NaCaGraph: Visualizing Event Graph for Natural Catastrophes from News Articles

Li Lin, Chong Shen, Xiru Tong

Institute for Natural Language Processing
University of Stuttgart
{*li.lin, chong.shen, xiru.tong*}*@ims.uni-stuttgart.de*

Introduction

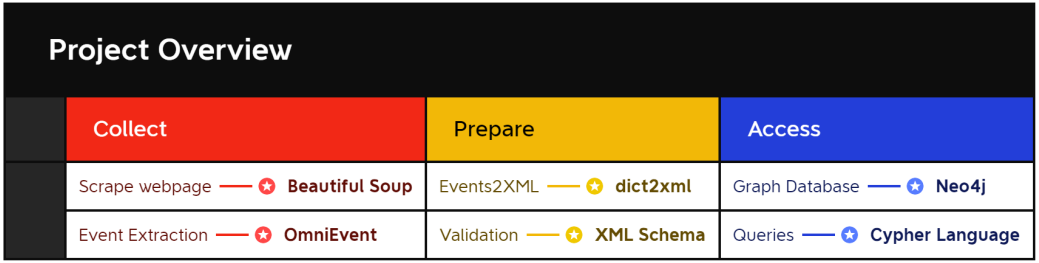
Motivation

- Natural catastrophes often occur unexpectedly, causing severe damages.
  $\Rightarrow$ Necessary to collect, query and analyze information.
- *Events* are fundamental components for describing the situation of natural catastrophes.
- News articles are an essential information source of natural catastrophe events, but have some limitations.

Introduction
Research Focus

- How do the components of events distribute in different articles?
- What are the potential inter-article and inter-topic relationships w.r.t. components of an event?

Introduction
Project Overview



Collect:

Web Scraping and Data Preprocessing

Collect: Web Scraping

Step 1: Search with the 7 category (i.e. topic) names in Wikinews [1] and copy the URL of the resulting page into the input file, one URL per line.

- *climate change*,
- *drought*,
- *earthquake*,
- *forest fire*,
- *greenhouse*,
- *high temperature*,
- *wildfire*.

[1] https://en.wikinews.org

Collect: Web Scraping

Step 2:
For each URL:

- Scrape the HTML content from the resulting page using the URL.
  - page = requests.get(url)
  - 100 searching results per category
- Parse the lengthy HTML content using **Beautiful Soup** [2], making the results cleaner and more accessible.
  - Create a Beautiful Soup object that takes page.content as input and parse it with html.parser.
  - soup = BeautifulSoup(page.content, "html.parser")
  - As a result, the page content is parsed to a tree structure.

[2] https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Collect: Web Scraping

Step 2 (cont.):
For each URL:

- Manually identify the elements in the parsed HTML corresponding to the target region in the web page.
  - Get the entire region of the searching results using the id ”mw-content-text”.
    - results = soup.find(id=”mw-content-text”) Go to figure
  - Find all single results that have the tag ”li” (list item) and an attribute ”class” with the value ”mw-search-result mw-search-result-ns-0”.
    - articles = results.find_all(”li”, class_=”mw-search-result mw-search-result-ns-0”) Go to figure

Collect: Web Scraping

Step 2 (cont.): ▸ Back to slides

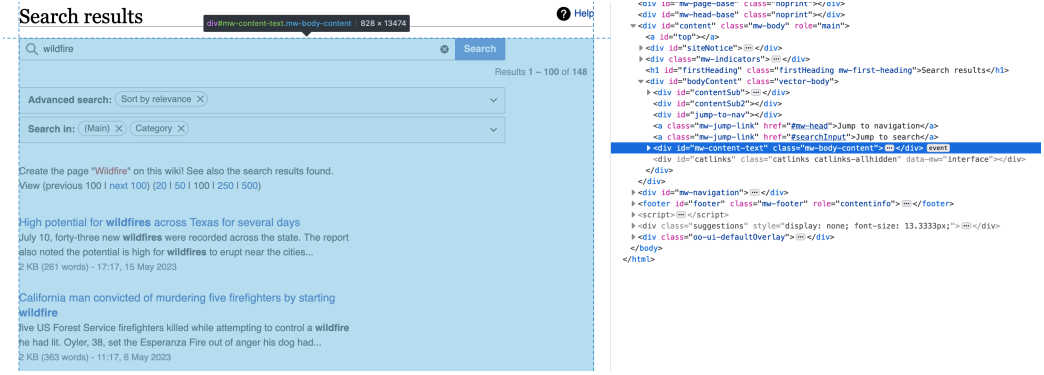


Figure: results = soup.find(id="mw-content-text")

Collect: Web Scraping

Step 2 (cont.):

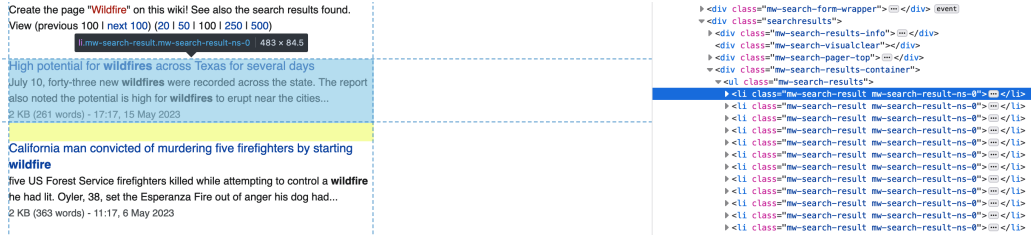
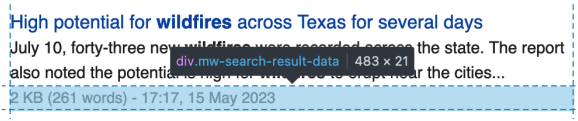


Figure: results = soup.find(id="mw-search-result mw-search-result-ns-0")

Collect: Web Scraping

Step 2 (cont.):
For each URL:

- For each article, find its metadata using the tag ”div” (”division”) and an attribute ”class” with the value ”mw-search-result-data”.
  - metadata = article.find(”div”, class_=”mw-search-result-data”)



- Extract the text span describing the number of words from the metadata using regular expression and convert it to an integer.
  - num_words = re.findall(rule, metadata.text)
  - article_words_num = int(””.join(num_words[0].split(’,’)))

Collect: Web Scraping

Step 2 (cont.):
For each URL:

- If the current article is not empty (number of words > 0), find the heading element of the article.
  - heading = article.find(”div”, class_=”mw-search-result-heading”)
- Build a mapping ”aid2metadata” from the articleID (aid) to the extracted metadata.

Collect: Web Scraping

Step 3: Get the content of actual article pages.
For each URL:

- Get the publish date of the article using the tag ”strong” and an attribute ”class” with the value ”published”.
  - publish_date = soup.find(”strong”, class_=”published”) Go to figure
- If the publish date exists, then get the article content using ”div” and class with value ”mw-parser-output”.
  - article_content = soup.find(”div”, class_=”mw-parser-output”)
  - texts_content = article_content.find_all(’p’, recursive=False) Go to figure
- Results are two mappings: ”aid2article_text”, ”aid2publish_date”. Add the data to the arcitle text.
- Write the final result ”aid2article_with_date” to JSON files.

Collect: Web Scraping

Step 3: Get the content of actual article pages. Back to slides

Figure: publish_date = soup.find(”strong”, class_=”published”)

Collect: Web Scraping

Step 3: Get the content of actual article pages.

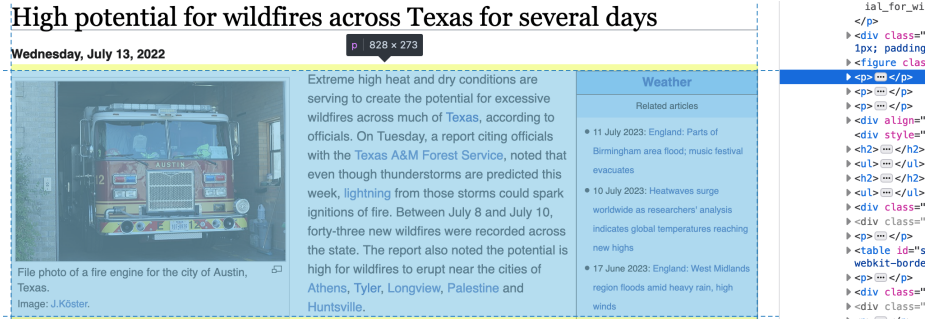


Figure: texts_content = article_content.find_all('p', recursive=False)

Collect: Data Preprocessing

Goal: To facilitate the next step (see **Extension**).

1. Add a stop '.' at the end of the news headline of each article.
2. Concatenate the news headline to the main text, such that each two sentences are split by a whitespace.
3. Build a mapping from articelID (*aid*) to the processed text.
4. Write the results of each article of each category to a **.txt** file under the input directory of the next step.

Extension:

Event Extraction from WikiNews

Event Extraction (EE): Definition

- Intuitively, an *event* reflects the emergence of a situation driven by an action, e.g., *Who did what to whom? Where, when, and how?*.
- In this project, we use the definition of (Li et al. 2022):
  1. **Event trigger**
  2. **Event type**
  3. **Event argument(s)**
  4. **Argument role(s)**
- Event Extractor: **OmniEvent**.[3]

[3] https://github.com/THU-KEG/OmniEvent

Event Extraction: Example

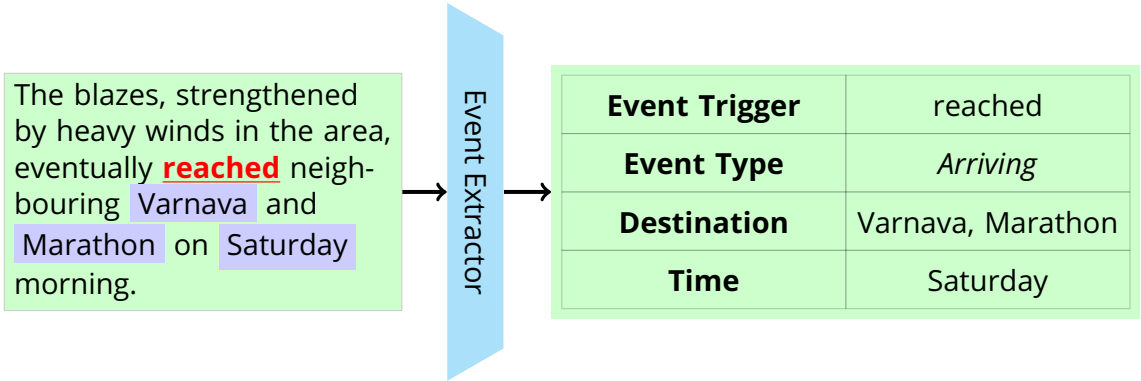
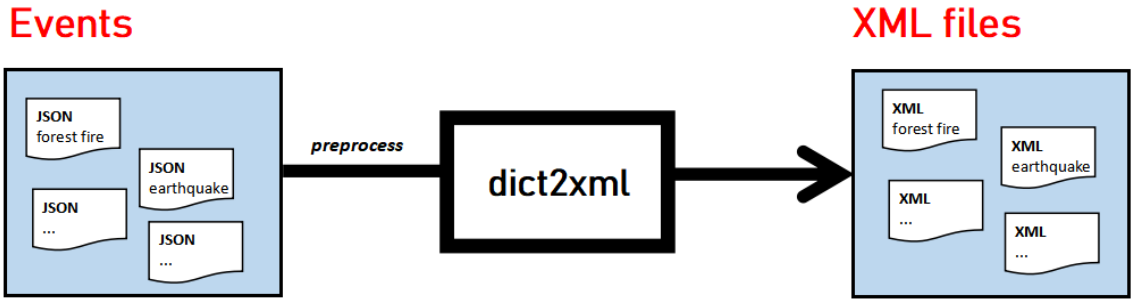


Figure: The structure of an event extracted from an input text.

Prepare:

- Events2XML
- XML Schema (only for validation)

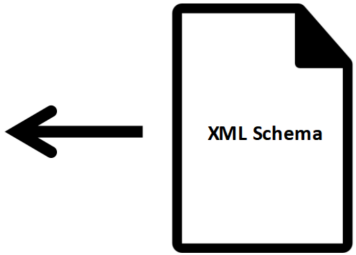
Prepare


Events
JSON
forest fire
JSON
earthquake
JSON
...
JSON
...
preprocess
dict2xml
XML files
XML
forest fire
XML
earthquake
XML
...
XML
...

Prepare

```xml
<?xml version="1.0" encoding="UTF-8"?>
<category topic="forest_fire" cid="1"
          xmlns="http://www.w3schools.com/NaturalCatastrophe"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.w3schools.com/NaturalCatastrophe xml_schema.xsd">
  <article aid="1-1">
    <title>Forest fires burn on in Greece; death toll nears 50</title>
    <date>25-August-2007</date>
    <event>
      <trigger>burn</trigger>
      <type>bodilyharm</type>
      <argument>
        <mention>Greece</mention>
        <role>place</role>
      </argument>
    </event>
    <event>
    ...
    </event>
  </article>
  <article>
  ...
  </article>
</category>
```

```
xmllint --schema xml_schema.xsd forest_fire.xml
```



Access:

Event Graph Visualization and Querying

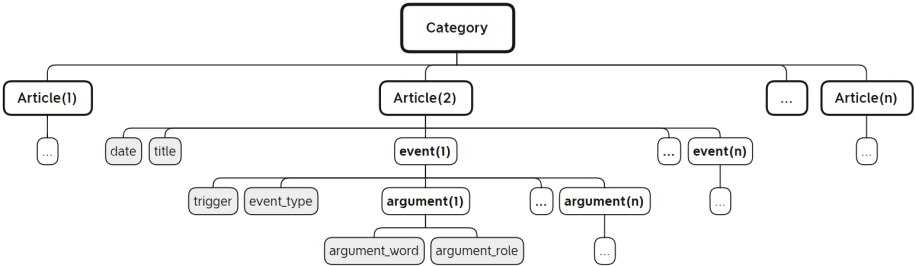
Access

1. Create Neo4j database
   1. Convert XML file to Neo4j database
   2. Graph structure design
2. Query Examples

Access: Create Neo4j database

Convert XML file to Neo4j database:

- Our XML file can now be understood as the following tree structure.



- The point of data structure is:
- 1). Category:Articles; Article:Events; Event:Arguments – 1 : many
- 2). Null values: Some articles are not extracted for any events; some events do not have an argument.

Access: Create Neo4j database

Convert XML file to Neo4j database:

- Loading Tool: apoc.load.xml [4]
- Steps(read the xml tree layer by layer):
  1. Use 'WITH' to get the nodes of the Category, read the attribute and use "MERGE" to set the node "Topic".

```
CALL apoc.load.xml("file:C:/Users/thisi/wildfire.xml")
YIELD value
WITH value, value.topic AS topic_text
MERGE (t: Topic {text: topic_text, type:"topic"} )
```

  2. Use "WITH" and "UNWIND" to get the Category's children nodes Article one by one, read the article's attributes. Use "MERGE" to set the nodes "Article", "Year" and relationships.

```
WITH value, t
UNWIND value._children AS article
WITH t, article.aid AS articleId,
    [item in article._children WHERE item._type = "title"][0] AS title,
    [item in article._children WHERE item._type = "date"][0] AS date,
    [item in article._children WHERE item._type = "event"] AS events
MERGE ( a:Article {text:title._text, type:"article"})
MERGE ( dat:Year {text: right(date._text,4), type:'year'})
MERGE (t)-[:published_time {text: date._text}]→(a)
MERGE (dat)-[:published_article ]→(a)
```

  3. Repeat the above steps...Iterate through to the last layer.

[4] https://neo4j.com/labs/apoc/4.4/import/xml/

Access: Create Neo4j database

Graph structure design:

- Trouble with the design of relationships:
- How to assign node-to-node relationships? How to deal with the nodes of the same text content?
- For example:
  1. (article)-[event-type]-(trigger) vs.
     (article)-[relation]-(event-type)-[relation]-(trigger) ?
  2. Under the same article, (:Trigger) vs. (:Trigger {article-id}) ?

Access: Create Neo4j database

Graph structure design: Structure overview

- Nodes * 9

*(29,350)

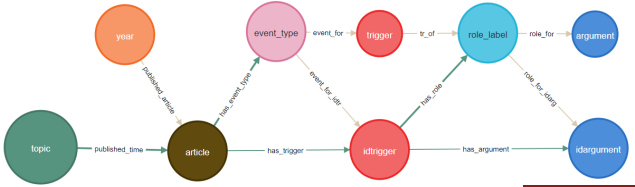
Nodes	Property keys		
Topic	type	text	
Year	type	text	
Article	type	text	
idTrigger	type	text	id
Trigger	type	text	
idArgument	type	text	id
Argument	type	text	
Event_type	type	text	
Role_label	type	text	

- Relationship types * 11

*(91,369)

Relationships	Property keys	
published_time	type	text
published_article	type	
has_trigger	type	text
has_argument	type	text
has_event_type	type	id
tr_of	type	text
has_role	type	id
event_for	type	
event_for_idtr	type	
role_for	type	
role_for_idarg	type	

- Structure



Access: Query Examples

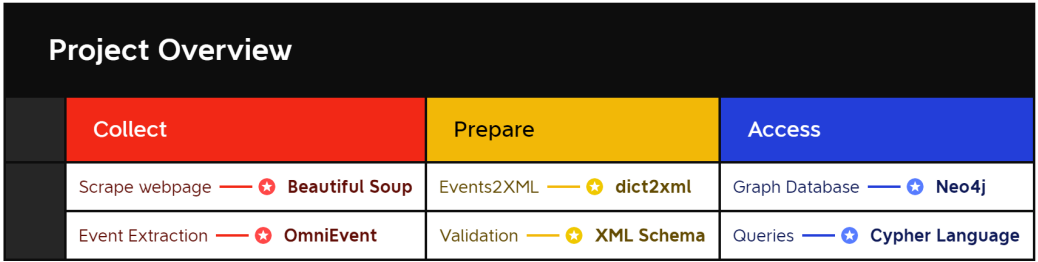
See Appendix: Go to examples

- Query Example 1
- Query Example 2
- Query Example 3

Difficulties

- The dict2xml library does not inherently support adding attributes or referencing a schema.
- Finding a data source that meets the requirements of the research questions.
- Limited performance of the event extraction system on our data.

Conclusion



Questions?

References I

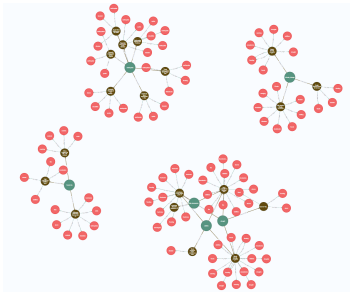
 Li, Qian et al. (2022). “A Survey on Deep Learning Event Extraction: Approaches and Applications”. In: *IEEE Transactions on Neural Networks and Learning Systems*.

Appendix - Access: Query Example 1

- **Example 1:**Query the distribution of triggers, articles and topics with event_type is ”catastrophe” after 2020.
- Command:

```
MATCH p=(:Topic )-[r1:published_time]→(arti:Article)
-[r2:has_trigger {text:"type:catastrophe"}]→(n:idTrigger)
where toInteger(right(r1.text,4)) >2020 return p
```

- Query result



Appendix - Access: Query Example 2 - Query about trigger ”fire”

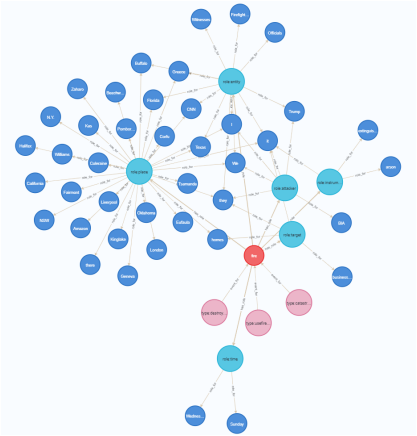
Example 2: Take the query of trigger ”fire” as an example.
query 1: Visual map of event_types, role_labels and arguments of trigger ”fire”.

Command:

```
1 match (:idTrigger{text:"fire"})⟶(idarg:idArgument)
2 match p=(:Event_type)⟶(:Trigger{text:"fire"})⟶
3 (:Role_label)⟶(:Argument{text:idarg.text})
4 return p
```

result: 3 event types, 6 role labels

Query result:



Appendix - Access: Query Example 2 - Query about trigger ”fire”

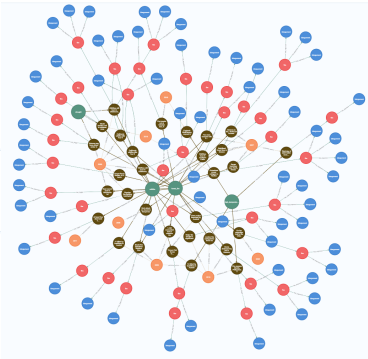
query 2: Visual map of the distribution of trigger ”fire” by topic, article and argument and published year.

Command:

```
match  p=(t:Topic)⟶(arti:Article)-[et:has_trigger]→
(idtr:idTrigger{text:"fire"})-[rl:has_argument]→(:idArgument )
match (y:Year)⟶(arti)
return p,y
```

result: 4 topics: wildfire, forest fire, drought, high temperature

Query result:



Appendix - Access: Query Example 2 - Query about trigger ”fire”

query 3: Count the frequency of trigger ”fire” under each topic in different year after 2003.

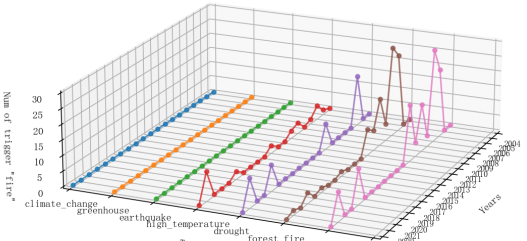
Command and results:

return: topics, years, frequency numbers.

```
match p=(topic:Topic)-[r1:published_time]→(arti:Article)⟶(idtr:idTrigger{text:"fire"})
match (topic)-[r1]→(arti)⟶(:Event_type)⟶(tr:Trigger{text:idtr.text})
match (year:Year)⟶(arti)
where toInteger(right(r1.text,4))>2003
with topic,year , count(tr) as num_tr
return topic.text as Topic,  toInteger(year.text) as Year, num_tr as num_trigger
order by Topic,Year desc
```

Topic	Year	num_trigger
"drought"	2022	9
"drought"	2019	8
"drought"	2011	8
"drought"	2006	16
"forest_fire"	2022	1
"forest_fire"	2020	3
"forest_fire"	2018	1

After python processing:



Appendix - Access: Query Example 3 - Similarity between topics

- **Example 3:** Statistical Examples. Count the first ten event types and triggers per topic to see how topics are related.
- Command and results: (count first ten number of event types)
- return: topics, ten event_tpyes, ten frequency numbers

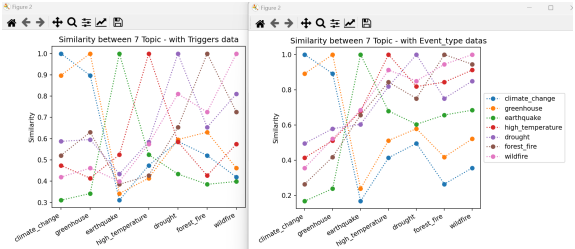
```
match p=(topic:Topic)-[r1:published_time]→(arti:Article)⟶(et:Event_type)
where toInteger(right(r1.text,4))>2000
with topic,et,count(et) as num_et
order by num_et desc
with topic, collect(et.text)[0..10] as event_types, collect( num_et )[0..10] as num_event_types
return topic.text as topic, event_types, num_event_types
```

Table | Text | Code

topic	event_types	num_event_types
"climate_change"	["type:statement", "type:causechangeofpositiononascale", "type:change", "type:communication", "type:agreeorrefus|etoact", "type:causetobeincluded", "type:influence", "type:socialevent", "type:know", "type:processend"]	[98, 92, 89, 69, 52, 47, 47, 46, 46, 44]
"greenhouse"	["type:statement", "type:causechangeofpositiononascale", "type:communication", "type:change", "type:agreeorrefus|etoact", "type:processstart", "type:socialevent", "type:know", "type:causation", "type:causetobeincluded"]	[88, 83, 58, 57, 44, 43, 43, 41, 40, 39]
"wildfire"	["type:statement", "type:catastrophe", "type:processstart", "type:bodilyharm", "type:motion", "type:reporting", |"type:causation", "type:destroying", "type:causechangeofpositiononascale", "type:know"]	[88, 80, 62, 59, 59, 58, 58, 50, 43, 43]
"drought"	["type:statement", "type:catastrophe", "type:causechangeofpositiononascale", "type:reporting", "type:motion", "t|ype:causation", "type:know", "type:influence", "type:processstart", "type:assistance"]	[83, 75, 60, 47, 44, 43, 42, 41, 38, 37]
"earthquake"	["type:catastrophe", "type:attack", "type:statement", "type:damaging", "type:reporting", "type:bodilyharm", "typ|e:causation", "type:motion", "type:perceptionactive", "type:comingtobe"]	[80, 66, 63, 62, 61, 48, 48, 39, 36, 34]
"high_temperature"	["type:statement", "type:catastrophe", "type:causation", "type:causechangeofpositiononascale", "type:reporting", | "type:bodilyharm", "type:motion", "type:creating", "type:know", "type:destroying"]	[78, 63, 53, 50, 49, 46, 43, 40, 40, 37]
"forest_fire"	["type:statement", "type:catastrophe", "type:destroying", "type:processstart", "type:motion", "type:bodilyharm", | "type:causation", "type:reporting", "type:know", "type:arriving"]	[77, 68, 57, 51, 51, 51, 51, 43, 43, 39]

Appendix - Access: Query Example 3 - Similarity between topics

- **Example 3:** Statistical Examples. Count the first ten number of event types and triggers per topic to see how topics are related.
- After python to calculate and draw the similarity scores:



Back to slides

result:

high similarity: climate_change & greenhouse, forest_fire & wildfire;

high temperature: has quite a high similarity with forest fire, wildfire and drought;

earthquake: minimal similarity with other six topics.