Section topics

Release 0.11.0

Marco Fossati, Matthias Mullie, Muniza Aslam, Xabriel Collazo

Nov 10, 2023

CONTENTS

Gather Wikidata items from Wikipedia blue links.

CHAPTER

ONE

RUN IT!

You need access to a Wikimedia Foundation's analytics client, AKA a *stat box*. Then:

```
me@my_box:~$ ssh stat1008.eqiad.wmnet  # Or pick another one
me@stat1008:~$ export http_proxy=http://webproxy.eqiad.wmnet:8080
me@stat1008:~$ export https_proxy=http://webproxy.eqiad.wmnet:8080
me@stat1008:~$ git clone https://gitlab.wikimedia.org/repos/structured-data/section-
↪topics.git st
me@stat1008:~$ cd st
me@stat1008:~/st$ conda-analytics-clone MY_ENV
me@stat1008:~/st$ source conda-analytics-activate MY_ENV
(MY_ENV) me@stat1008:~/st$ conda env update -n MY_ENV -f conda-environment.yaml
(MY_ENV) me@stat1008:~/st$ python section_topics/pipeline.py MY_WEEKLY_SNAPSHOT
```

1.1 Get –help

```
(MY_ENV) me@stat1008:~/st$ python section_topics/pipeline.py --help
usage: pipeline.py [-h] [-w /hdfs_path/to/dir/] [-i /path/to/file.txt]
                [-p hdfs_path/to/parquet] [-s /path/to/file.json] [-l N]
                [-t hdfs_path/to/parquet] [-q /path/to/file1.txt ...] [--handle-media]
                [-m /path/to/file.txt] [--keep-lists-and-tables]
                YYYY-MM-DD

Gather section topics from Wikitext

positional arguments:
YYYY-MM-DD            snapshot date

options:
-h, --help            show this help message and exit
-w /hdfs_path/to/dir/, --work-dir /hdfs_path/to/dir/
                        Absolute HDFS path to the working directory. Default:
                        "section_topics" in the current user home
-i /path/to/file.txt, --input-wikis /path/to/file.txt
                        plain text file of wikis to process, one per line. Default: all
                        Wikipedias, see "data/wikipedias.txt"
-p hdfs_path/to/parquet, --page-filter hdfs_path/to/parquet
                        HDFS path to parquet of (wiki, page revision ID) rows to exclude,
                        as output by "scripts/check_bad_parsing.py". Must be relative to
```

(continues on next page)

(continued from previous page)

```
                        the working directory. Default: badly parsed ptwiki articles, see
                        "2022-10_ptwiki_bad" in the working directory
-s /path/to/file.json, --section-title-filter /path/to/file.json
                        JSON file of `{ wiki: [list of section titles to exclude] }`.
                        Default: see "data/section_titles_denylist.json"
-l N, --length-filter N
                        exclude sections whose content length is less than the given
                        number of characters. Default: 500
-t hdfs_path/to/parquet, --table-filter hdfs_path/to/parquet
                        HDFS path to parquet with a dataframe to exclude, as output by
                        "scripts/detect_html_tables.py". Must be relative to the working
                        directory. The dataframe must include ('wiki_db', 'page_id',
                        'section_title') columns. Default: ar, bn, cs, es, id, pt, ru
                        sections with tables, see "20230301_target_wikis_tables" in the
                        working directory
-q /path/to/file1.txt ..., --qid-filter /path/to/file1.txt ...
                        plain text file(s) of Wikidata IDs to exclude, one per line.
                        Default: see "data/qids_for_all_points_in_time.txt" and
                        "data/qids_for_media_outlets.txt"
--handle-media        separate media links and dump them to "media_links". WARNING: the
                        pipeline execution time will increase to roughly 20 minutes
-m /path/to/file.txt, --media-prefixes /path/to/file.txt
                        plain text file with media prefixes, one per line. Default: all
                        Wikipedia ones, see "data/media_prefixes.txt". Ignored if "--
                        handle-media" is not passed
--keep-lists-and-tables
                        don't skip sections with at least one standard wikitext list or
                        table
```

CHAPTER

TWO

GET YOUR HANDS DIRTY

Install the development environment:

```
me@stat1008:~/st$ conda-analytics-clone MY_DEV_ENV
me@stat1008:~/st$ source conda-analytics-activate MY_DEV_ENV
(MY_DEV_ENV) me@stat1008:~/st$ conda env update -n MY_DEV_ENV -f dev-conda-environment.
↪yaml
```

2.1 Test

```
(MY_DEV_ENV) me@stat1008:~/st$ python -m pytest tests/
```

2.2 Lint

```
(MY_DEV_ENV) me@stat1008:~/st$ pre-commit install
```

At every `git commit`, pre-commit will run the checks and autofix or tell you what to fix.

2.3 Docs

```
(MY_DEV_ENV) me@stat1008:~/st$ sphinx-build docs/ docs/_build/
```

CHAPTER

THREE

TRIGGER AN AIRFLOW TEST RUN

Follow this walkthrough to simulate a production execution of the pipeline in your stat box. Inspired by this snippet.

3.1 Build your artifact

1. Pick a branch you want to test from the drop-down menu
2. Click on the pipeline status button, it should be a green tick
3. Click on the *play* button next to `publish_conda_env`, wait until done
4. On the left sidebar, go to **Packages and registries > Package Registry**
5. Click on the first item in the list, then copy the Asset URL. It should be something like `https://gitlab.wikimedia.org/repos/structured-data/section-topics/-/package_files/1321/download`

3.2 Get your artifact ready

```
me@stat1008:~$ mkdir artifacts
me@stat1008:~$ cd artifacts
me@stat1008:~$ wget -O MY_ARTIFACT MY_COPIED_ASSET_URL
me@stat1008:~$ hdfs dfs -mkdir artifacts
me@stat1008:~$ hdfs dfs -copyFromLocal MY_ARTIFACT artifacts
me@stat1008:~$ hdfs dfs -chmod -R o+rx artifacts
```

3.3 Spin up an Airflow instance

On your stat box:

```
me@stat1008:~$ git clone https://gitlab.wikimedia.org/repos/data-engineering/airflow-
↪dags.git
me@stat1008:~$ cd airflow-dags
me@stat1008:~$ sudo -u analytics-privatedata rm -fr /tmp/air/MY_AIRFLOW_HOME  # If you've␣
↪previously run the next command
me@stat1008:~$ sudo -u analytics-privatedata ./run_dev_instance.sh -m /tmp/MY_AIRFLOW_
↪HOME -p MY_PORT platform_eng
```

On your local box:

```
me@my_box:~$ ssh -t -N stat1008.eqiad.wmnet -L MY_PORT:stat1008.eqiad.wmnet:MY_PORT
```

3.4 Trigger the DAG run

1. Go to `http://localhost:MY_PORT/` on your browser
2. On the top bar, go to **Admin > Variables**
3. Click on the middle button (*Edit record*) next to the `platform_eng/dags/section_topics_dag.py` Key
4. Update `{ "conda_env" : "hdfs://analytics-hadoop/user/ME/artifacts/MY_ARTIFACT" }`
5. Add any other relevant DAG properties
6. Click on the *Save* button
7. On the top bar, go to **DAGs** and click on the `section_topics` slider. This should trigger an automatic DAG run
8. Click on `section_topics`

You're all set!

CHAPTER

FOUR

RELEASE

1. On the left sidebar, go to **CI/CD > Pipelines**
2. Click on the *play* button, select `trigger_release`
3. If the job went fine, you'll find a new artifact in the Package Registry

We follow Data Engineering's workflow_utils: - the `main` branch is on a `.dev` release - releases are made by removing the `.dev` suffix and committing a tag

CHAPTER

FIVE

DEPLOY

1. On the left sidebar, go to **CI/CD > Pipelines**
2. Click on the *play* button and select `bump_on_airflow_dags`. This will create a merge request at airflow-dags
3. Double-check it and merge
4. Deploy the DAGs:

```
me@my_box:~$ ssh deployment.eqiad.wmnet
me@deploy1002:~$ cd /srv/deployment/airflow-dags/platform_eng/
me@deploy1002:~$ git pull
me@deploy1002:~$ scap deploy
```

See the docs for more details.

CHAPTER

SIX

API DOCUMENTATION

6.1 The data pipeline

The section topics data pipeline is a sequence of `pyspark.sql.DataFrame` extraction and transformation functions.

Inputs come from Wikimedia Foundation's Analytics Data Lake:

- Wikipedias wikitext (all Wikipedias by default as per wikipedias.txt)
- Wikidata item page links

High-level steps:

- gather wikitext of sections at a given hierarchy level via the MediaWiki parser from hell. Default: *section_topics.pipeline.SECTION_LEVEL*, lead section included
- optionally filter out sections that don't convey relevant content, typically lists and tables
- extract Wikidata QIDs from wikilinks: the so-called **section topics**
- optionally filter out noisy topics, typically dates and numbers
- compute the relevance score

Output row example:

snap-shot	wiki_(	page_name	revi-sion_id	page_	page_	page_t	sec-tion_inde	sec-tion_title	topic_(	topic_ti	topic_score
2023-01-16	en-wiki	0	1127523	Q3672	841	Attila	5	Solitary kingship	Q3623	Arnegis clus	1.13

More documentation lives in MediaWiki.

Functions are ordered by their execution in the pipeline.

`section_topics.pipeline.SECTION_LEVEL = 2`

Section hierarchy level to be extracted. Level 1 is just for page titles, actual sections start from level 2.

`section_topics.pipeline.SECTION_ZERO_TITLE = '### zero ###'`

Reserved title for the lead section, AKA section zero.

`section_topics.pipeline.STRIP_CHARS = '!"#$%&\' *+,-./:;<=>?@[\\]^_`{|}~'`

ASCII punctuation characters to be stripped from section headings. Include the ASCII white space, don't strip round brackets.

section_topics.pipeline.**SUBSTITUTE_PATTERN = '[\\s_]'**

All kinds of white space to be substituted for the ASCII one; underscores turn into spaces as well.

section_topics.pipeline.**get_monthly_snapshot**(*weekly*)

Get the most recent monthly snapshot given a weekly one.

A snapshot date is the **beginning** of the snapshot interval. For instance:

- `2022-05-16` covers until `2022-05-22` (at 23:59:59). May is not over, so the May monthly snapshot is not available yet. Hence return end of **April**, i.e., `2022-04`
- `2022-05-30` covers until `2022-06-05`. May is over, so the May monthly snapshot is available. Hence return end of **May**, i.e., `2022-05`

Parameters
weekly (str) – a `YYYY-MM-DD` date

Raises
ValueError – if the passed date has an invalid format

Return type
str

Returns
the relevant monthly snapshot

section_topics.pipeline.**load_pages**(*spark*, *monthly_snapshot*, *wikis*, *namespace*)

Load wiki pages with their wikitext through the *section_topics.queries.PAGES* Data Lake query.

Parameters

- **spark** (SparkSession) – an active Spark session
- **monthly_snapshot** (str) – a `YYYY-MM` date
- **wikis** (str) – a string of comma-separated wikis to process
- **namespace** (int) – a page namespace number, e.g., `0` for content pages

Return type
DataFrame

Returns
the dataframe of pages and wikitext

section_topics.pipeline.**load_qids**(*spark*, *weekly_snapshot*, *wikis*, *namespace*)

Load Wikidata QIDs with their page links through the *section_topics.queries.QIDS* Data Lake query.

Parameters

- **spark** (SparkSession) – an active Spark session
- **weekly_snapshot** (str) – a `YYYY-MM-DD` date
- **wikis** (str) – a string of comma-separated wikis to process
- **namespace** (int) – a page namespace number, e.g., `0` for content pages

Return type
DataFrame

Returns
the dataframe of QIDs and page links

section_topics.pipeline.**load_redirects**(*spark*, *monthly_snapshot*, *wikis*)

Load wiki page redirects through the *section_topics.queries.REDIRECTS* Data Lake query.

Parameters

- **spark** (SparkSession) – an active Spark session
- **monthly_snapshot** (str) – a YYYY-MM date
- **wikis** (str) – a string of comma-separated wikis to process

Return type
DataFrame

Returns
the dataframe of page redirects

section_topics.pipeline.**apply_filter**(*df*, *filter_df*, *broadcast=False*)

Exclude rows of an input dataframe given a filter dataframe.

Anti-join all filter columns against input ones.

Parameters

- **df** (DataFrame) – a dataframe to be filtered
- **filter_df** (DataFrame) – a dataframe acting as a filter. Columns must be a subset of df
- **broadcast** (bool) – whether to broadcast filter_df, which tells Spark to perform a *broadcast hash join*, i.e., pyspark.sql.functions.broadcast(). Much faster if filter_df is small

Return type
DataFrame

Returns
the filtered df dataframe

section_topics.pipeline.**look_up_qids**(*pages*, *qids*)

Look up page QIDs through page IDs.

Parameters

- **pages** (DataFrame) – a dataframe of pages as output by *apply_filter()*. Pass the output of *load_pages()* if you want the full raw dataset.
- **qids** (DataFrame) – a dataframe of page IDs and Wikidata QIDs as output by *load_qids()*

Return type
DataFrame

Returns
the pages dataframe with page QIDs added

section_topics.pipeline.**wikitext_headings_to_anchors**(*headings*)

Transform wikitext headings into URL anchors.

For instance, `=== Album in studio ===` becomes Album_in_studio, and serves as a section link in https://it.wikipedia.org/wiki/Gaznevada#Album_in_studio.

Anchors that occur more than once get a numeric suffix in the form anchor_N.

Parameters
headings (List[str]) – a list of wikitext headings

Return type
List[str]

Returns
the corresponding URL anchors

section_topics.pipeline.**normalize_heading**(*heading*, *substitute_re=re.compile('[\\\\s_]')*, *strip_chars='!"#$%&\\' *+, -./:;<=>?@[\\\\]^_`{|}~'*)

Normalize section headings for better matching.

Normalization steps:

- remove `_N` suffixes in case of duplicate section anchors as added by *wikitext_headings_to_anchors()*
- replace characters matched by `substitute_re` with one ASCII white space
- strip leading and trailing `strip_chars`
- lowercase

Note: This normalization is not perfect: it's a trade-off between several ones, some of which may prevent from converging to a lowest common denominator. However, only extreme edge cases might be affected.

Parameters

- **heading** (str) – a section heading without trailing white space or wikitext markup
- **substitute_re** (Pattern) – (optional) a compiled regular expression whose matches will be replaced by one ASCII white space
- **strip_chars** (str) – (optional) a string of characters to be stripped

Return type
str

Returns
the normalized section headings

section_topics.pipeline.**normalize_heading_column**(*column*, *substitute_pattern='[\\\\s_]'*, *strip_chars='!"#$%&\\' *+, -./:;<=>?@[\\\\]^_`{|}~'*)

Normalize a dataframe column of section headings for better matching.

Same as *normalize_heading()*, but implemented with PySpark SQL functions.

Parameters

- **column** (str) – a dataframe column name of section headings
- **substitute_pattern** (str) – (optional) a regular expression pattern whose matches will be replaced by one ASCII white space
- **strip_chars** (str) – (optional) a string of characters to be stripped

Return type
Column

Returns
the column of normalized section headings

section_topics.pipeline.**normalize_denylist**(*denylist*)

Normalize a denylist of section headings for better matching.

Apply *normalize_heading()* to the given input.

Parameters
denylist (dict) – a dict of `{ wiki: [list of section headings to exclude] }`

Return type
dict

Returns
the normalized denylist

section_topics.pipeline.**parse_excluding**(*section_denylist*, *keep_lists_and_tables*, *minimum_section_size*)

Currying function that passes a denylist of section headings to the underlying *parse()* PySpark user-defined function (UDF).

See also this gist.

Parameters

- **section_denylist** (dict) – a dict of *normalized* `{ wiki: [list of section headings to exclude] }`. Pass an empty dict for no denylist. You can normalize via *normalize_denylist()*
- **keep_lists_and_tables** (bool) – whether to keep sections with at least one standard wikitext list or table
- **minimum_section_size** (int) – minimum content character length for the section to be considered

Return type
udf

Returns
the actual UDF

section_topics.pipeline.**extract_sections**(*articles*, *keep_lists_and_tables*, *minimum_section_size*, *denylist={}*)

Apply the *parse()* UDF to extract sections and wikilinks from articles.

Create one row per link and don't select processed columns.

Parameters

- **articles** (DataFrame) – a dataframe of article pages as output by *look_up_qids()*
- **keep_lists_and_tables** (bool) – whether to keep sections with at least one standard wikitext list or table
- **minimum_section_size** (int) – minimum content character length for the section to be considered
- **denylist** (dict) – (optional) a dict of *raw* `{ wiki: [list of section headings to exclude] }`

Return type
DataFrame

Returns
the dataframe of sections and links extracted as per *parse()*

section_topics.pipeline.**normalize_wikilinks**(*link_column*)

Lowercase the first character of wikilink target titles.

The link target is case-sensitive except for the first character.

Parameters
link_column (str) – a dataframe column name of wikilinks

Return type
Column

Returns
the column of normalized wikilinks. None values are kept.

section_topics.pipeline.**handle_media**(*sections*, *media_prefixes*)

Separate media links from other ones.

Detect media links via lowercased lookup of namespace prefixes.

Parameters

- **sections** (DataFrame) – a dataframe of sections and wikilinks as output by *extract_sections()*
- **media_prefixes** (list) – a list of namespace prefixes for media pages

Return type
Tuple[DataFrame, DataFrame]

Returns
the dataframe of media links and the remainder of the sections dataframe

section_topics.pipeline.**clean_up_links**(*sections*)

Filter empty strings and add a column of normalized wikilinks.

Parameters
sections (DataFrame) – a dataframe of sections and wikilinks as output by *extract_sections()*

Return type
DataFrame

Returns
the cleaned sections dataframe

section_topics.pipeline.**resolve_redirects**(*sections*, *redirects*)

Follow section wikilinks redirects.

If a wikilink points to a redirect page, replace its title with the redirected one. Normalize redirects via *normalize_wikilinks()*.

Parameters

- **sections** (DataFrame) – a dataframe of sections and cleaned wikilinks as output by *clean_up_links()*
- **redirects** (DataFrame) – a dataframe of page titles and redirected page titles as output by *load_redirects()*

Return type
DataFrame

Returns
the dataframe of redirected wikilinks. Both original and normalized ones are kept.

section_topics.pipeline.**gather_section_topics**(*sections*, *articles*, *categories*)

Align section wikilinks to their Wikidata QIDs: the so-called **section topics**.

Parameters

- **sections** (DataFrame) – a dataframe of sections and normalized wikilinks as output by *resolve_redirects()*. Pass the output of *clean_up_links()* to skip wikilinks pointing to redirect pages.
- **articles** (DataFrame) – a dataframe of articles as output by *look_up_qids()*
- **categories** (DataFrame) – a dataframe of categories as output by *look_up_qids()*

Return type
DataFrame

Returns
the dataframe of sections and topics (as Wikidata QIDs). Original topic titles are kept.

section_topics.pipeline.**compute_relevance**(*topics*, *level='section'*)

Compute either the section-level or the article-level relevance score for every section topic.

The section-level score is a standard term frequency-inverted document frequency (TF-IDF). The article-level score is a custom TF-IDF, where TF is **across wikis** and IDF is **within one wiki**.

Workflow:

1. filter null topic QIDs
2. compute TF numerator: occurrences of *one* topic QID in a section or page QID
3. compute TF denominator: occurrences of *all* topic QIDs in a section or page QID
4. compute TF: numerator / denominator
5. join with input on section or page QID and topic QID
6. compute IDF numerator: count of sections or page QIDs in a wiki
7. compute IDF denominator: count of sections or page QIDs where a topic QID occurs, in a wiki
8. compute IDF: log(numerator / denominator)
9. join with input on wiki and topic QID
10. compute TF-IDF: TF * IDF

Parameters

- **topics** (DataFrame) – a dataframe of section topics as output by *gather_section_topics()*
- **level** (str) – (optional) at which level relevance is computed, `section` or `article`

Return type
DataFrame

Returns
the input dataframe with the `tf_idf` column added

section_topics.pipeline.**compose_output**(*scored_topics*, *all_topics*, *snapshot*, *page_namespace=0*)

Fuse scored topics with null ones and build the output dataset.

Parameters

- **scored_topics** (DataFrame) – a dataframe of scored topics as output by *compute_relevance()*
- **all_topics** (DataFrame) – a dataframe of all topics as output by *gather_section_topics()*
- **snapshot** (str) – a weekly snapshot to serve as the constant value for the `snapshot` column of the output dataframe
- **page_namespace** (int) – (optional) a page namespace to serve as the constant value for the `page_namespace` column of the output dataframe

Return type
DataFrame

Returns
the final output dataframe

section_topics.pipeline.**parse**(*wiki*, *wikitext*, *section_level=SECTION_LEVEL*, *section_zero_title=SECTION_ZERO_TITLE*)

Note: This is the core function reponsible for the data heavy lifting. It's implemented as a PySpark user-defined function (pyspark.sql.functions.udf()). It must be called by the *parse_excluding()* currying function, which seems the only way to pass an optional denylist of section headings.

Parse a wikitext into section indices, titles, and wikilinks. The lead section is included.

A section title is the wikitext heading normalized through *wikitext_headings_to_anchors()*. Clean wikilinks with mwparserfromhell's strip_code(). This can lead to empty strings that should be filtered.

Parameters

- **wiki** (*str*) – a wiki
- **wikitext** (*str*) – a wikitext
- **section_level** (*int*) – (optional) a section hierarchy level to be extracted
- **section_zero_title** (*str*) – (optional) a title reserved to the lead section

Returns
the list of *(index, title, wikilinks)* section dictionaries

Return type
List[str]

6.2 Queries to the Data Lake

A set of Spark-flavoured SQL queries that gather relevant data from the Wikimedia Foundation's Analytics Data Lake.

section_topics.queries.**PAGES = "SELECT wiki_db, revision_id, page_id, REPLACE(page_title, ' ', '_') AS page_title, revision_text\nFROM wmf.mediawiki_wikitext_current\nWHERE snapshot='{monthly}' AND wiki_db IN ({wikis}) AND page_namespace={namespace} AND page_redirect_title=''\n"**

Gather pages with their wikitext.

section_topics.queries.QIDS = "SELECT wiki_db, item_id, page_id\nFROM
wmf.wikidata_item_page_link\nWHERE snapshot='{weekly}' AND wiki_db IN ({wikis}) AND
page_namespace={namespace}\n"

Gather Wikidata items with their page links.

section_topics.queries.REDIRECTS = "SELECT wiki_db, REPLACE(page_title, ' ', '_') AS
page_title,\nREPLACE(page_redirect_title, ' ', '_') AS page_redirect_title\nFROM
wmf.mediawiki_wikitext_current\nWHERE snapshot='{monthly}' AND wiki_db IN ({wikis}) AND
page_namespace=0 AND page_redirect_title!=''\n"

Gather page redirects.

PYTHON MODULE INDEX

S

INDEX