Evaluating quality of article body extraction for commercial services and open-source libraries

Konstantin Lopukhin
Scrapinghub
konstantin@scrapinghub.com

Abstract

We describe the task of article extraction and evaluate the quality of article body extraction for AutoExtract[1] (ours), Diffbot[2], newspaper3k[3], readability-lxml[4], dragnet[5], boilerpipe[6] and html-text[7]. We release evaluation datasets and scripts, describe our criteria for a high quality extraction, and analyse results.

1 Introduction

Article extraction is the task of extracting multiple fields from an article page given its URL. An "article" page could be a news article, blog post, press release, etc. The main fields are the headline, article body (text of the article), cleaned HTML of the article, publication date, authors, main image and other images. Article body is probably the most important and the hardest to get right, so we concentrate on it in this report.

The task may look straightforward, but it is surprisingly tricky and nuanced. Getting an article body right requires not only understanding where an article begins or ends, but also which parts to exclude from it: for example, there are often ads, links to "related" (not really) content, share buttons, author information, and other undesired elements. For an example, see Figure 1 below and Figure 4 further.

Figure 1: Examples of excluded content (crossed in cyan) and desired content (in green)

Here we consider only generic extraction systems, which can work on any page or any domain without additional tuning. Writing a good rule-based article extraction system for just one domain is already hard, and maintaining it for thousands of domains is extremely labour intensive. That's why

[1] https://scrapinghub.com/data-api-news
[2] https://www.diffbot.com/products/automatic/article/
[3] https://github.com/codelucas/newspaper
[4] https://github.com/buriy/python-readability
[5] https://github.com/dragnet-org/dragnet
[6] https://github.com/misja/python-boilerpipe
[7] https://github.com/TeamHG-Memex/html-text

all systems we evaluate today are based on either machine learning, a set of generic heuristics, or both.

For this comparison, we collected a diverse set of URLs: news articles from popular domains and less popular sites and less typical pages, including blogs and non-news articles. In the end, the size of the dataset was 181 pages. Both commercial services (AutoExtract and Diffbot) get a page URL, render it in a headless browser and extract various features, including visual features, HTML structure and text. Open-source libraries in this comparison take only HTML as input, which was obtained from the headless browser.

During the evaluation, we compare extraction results from each system with "ground truth" results, that is, desired output. For a fair comparison, we need to make sure that all systems follow the same rules which we use for ground truth annotation. We checked AutoExtract and Diffbot output in considerable detail and it turned out that both services have the same goals behind them, most likely informed by what the market wants.

For the final metric we used precision, recall and F1, which is a common choice for such evaluations. Precision here measures how well the systems are at excluding undesired parts of the article body, and recall measures how well the systems are at capturing desired parts of the article body. F1 is a summary of precision and recall measuring overall quality (a harmonic mean), but does not have a nice interpretation here. Additionally, we checked accuracy, which measures the ratio of "perfect" articleBody predictions, having exactly the same content as ground truth after normalization. This metric is easy to understand but quite noisy, as a single small difference makes the whole page a failure.

2 Main results

We give more details about all aspects of evaluation below, but let's check results first, see Figure 2, Figure 3, Table 1.

Table 1: Main results

	F1	Precision	Recall	Accuracy
AutoExtract	0.970 ± 0.005	0.984 ± 0.003	0.956 ± 0.010	0.470 ± 0.038
Diffbot	0.951 ± 0.010	0.958 ± 0.009	0.944 ± 0.013	0.348 ± 0.036
readability	0.922 ± 0.013	0.913 ± 0.014	0.931 ± 0.015	0.315 ± 0.035
newspaper3k	0.912 ± 0.014	0.917 ± 0.014	0.906 ± 0.018	0.260 ± 0.033
dragnet	0.907 ± 0.014	0.925 ± 0.012	0.889 ± 0.019	0.221 ± 0.031
boilerpipe	0.860 ± 0.016	0.850 ± 0.016	0.870 ± 0.019	0.006 ± 0.006
html-text	0.665 ± 0.015	0.500 ± 0.017	0.994 ± 0.001	0.000 ± 0.000

Figure 2: Ratio of undesired content (lower is better)

Figure 3: Ratio of missed content (lower is better)

While F1 is the main single metric, we think that looking at precision and recall is more informative. For example, "html-text", which has the worst F1, has the best recall, because this is a baseline system which extracts all text from an article (can you guess why recall is not perfect?). If we

compare AutoExtract and Diffbot, we see that AutoExtract is better on all metrics, but only the precision difference is significant (in the table and charts above we show standard deviation estimated with bootstrap): the value of 0.984 means that text extracted by AutoExtract would have only 1.6% of unneeded content on an average webpage, compared to 4.2% for Diffbot. While looking at recall values, we see that AutoExtract would miss 4.4% of desired content, compared to 5.6% for Diffbot. All open-source libraries we studied have significantly worse metrics, but it's interesting that some modern libraries (such as readability, newspaper, dragnet) are significantly better than an older boilerpipe. In terms of accuracy, the order of different systems is the same as for F1, but the difference is bigger. Notably, even the best score is only 0.47, so about half of the pages still have some differences with ground truth, however minor.

Here are our main take-aways from this study:

- **AutoExtract vs Diffbot quality assessment.** Based on results of this comparison, we're confident that AutoExtract quality is the best overall, and we have customers providing us with the same feedback. Still, there can be biases in our evaluation which we missed, so external evaluation is always welcome.
- **Quality of Open Source libraries** is significantly worse than the quality of commercial services (e.g. even the most precise OSS library provides 4.6x more unwanted content in the results, while missing 2.5x more content than AutoExtract). That said, they do work, and quality is not that bad; it can be adequate for some use cases, where unclean data is acceptable. We provide more analysis below. Boilerpipe library should be probably avoided for the purpose of clean article body extraction, if using better performing Python libraries is an option, as it provides significantly worse quality, especially in terms of precision.
- When the goal is to make sure no content is missing, use html-text[8] or a similar "dumb" library, because it gives the best recall. Keep in mind that even basic html $\rightarrow$ text conversion is surprisingly tricky and easy to get wrong: simple approaches, like XPath `string()` functions, can produce noisy results: bad whitespaces, missing or extra text. Because of this reason, we suggest using a library, where these issues are solved.
- Besides better quality, commercial services provide many other features. For example, AutoExtract can get you cleaned and normalized HTML of the article[9], article author, headline, datePosted, images and many other attributes. Commercial services also handle downloading, which is a whole different aspect, significantly affecting the final result. This comparison focused only on the quality of article body extraction.

3 What is an article body?

For a fair evaluation, we need to define what should be included into the article body, and make sure that all systems follow the same goals. Underlying principle is that the article body should be "clean text", without other fields (like author), navigation elements, ads, etc. At the same time we want to include all sub-headings, block-quotes and other elements which constitute article content. Here is a list of items not included into an article:

- other fields such as authors, publication date, keywords, image captions, etc., and markup around them
- share buttons and suggestions to share an article
- comments and UI related to comments (e.g. "Comments" header, forms, links to comments)
- "related" articles, "read next" links, "recommended to you", etc.: any links which are not part of the article and whose goal is to keep the reader on the platform
- newsletter signup links, generic calls to action, various forms
- advertisements
- author information, usually at the end of the article (such as "X X is an X reporter ..."), as this is not directly related to an article

[8] `https://github.com/TeamHG-Memex/html-text`

[9] `https://blog.scrapinghub.com/extracting-clean-article-html-with-autoextract`

- control elements around images which produce unneeded text (e.g. number of images in a gallery, buttons overlaid over an image/video, etc.)
- blocks denoting reading time of the article
- other items unrelated to the article content

Here is a list of items that should be included into an article (except for the article body proper):

- links to read in more detail, to the source, or to other content directly related to this article: these might require deeper understanding of the article content to annotate correctly
- embedded posts related to the article, such as tweets/instagram/facebook posts
- copyright notices at the end

Figure 4 provides some examples, where desired content is highlighted in green, and undesired is crossed-out or highlighted in cyan.

We checked AutoExtract and Diffbot output in considerable detail and verified that both systems follow above guidelines, occasionally making understandable errors.

Note: even though these rules are a good default for most use cases, and reasonable for evaluation purposes, sometimes a concrete task may require something different. For example, if the article body is passed as an input to a ML-based classifier (e.g. to categorize articles), it could be beneficial to keep image captions in the article body itself. For AutoExtract it is possible to customize the output using articleBodyHtml field[10].

4 How was the dataset collected?

First step was URL seeds collection. We wanted the dataset to be diverse, representative and unbiased. By diversity we mean that it should contain different kinds of pages in different languages, not just news articles in English, but also a variety of non-news articles. A representative dataset should contain pages which are similar to what people want to achieve by article extraction, including modern complex news articles. We wanted the dataset to be unbiased in a sense that it should be different from our internal datasets in terms of how the initial URLs were collected, to avoid biasing results in our favor.

We did URL collection in two stages, both being significantly different from the approach we use for internal training or testing datasets. First stage was obtaining long-tail URLs: for that we took a random sample of 1,000 domains from top million most popular sites according to Alexa. For each site, we checked if it contained any articles, and if it did, added two random articles from it. This resulted in a very diverse set of pages in different languages, including not only news articles, but also blog posts, non-news articles, press releases, etc. We obtained around 120 URLs from around 60 domains this way. Second stage was getting news articles from modern popular sites. For that, we went to `https://news.google.com/` in incognito mode, set "Language and region" to "English (United States)", and collected all links to articles, getting around 190 URLs from 90 domains. Next we went to a news aggregator `http://www.gigablast.com/news` and obtained all article links from the first 3 pages, getting around 60 URLs from 40 domains. After all, we combined all URLs, excluding youtube links, obtaining 356 URLs from 189 domains (many domains had more than 10 pages).

Next we extracted articles from all seed URLs via AutoExtract and Diffbot at the same time, minimizing chances of getting an article updated. Later we also made sure that pages didn't change by analyzing extraction results. If one of the systems was not able to download a page, or a downloaded page had unexpected content, we excluded it from the evaluation entirely, to focus on extraction quality only: Open Source libraries were not supposed to download anything, and we didn't want to penalize Diffbot for having less pages downloaded, as it could be a temporary issue. Analysis of download success/failure rates is out of scope for this comparison - it is large enough already. HTML files for open-source libraries were obtained from the headless browser output.

After that, a maximum of two pages per domain were annotated. After performing initial evaluation, we had to exclude a few pages which we could not evaluate properly. We had to exclude 2 pages

[10] `https://blog.scrapinghub.com/extracting-clean-article-html-with-autoextract`

Figure 4: Examples of excluded content (crossed in cyan) and desired content (in green)

where clearly different content was downloaded by AutoExtract and Diffbot (e.g. Diffbot didn't handle some non-ascii URLs well). We also excluded 6 pages in Japanese and Korean where slightly different whitespace handling made it hard to compare results using our metric fairly. At the end, we were left with 181 pages with a non-empty article body. This dataset is released, see below for more details.

5 How is the metric defined?

Measuring precision, recall and F1 is a common and sensible choice for such evaluation. For that particular task, precision measures how "clean" is the output article body - in other words, how well unneeded content is excluded. Recall measures how well does the system preserve desired parts of the article body. And F1 is a common way to combine precision and recall.

But the most interesting part is how do we actually compute precision and recall. A possible choice is to treat the extraction result as a set of words (also known as "bag" of words), and then calculate precision and recall based on such sets. Our approach is different in two aspects. First, we use n-grams instead of words: this makes sense because single words can be seen both in the desired text of the article, and in undesired part of it, while for n-grams (we use 4-grams) it's significantly less likely. Another difference is that a set of words does not penalize unneeded repeated text, so instead of a set we also consider counts of n-grams. You can check the full metric implementation in the evaluation script[11]. Now you can probably guess why html-text, which extracted all text, didn't get a perfect recall score: because if we need to exclude some text in the middle, e.g. "bad" from "good good bad very good", then ground truth output would contain 4-gram "good good very good", while html-text output would lack it.

The final aspect of the metric is how to obtain n-grams (in other words, tokenization): for that we use a very simple approach of extracting all alpha-numeric sequences, discarding extra whitespace and punctuation. For example "- This is a 2020/01 article!Next" input would be normalized to "This is a 2020 01 article Next". This works well in most cases, as usually text to be excluded lies in separate blocks separated by newlines or spaces. But there are some interesting cases here: first, different systems might insert whitespace in different places, e.g. if we have HTML like "aabb" then it's not immediately clear if a space should be added between "aa" and "bb". In practice we have seen some weird markup like "A day in ...", where it's clear that "A day in ..." is correct. Another case when this tokenization approach can be problematic is languages like Japanese, where words commonly don't have spaces separating them, and a small difference in space handling can lead to big changes in the final metric. For this comparison, we opted to exclude 6 pages in Japanese and Korean where we were not confident in our ground truth and didn't have a smart enough tokenizer for evaluation. Excluding them doesn't mean the tools we're comparing were not working with these languages, it is just that evaluation was a bit more complicated in these cases, and we decided those few pages were not important enough. Additionally to measuring precision, recall and F1, we added accuracy, which measures the ratio of "perfect" article bodies, which have exactly the same content as ground truth after normalization. This metric is easy to understand but quite noisy, as a single small difference renders the whole page as a failure. Normalization here was performed by joining all tokens with whitespace, using the same tokenization as above.

Since the dataset is quite small (only 181 pages), there is a great deal of uncertainty in results. Bootstrap allows us to quantify this uncertainty: we took 1000 samples from the dataset with replacement and calculated mean and standard deviation. This allows us to see that difference in precision is much more significant compared to recall differences, and that accuracy is indeed quite noisy, as you can see in the results table.

6 Error analysis

We did an extensive comparison of ground truth and extraction results for AutoExtract and Diffbot, and noticed that a lot of errors are quite similar, although there are some differences:

[11] `https://github.com/scrapinghub/article-extraction-benchmark/blob/master/evaluate.py`

- A common error is when some extra text is present at the end of the article, e.g. author information – such cases are often hard to judge during annotation as well.
- In some cases both systems exclude links to the original article or to the source towards the end, confusing them with links to unrelated content which also often come at the end in similar style. Correct annotation often requires understanding the article in question, so it's not surprising automated systems struggle here.
- Figure caption present in the output is a more common error for Diffbot, especially when there are multiple elements around the image, e.g. one for copyright and one which describes what is in the image, although AutoExtract also has some amount of errors here.
- Subheadings in non-default style (not black) are sometimes erroneously excluded by Diffbot, perhaps they are confused with headings for unrelated links.
- In some cases, AutoExtract inserts unneeded spaces around inline tags, e.g. "Macbook s" instead of "Macbooks".
- Not excluding all "sharing" suggestions happens with both systems, although it's quite rare.
- Diffbot is more likely to exclude tables or more often parts of tables, although it's a rare occurrence.
- In a few cases, Diffbot output stopped too early, with large portions of the article body missing – e.g. if an article was broken up with a big advertising block.
- Failure to exclude links to "related" (not really) content seemed to happen more for Diffbot than to AutoExtract.

But overall, the errors both systems were making were quite similar and expected, given how complicated and sometimes ambiguous is the task. There is always room for improvement though! We actually fixed some of the AutoExtract issues found during this evaluation; of course, these improvements are not included in the metrics because of fairness considerations.

7 Open-source libraries

We also included the following open-source libraries:

- newspaper3k[12]: news, full-text, and article metadata extraction in Python 3.
- readability-lxml[13]: fast python port of arc90's readability tool, updated to match latest readability.js.
- dragnet[14]: an ML web page content extraction library.
- boilerpipe[15]: a python interface to Java boilerpipe library.
- html-text[16]: a baseline which extracts the full text of an HTML page.

We selected some popular and well-supported libraries (such as readability), we also added dragnet as it was featured in a moz.com benchmark[17], and also an older but well-known boilerpipe library. No retraining or tuning was performed, except for choosing boilerpipe extractor which provided the best score.

When checking predictions qualitatively, we noticed that open-source libraries make two kinds of errors:

- Not excluding unrelated content: this is expected given our high requirements for what should be excluded. Also not all libraries might follow the same article body criteria: e.g. boilerpipe tends to include a lot more content both at the start and at the end of the article which we consider to be unrelated.

[12] `https://github.com/codelucas/newspaper`
[13] `https://github.com/buriy/python-readability`
[14] `https://github.com/dragnet-org/dragnet`
[15] `https://github.com/misja/python-boilerpipe`
[16] `https://github.com/TeamHG-Memex/html-text`
[17] `https://moz.com/devblog/benchmarking-python-content-extraction-algorithms-dragnet-readability-goose-and`

- Occasionally making quite significant errors when determining the article body, with large chunks excluded or included incorrectly.

While the first kind of errors happens with AutoExtract and Diffbot as well (although less often), the second kind is more serious and is extremely rare for both commercial systems, but still common for open-source libraries. We realise that in our comparison open-source libraries work on bare HTML, while commercial services have access to the rendered page, but we are not aware of open-source libraries using a similar approach.

We also plotted how precision and recall scores are distributed across different pages for all tested systems:

Figure 5: Precision distribution

Figure 6: Recall distribution

8 Dataset and evaluation release

We release the dataset and evaluation scripts on Github[18]. Dataset includes the following:

- Ground truth in JSON, with "articleBody" and "url" fields for all pages
- HTML sources for all pages, rendered in a headless browser, utf-8 encoded and gzipped
- Screenshots of all pages (available from github releases[19] tab)

Evaluation scripts contain full metric implementation, including bootstrap estimation code, and allow to reproduce results of the evaluation. Python 3.6 or later is enough, no dependencies are required.

Acknowledgments

Thanks to the whole Scrapinghub Data Science team for help with dataset collection, analysis and useful suggestions.

[18] `https://github.com/scrapinghub/article-extraction-benchmark`
[19] `https://github.com/scrapinghub/article-extraction-benchmark/releases`