miner.miner.**get_node_table**(*arg_ids: List[int]*, *arguments: List[str]*, *scores: List[int]*, *labels: List[str]*) → pandas.DataFrame

Get the node dataframe.

The node dataframe will contain 4 columns, that are 'argument_id', 'argument', 'score', and 'label'.

Parameters

- **arg_ids** (*List[int]*) – The argument id list.
- **arguments** (*List[str]*) – The argument text list.
- **scores** (*List[int]*) – The list of argument overall score.
- **labels** (*List[str]*) – The argument label list.

Returns
The result node dataframe.

Return type
pd.DataFrame

orangearg.argument.miner.processor

Argument processor module.

Module Contents

Functions

_match_list_size(*args)	With an arbitrary number of lists as input, check if they are in the same size.
_aggregate_list_by_another(→ Dict)	Aggregate a list according to elements of another list.
get_argument_topics(→ List[Tuple[int]])	Get argument topics.
get_argument_sentiment(→ List[float])	Get argument sentiment score.
get_argument_coherence(→ List[float])	Get argument coherence.
update_argument_table(→ pandas.DataFrame)	Return a copy of argument dataframe, with new columns of argument topics, sentiments, and coherences.

orangearg.argument.miner.processor.**_match_list_size**(**args: List*)

With an arbitrary number of lists as input, check if they are in the same size.

orangearg.argument.miner.processor.**_aggregate_list_by_another**(*keys: List*, *values: List*) → Dict

Aggregate a list according to elements of another list.

Parameters

- **keys** (*List*) – The group keys.
- **values** (*List*) – The list to be aggregated.

Returns
The aggregation result.

Return type
Dict

orangearg.argument.miner.processor.**get_argument_topics**(*arg_ids: List[int]*, *topics: List[int]*) → List[Tuple[int]]

Get argument topics.

The topics of an argument is a combination of the topics of all chunks that belong to this argument. Duplications are not removed, and the reason behind is that duplications can be treated as a sign of topic importance. Also, even though two chunks can belong to the same topic, they could still have different ranks within an argument.

Parameters

- **arg_ids** (*List[int]*) – the argument ids of chunks.
- **topics** (*List[int]*) – the topic indices of chunks.

Returns
list of argument topics, which is also a list containing topic indices of chunks belonging to this argument.

Return type
List[list[int]]

orangearg.argument.miner.processor.**get_argument_sentiment**(*arg_ids: List[int]*, *ranks: List[float]*, *p_scores: List[float]*, *min_sent: int = -1*, *max_sent: int = 1*) → List[float]

Get argument sentiment score.

The sentiment score of an argument is calculated as a weighted sum of sentiment scores of chunks belonging to this argument, where weights are ranks of the chunks. The result score is then normalized into range [0, 1].

Parameters

- **arg_ids** (*List[int]*) – the argument ids of chunks.
- **ranks** (*List[float]*) – the pagerank of chunks within arguments.
- **p_scores** (*List[float]*) – the sentiment polarity scores of chunks.
- **min_sent** (*int*) – minimun of argument sentiment before normalization. Defaults to -1.
- **max_sent** (*int*) – maximum of argument sentiment before normalization. Defaults to 1.

Returns
List of argument sentiment scores, which are floats in range [0, 1].

Return type
List[float]

orangearg.argument.miner.processor.**get_argument_coherence**(*scores: List[int]*, *sentiments: List[float]*, *min_score: int = 1*, *max_score: int = 5*, *variance: float = 0.2*) → List[float]

Get argument coherence.

Coherence is computed as inversed difference between sentiments and overall scores. Overall scores are first normalized into the same range as argument sentiments, which is [0, 1]. Then their differences are computed and applied a Gaussian kernal to invert and scale the differences to [0, 1].

Parameters

- **scores** (*List[int]*) – List of argument overall scores.
- **sentiments** (*List[float]*) – List of argument sentiment scores.
- **min_score** (*int, optional*) – Lower bound of scores. Defaults to 1.
- **max_score** (*int, optional*) – Upper bound of scores. Defaults to 5.
- **variance** (*float*) – variance of the Gaussian kernal.

Returns
List of argument coherence scores, in range of (0, 1]

Return type
List[float]

orangearg.argument.miner.processor.**update_argument_table**(*df_arguments: pandas.DataFrame*, *topics: List[List[int]]*, *sentiments: List[float]*, *coherences: List[float]*) → pandas.DataFrame

Return a copy of argument dataframe, with new columns of argument topics, sentiments, and coherences.

Parameters

- **df_arguments** (*pd.DataFrame*) – argument dataframe.
- **topics** (*List[List[int]]*) – list of argument topics
- **sentiments** (*List[float]*) – list of argument sentiment scores
- **coherences** (*List[float]*) – list of argument coherence scores

Returns
description

Return type
pd.DataFrame

orangearg.argument.miner.reader

Argument filre reader module

This module implements functions for reading input data files in different formats. So far, we only have the support to JSON file. But we forsee the need of supporting other formats, and all future functions in this scope should be in this module.

Module Contents

Functions

read_json_file(→ pandas.DataFrame)	Read a local JSON file and return its content as a pandas dataframe.

orangearg.argument.miner.reader.**read_json_file**(*fpath: str*) → pandas.DataFrame

Read a local JSON file and return its content as a pandas dataframe.

This function will automatically handle the case that a JSON file contains multiple JSON objects. It will also normalize semi-structured JSON strings.

Parameters
fpath (*str*) – The file path

Returns
The pandas dataframe object that contains content of the JSON file read from the given path.

Return type
pd.DataFrame

orangearg.argument.miner.utilities

Collection of helper functions.

Module Contents

Functions

check_columns(expected_cols, data)	Check if a list of given columns exist in a given Pandas dataframe.

orangearg.argument.miner.utilities.**check_columns**(*expected_cols: List[str]*, *data: pandas.DataFrame*)

Check if a list of given columns exist in a given Pandas dataframe.

Parameters

- **expected_cols** (*List[str]*) – list of columns to check
- **df** (*pd.DataFrame*) – pandas dataframe to check

Raises

ValueError – if any of the expected columns are missing.

PYTHON MODULE INDEX

O

INDEX

Symbols

C

F

G

I

L

M

O

R

S

T

U