<https://ws-dl.blogspot.com/2017/03/2017-03-20-survey-of-5-boilerplate.html>

* 1. I used “**Justtext**” : Gives tags on the image and remove some parts of the text. It capture the headers as header ( it was designed to preserve mainly text containing full sentences, thus, well suited for creating linguistic resources. ) But not put them in the correct class(it removes them).There is a API to test it: <https://nlp.fi.muni.cz/projects/justext/>
  2. <https://getpocket.com/developer/docs/v3/article-view> : I couldn’t find anything useful how to use it.
  3. I used **python-boilerpipe**: It doesn’t give the title . It shows video alternative. Includes the text for related articles in between. The link to the repository : <https://github.com/misja/python-boilerpipe>
  4. <https://boilerpipe-web.appspot.com/> which is a **Web API for the boilerpipe** Java library. It returns a Json which gives title and content. Sometimes the text is not complete. It shows video alternative. It can include a link label or some icon alternatives. (play video, just watch, must watch, read more, , pinterest, Share via Email, Shares, Advertisement but all of this words are between 2 \n). Includes the text for related articles in between(I still didn’t manage to try the local version).
  5. <https://www.diffbot.com/products/automatic/> : They give title, date, author. It gives the best result but it also gives caption and label of the image as text or includes a part of the ad for signing for the newsletter, removes the sub header, more button appears as text.
  6. **Mercury parser**: is available as a chrome extension and open source project: <https://github.com/postlight/mercury-parser>. The parser gives a json file. The part that includes the main text is returned in html format(content). The other keys are title, author, published date, lead\_image\_url, url, domain, excerpt(sub header), word\_count, direction, total\_pages, rendered\_pages. It is wriiten in javascript. It can be used as a chrome extension or used on local machine by nodejs. *The good news is that you can pass pre-fetched html as an argument but for facebook pages if you save the page you will not see the content*. In general passing the pre-fetched html page gives a bad result since it includes other parts of the page. It leaves HTML tags intact inside the “text” content, so we can still see afterwards what was a link or a caption for a link, which is not the case for boilerpipe (which removes the< a href > for example).
  7. **Portia** is a tool that allows you to visually scrape websites without any programming knowledge required. With Portia you can annotate a web page to identify the data you wish to extract, and Portia will understand based on these annotations how to scrape data from similar pages. The link the repository is: <https://github.com/scrapinghub/portia>. In portia you have to annotate (An annotation defines the location of a piece of data on the web page and how it should be used by the spider.) each domain to make at least one sample. When the crawler visits a page, it matches the page against each sample. Samples with more annotations take precedence over those with less. Spiders consist of one or more samples and each sample is made up of annotations that define the elements you wish to extract. This is more useful if you have a small set of domain and you can annotate them by hand to extract features you want. For example you can annotate an image and it its price so the spider finds all the items and their price on a online-shop.
  8. **Newspaper** (Python3 library): given a url it download the html. It parses the article to extract author, publish date, text, top image(gives the Url ). As I tried different articles it is not very successful to extract the features, but an interesting feature is that you can call article.nlp() which you can use to get article keywords and article summery. For more information: <https://newspaper.readthedocs.io/en/latest/> .
  9. **Readability (Java Script ):** A standalone version of the readability library used for Firefox Reader View( it is a Firefox feature). The link to the repository is : <https://github.com/mozilla/readability>

There is a python wrapper for readability here: <https://github.com/alan-turing-institute/ReadabiliPy>

The wrapper is based on an old version of Readability.js (commit 876c81f which is committed on Nov 20, 2018 ) so I replaced the file with the newest version (Commit d6fc38c on Oct 21,2019).I also did a few changes to make it compatible with python 3. The function simple\_json\_from\_html\_string()takes the html document and return a dictionary with the following fields:

title: The article title

byline: Author information

content: A simplified HTML representation of the article, with all article text contained in paragraph elements.( I used this to compare with the mercury)

plain\_content: A "plain" version of the simplified Readability.js article HTML present in the content field. This attempts to retain only the plain text content of the article, while preserving the HTML structure.

plain\_text: A list containing plain text representations of each paragraph (<p>) or list (<ol> or <ul>) present in the simplified Readability.js article HTML in the content field. Each paragraph or list is represented as a single string. List strings look like "\* item 1, \* item 2, \* item 3," for both ordered and unordered lists (note the trailing,).

**How to install Mercury:**

1. **Mercury chrome extension:** go to <https://chrome.google.com/webstore/detail/mercury-reader/oknpjjbmpnndlpmnhmekjpocelpnlfdi?hl=en> and add it to your chrome. It is good for visualization so you can see the output of mercury.
2. **Mercury Parser( Java Script):** 
   1. Download and install npm : <https://www.npmjs.com/get-npm>
   2. Install mercury: in your command line write: ‘’npm -g install @postlight/mercury-parser’’
   3. Now you should be able to use it ,in cmd try: ‘’mercury-parser URL’’

You can pass different parameters also as it is mentioned in: <https://github.com/postlight/mercury-parser> (but I couldn’t find a way to pass the html file)

* 1. **Automatic parsing:**Install pixiedust\_node using !pip install pixiedust\_node so you can run javascript inside jupyter notebook then open mercury.ipynb to parse as many html you want!
  2. **Not automatic parsing:**You can also create a .js file and follow the documentation in their github or copy the folder that I gave, open the file mercury.js , change the name of the html file that you want to parse(the url can be anything it does not matter, it is just a default url in case it cant open the page), save it, open cmd in the same directory and type ‘’node mercury.js’’ .

1. **Mercury parser (Python):** There is a python api wrapper for Mercury parser which can be found in: <https://github.com/Girbons/mercury-parserpy> I couldn’t make it run because the API refuse the connection but this wrapper is just for passing urls and you can pass HTML documents ( see <https://github.com/Girbons/mercury-parserpy/blob/master/mercury_parser/client.py>)

**Mercury vs Readability:**

As I compared the first 7 urls in ‘report.xlsx’ ,

Readability: The author is recognized most of the time, it is slow, and can recognize the subtitle most of the time

Mercury: The author is missing most of the time but the subtitle is recognized as ‘excerpt’ and mercury is much faster than readability.