Debate Crawler

The crawler crawls five popular opinion pages from *debate.org* and saves them to *data.json* file.

We used debate.org as `allowed_domains` to crawl the spider and base_url which is used to crawl top 5 pages dynamically using `parse_1`. In `settings`, we are overriding `ROBOTSTXT_OBEY = False` and declared `FEED_FORMAT= Json` and `FEED_URI=data.json` as we are saving the crawled data in Json format. To load more information dynamically on ajax call, `ajax_url` and `headers` is declared after that. An item object containing three elements url,topic and category is created in class `DataAcquisitionItem`. The following part discusses how several functions of the crawler work.

construct_json_str(index, debateId)

This is a static method which takes `index` and `debateId` as input perameter and returns a Json string which is used to show the logging info using `loging.info`.

start_requests(self)

This function passes the url in `scrapy.http.Request` class and writes the response in `data.json` file.

parse_1(self, response)

This function dynamically selects top 5 opinion pages for crawling. The url of the opinions is found from the `<img>` tag under class `a-image-contain`. Finally, it yields the response for those webpages which is generated from `scrapy.http.Request` class.

parse(self, response)

The `parse(self, response)` function is called after every successful crawl and then initial parsing is done on the returned HTML code to extract title and category of each debate.

- `span.q-title::text` selector is being used for extracting the title text enclosed between the `<span>` tags and `q-title` class excluding the HTML code.
- To extract the `category` of the debate topic, `response.css('div#breadcrumb a::text')[2].get()` is selecting the set of all text under the breadcrumb div and `<a>` tag, then it takes the third level of div.
- `scrapy.http.Request()` is yielded to iterate over all pages of the link passed with `ajax_url` and parse them further with callback function `parse_detail`. A dict is defined in the meta parameter with parsed items and passed to `parse_detail` function to be used later.

parse_detail(self, response)

With `parse_detail` function, each response is further parsed to extract pro and con arguments and their titles. First the json response is loaded from `response.body`. The extrcated data is then splitted and first half is stored as pro argument and second half as con argument. For every pro and con argument response, with `li.hasData` selector, the argument is extracted and a counter is incremented. The argument title is then extracted from between `<h2>` tags and the raw argument body from `<p>` tags. All additional html tags are removed later and stored in `body`. All arguments are tokenized with `nltk` and the length of each argument is stored in `token_counts` list to draw the histogram later. The meta parameter is then updated with `title` and

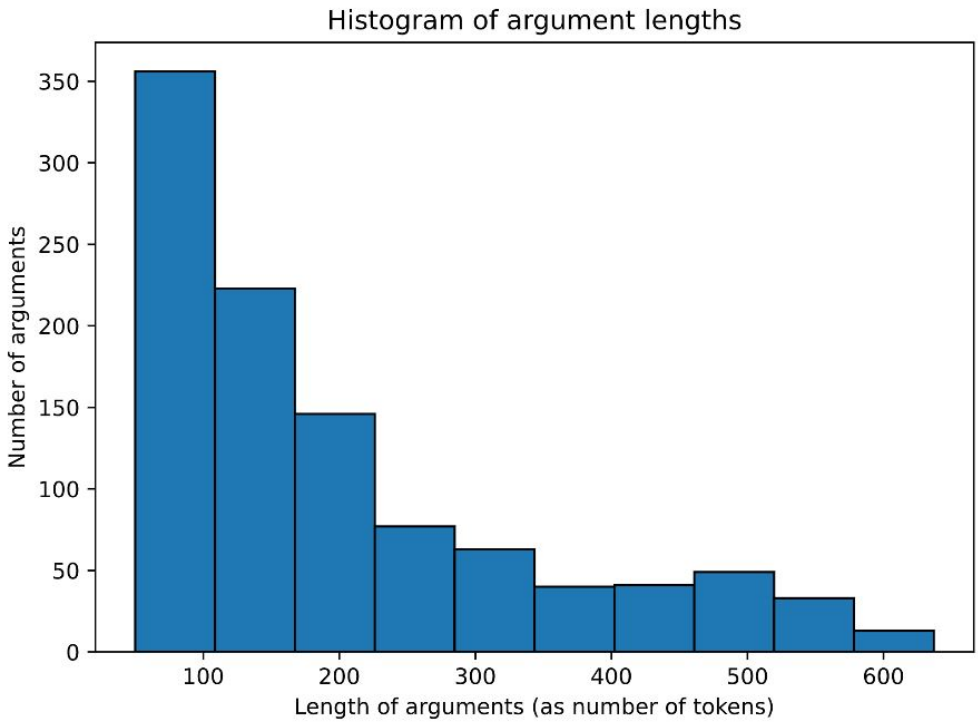
`body` of both types of arguments. When there is no more arguments to parse, the updated parameters are yielded for the following call of the function. Additionaly they are stored in `stats` and `category_stats` for plotting.

Plots

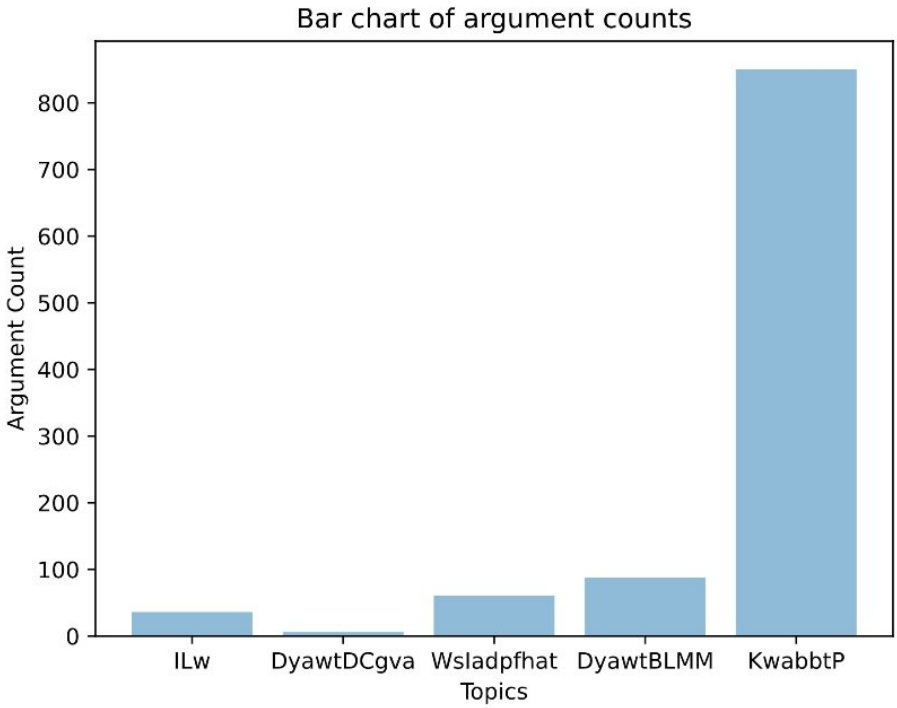
closed(self, reason)

Once the spider closed, preliminary statistics on crawled data are generated in a pdf file.

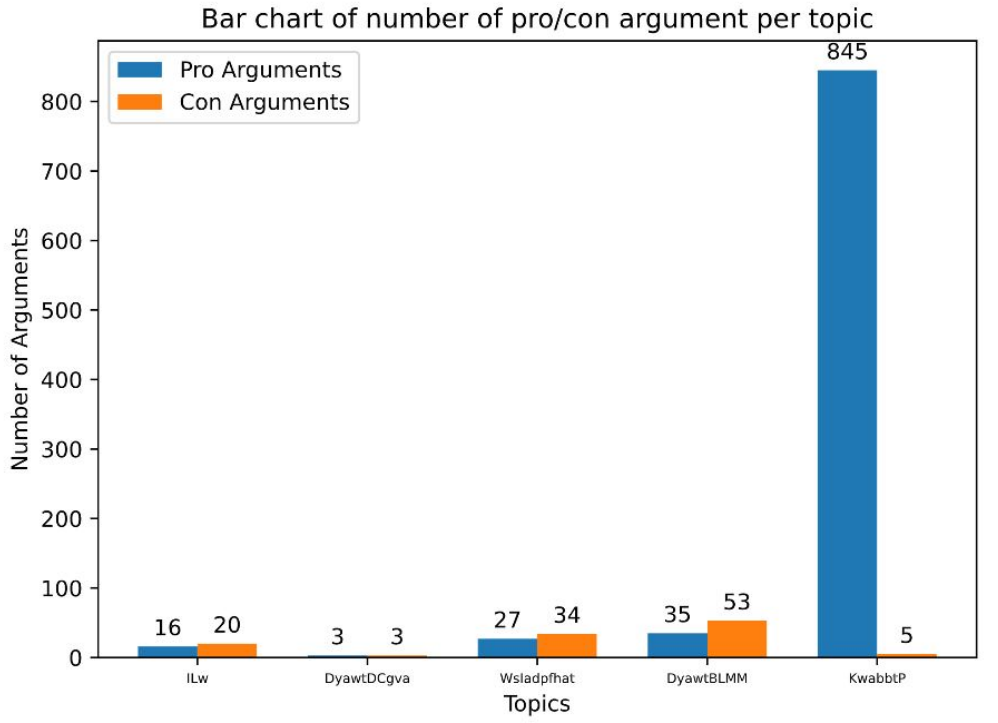
1. **Histogram of argument lengths** In this plot, x axis illustrates the length of arguments and y axis total number of arguments for top five popular pages. We considered the total number of tokens as the length of arguments. The token number is the total count of words that was presented in the body of both pro and con arguments.



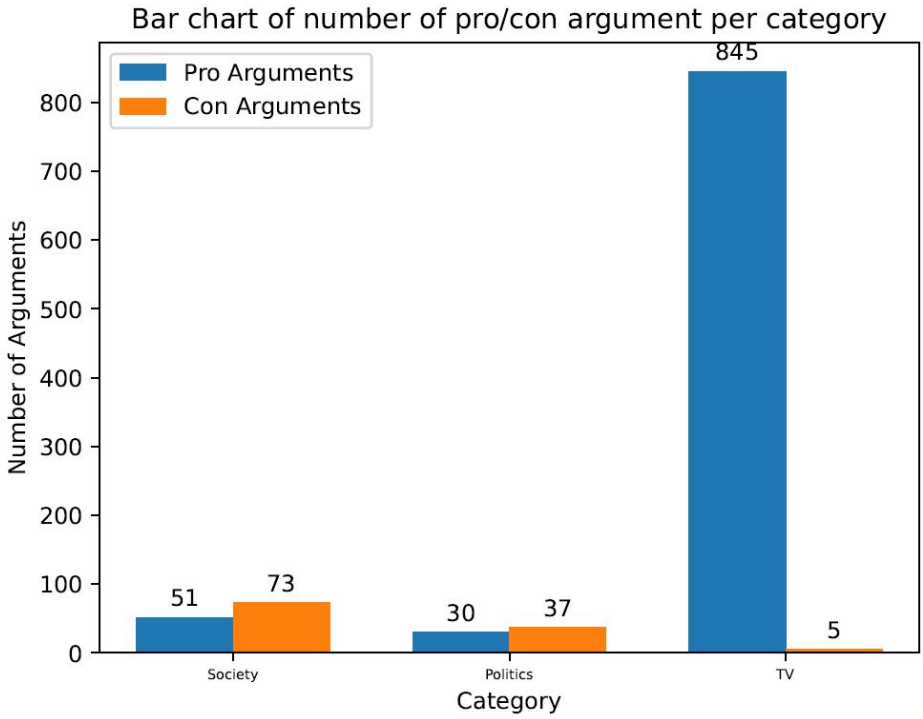
2. **Bar chart of argument counts** Here, x axis represents five different topics and y axis represents the total argument count. Argument count is the number of pro and con arguments for each topics.



3. **Bar chart of number of pro/con argument per topic** The bar chart depicts the number of pro and con arguments for each topic.



4. **Bar chart of number of pro/con arguments per category** This bar chart depicts the number of pro and con arguments for each distinct category.



Instructions to run the code

```
scrapy crawl debate_crawler
```

Required Libraries

- json
- scrapy
- matplotlib
- numpy
- nltk
- w3lib
- logging