Web Content Extraction Through Machine Learning *

Ziyan Zhou
ziyanjoe@stanford.edu

Muntasir Mashuq
muntasir@stanford.edu

ABSTRACT

Web content extraction is a key technology for enabling an array of applications aimed at understanding the web. While automated web extraction has been studied extensively, they often focus on extracting structured data that appear multiple times on a single webpage, like product catalogs. This project aims to extract less structured web content, like news articles, that appear only once in noisy webpages. Our approach classifies text blocks using a mixture of visual and language independent features. In addition, a pipeline is devised to automatically label datapoints through clustering where each cluster is scored based on its relevance to the webpage description extracted from the meta tags, and datapoints in the best cluster are selected as positive training examples.

Keywords

Content Extraction; SVM; DBSCAN; Web Scraping

1. INTRODUCTION

Content extraction algorithms have been well studied and there are many interesting and useful applications[1][3]. A survey by Laender et al.[5] systematically covers a wide range of techniques used for this purpose. The techniques range from wrapper development to NLP-based and modeling-based approach. Miao et al.[6] proposed a fully automatic tag path clustering approach to extract structures on a single webpage. Sun et al.[7] applied a language independent approach by looking at text density at each area of the document.

We wanted to take a different approach to this problem. Our goal is to extract content of a webpage in a language independent way. We wanted to understand how we (humans) can identify the main content of a webpage even if we do not recognize the language of the website. The relevant information for us for those webpages are: visual features like font size, color and style; line spacing and block size; layout of the webpage; text density in different parts of webpage; density of links etc. In our algorithm we have used these visual features to train a SVM classifier to find out the content of a webpage.

Another aspect of our approach is that we wanted to run the complete pipeline automatically, from data collection —labeling—learning—testing. Specially we tried to avoid manual labeling of the dataset. So we devised a method using clustering to label the dataset.

Our overall algorithm looks like follows: label the collected dataset using clustering, train SVM with the labeled dataset, use SVM model to extract content from new webpages.

2. TERMINOLOGY

Lets define a few terminology that would help us to understand the rest of the paper.

2.1 Webpage

A webpage is a document that is rendered by a browser corresponding to an URL.

2.2 Block

A block is part of a webpage that contains text and have both height and width. For example, a paragraph in the content of a webpage, a navigational link in menu, title of an article—each of these are separate blocks. The blocks are highlighed in *Figure 1*.

Each block is an input datapoint to our clustering and SVM algorithm.

2.3 Content

Content is the main text of a webpage that we aim to extract. In content extraction literature, it is often referred as *gold text*.

3. DATA COLLECTION

The dataset is extracted from several popular English, Simplified Chinese, and Bengali news article websites on the Internet.[1] The Chaos dataset URLs are collected from Sun

*Source code for this project is avaiable on GitHub under the MIT License at `https://github.com/ziyan/spider`

[1]The collected dataset is available at `https://github.com/ziyan/spider/tree/master/data`

Table 1: Collected Dataset

Website	Language	# of Webpages
NPR	English	25
Prothom Alo	Begali	24
QQ	Simplified Chinese	25
Sina	Simplified Chinese	25
TechCrunch	English	16
The Verge	English	16
USA Today	English	20
Chaos	Mixed, includes RTL	189

et al.[7] This dataset contains URLs from diverse websites across the Internet, such as, personal blogs, articles, news etc. See *Table 1* for some information about our dataset.

For all the dataset, we collected the URLs and then extracted the webpage ourselves, as various existing dataset often cleans up the HTML and removes CSS properties that we need. Each webpage is downloaded and rendered in a virtual webkit browser[2], restoring its original layout intended for human audience. Javascript is then injected into the webpage to efficiently inspect DOM elements. Text enclosing blocks are identified. For each DOM element containing text, the algorithm finds its closest parent element that are displayed as block. Inline links and other text decorators are not individually identified. Instead, their common parent element, which might be a paragraph or a div, is identified as a whole. See *Figure 1*.

For each chosen block, a set of features are then extracted, including the size and position of the block, the contained text, the font configurations, color, line height, tag path, etc. In fact, there are over 300 different CSS properties for each block. Our algorithm automatically extracts those with a *non-default* value. Therefore, the number of features varies from block to block in the data collection phase.

4. AUTOMATIC LABELING

4.1 Clustering Blocks

The cluster shape of similar blocks on a webpage (i.e. navigational buttons) are not necessarily well-shaped (i.e. spherical). Also there can be random noises that appears on some of the webpages. Furthermore, it is unclear that how many clusters will there be, since it depends heavily on the design and layout of the webpage. Therefore, we have chosen DBSCAN[4], density-based spatial clustering of applications with noise, as our clustering algorithm. This algorithm handles the problems of unknown number of clusters, unknown shape of clusters and noise quite nicely.

The clustering result shows that, the body text of the articles across multiple webpages, which usually consist of multiple blocks, are clustered together in a single cluster. The navigational links are also clustered together, as well as the links in the footers. Other common elements, like similar sidebar, advertisement, comments, etc., are also successfully clustered. Given all the visual features represented by the CSS properties, the algorithm is quite effective in discerning the different visual elements on the webpages.

Figure 1: Webpage rendered by virtual webkit browser with text blocks identified by red boxes

Table 2: Separate classifier for different website

Website	Precision	Recall	F_1
NPR	100.00%	100.00%	100.00%
Prothom Alo	100.00%	100.00%	100.00%
QQ	100.00%	100.00%	100.00%
Sina	100.00%	100.00%	100.00%
TechCrunch	100.00%	100.00%	100.00%
The Verge	100.00%	100.00%	100.00%
USA Today	100.00%	100.00%	100.00%

4.2 Labeling Clustered Block

Although all the blocks have been clustered quite precisely based on their visual properties, it is not trivial to find out which cluster contains the content text. The DBSCAN algorithm views clusters as areas of high density separated by areas of low density. Cluster number varies from webpage to webpage, and the cluster containing content can be any of those. For our dataset, the number of output clusters ranges from a few clusters to above 20.

Fortunately, most websites have some metadata stored in meta tags in order to accommodate web crawlers. So we can extract the *description* of an article by parsing the meta tags in the webpage. The description usually contains a brief summary or just the first few words of the article content. With that, we are able to calculate the similarity between each cluster and the *description* using the Longest Common Sub-sequence (LCS) algorithm. The longer the common sub-sequence (we call this number "relevance score"), the more likely is the cluster to contain the content text. Note that we used LCS instead of Term Frequency or Inverse Document Frequency to ensure the language independence, as word segmentation varies from language to language.

At first we tried to automatically label the blocks by finding the best cluster local to each webpage on a website. After training the SVM on this labeling, we found that for some websites, the precision on the test set was not ideal. On closer examination, we found that on some rare pages, the best cluster according to that webpage's description is comment instead of the main text. To fix this issue, we implemented *Algorithm 1* which scores the blocks across the entire website (rather than working locally on a single webpage).

```
score := array of zeros with length equal to # of clusters;
for each cluster i of a website do
    for each block j under that cluster do
        score[i] := score[i] + relevance score of block j;
    end
end
Pick the cluster(C) with the highest similarity score;
Label all the blocks of in the same cluster C as 1;
Label all other blocks as 0;
```

Algorithm 1: Labeling from global score in a website

This algorithm performs very accurately and removes the occasional labeling hiccups on a single webpage.

Table 3: Testing on previously unseen websites

Dataset	Precision	Recall	F_1
Chaos	71.75%	20.65%	31.28%

Table 4: Using random mixture of webpages

Dataset	Precision	Recall	F_1
Chaos	96.86%	89.64%	92.83%

5. SVM AND CROSS VALIDATION

Using the collected and labeled text blocks, the web content extraction problem can be formulated as a classification problem, where the goal content consists of multiple text blocks that are classified as content while the other text blocks are classified as non-content.

We have constructed a support vector classifier with a linear kernel to perform text block classification. Due to the imbalance between the number of content blocks and the number of non-content blocks on a webpage, we applied class weights to give our positive (content) examples higher weights.

We have employed three different approaches to the classification problem. For each approach, we have performed 4-fold cross validation to evaluate its performance.

In our first approach, we have trained separate classifier for different website. For each given website, the collected webpages are shuffled then divided into four groups. One of the groups is chosen as evaluation group, while the other three are used as training examples. Intuitively, the support vector classifier in this case tries to learn the underlying structure and template used by the particular website. This approach worked very well. See *Table 2*.

In our second approach, we have trained one classifier model using webpages from multiple websites in the chaos dataset and then tested the model on webpages from the rest of the dataset. The classifier in this case learns some structures from only a subset of websites then tries to generalize the model on previously unseen websites. This approach produced the worst results. See *Table 3*. It make sense that specific visual characteristics of one website rarely appears the same in another website.

And lastly, we have trained one classifier model using random webpages chosen from the chaos dataset then tested the model on the rest of the dataset. In this approach, since the webpages used in training are picked randomly, our training examples included a wide variety of websites. Therefore, it is likely that the classifier have previously seen at least one example webpage per website for the websites in the evaluation set. This approach worked much better than the second approach. See *Table 4*.

The results from the third approach has shown that our algorithm is able to learn and generalize more than one template at a time. Combined with results from the first approach, these results indicate that our algorithm will work well for any website including websites using multiple different templates as long as the classifier is trained on a representative

Table 5: Using text lengths as features

Website	Precision	Recall	F_1
NPR	78.99%	91.34%	84.70%
Prothom Alo	42.14%	75.01%	53.31%
QQ	31.95%	75.43%	44.70%
Sina	62.92%	60.63%	61.69%
TechCrunch	32.84%	83.18%	46.96%
The Verge	53.56%	86.86%	66.08%
USA Today	75.45%	82.98%	78.83%

Table 6: Using tag paths as features

Website	Precision	Recall	F_1
NPR	99.74%	100.00%	99.87%
Prothom Alo	100.00%	100.00%	100.00%
QQ	58.68%	100.00%	73.88%
Sina	94.61%	100.00%	97.22%
TechCrunch	100.00%	100.00%	100.00%
The Verge	92.34%	100.00%	96.00%
USA Today	100.00%	100.00%	100.00%

mixture of example webpages.

6. FEATURE SELECTION

In order to develop deeper insights into our classification problem, we experimented with different feature sets.

6.1 Text Length

A common naive approach to web content extraction is to find the longest contiguous block of text in a webpage. The intuition is that paragraphs in an article are usually long. To establish some baseline, we've experimented with using only text length as feature. The text length of each text block is simply the number of non-whitespace characters in the block. We normalize this number by scaling it to have a zero mean and a variance of one.

The results show that this approach works for some websites but performs quite poorly for others. See *Table 5*. Due to the variation of paragraph length in the real world, text length is not a very good determinator of article content.

6.2 Tag Path

The second approach is to use tag path of the block. Tag path is simply the path it takes to navigate the DOM tree from its root to the text block. It consists of element names for each node along the path. For example, a paragraph inside a *div* container in the *body* of the HTML webpage would have a tag path like *"body > div > p"*. We treat each different tag path uniquely and vectorized this discrete feature.

The results show that this approach works quite well in general but fails for some particular website. We found that in the failure cases, the tag paths are indeed indistinguishable for content and some non-content text blocks. Therefore, this feature alone cannot sufficiently distinguish the two classes of blocks. See *Table 6*.

Table 7: Using CSS selectors as features

Website	Precision	Recall	F_1
NPR	100.00%	100.00%	100.00%
Prothom Alo	100.00%	97.79%	98.87%
QQ	61.74%	100.00%	73.94%
Sina	97.56%	100.00%	98.76%
TechCrunch	100.00%	100.00%	100.00%
The Verge	100.00%	96.21%	97.95%
USA Today	100.00%	98.29%	99.11%

Table 8: Using CSS visual properties as features

Website	Precision	Recall	F_1
NPR	100.00%	100.00%	100.00%
Prothom Alo	100.00%	100.00%	100.00%
QQ	100.00%	100.00%	100.00%
Sina	100.00%	100.00%	100.00%
TechCrunch	94.27%	100.00%	97.03%
The Verge	100.00%	100.00%	100.00%
USA Today	100.00%	100.00%	100.00%

6.3 CSS Selectors

CSS selectors are essentially decorated tag paths. We have incorporated CSS class name in addition to the element name into the CSS selectors. These selectors have the potential to discriminate previously indistinguishable text blocks, thus allowing the classifier extract more concise template information for a given website.

The results shows that this approach produces improvement for most of the websites compared to the tag path approach, but also introduces some regressions for other. See *Table 7*. Upon closer examination, we learned that some CSS selectors can be overly unique and tied only to a particular text block. Its idiosyncrasy presents a challenge for the classification model to generalize to other text blocks.

6.4 CSS Properties

The third approach is to use all visual related CSS properties (color, font-size, font-style, line-height etc.) as features. These features are also treated as discrete features and vectorized.

The results show that similar to tag path and CSS selectors, these features work quite well in general but not ideal in some rare cases. Especially for websites that do not visually distinguish content with other parts of the webpage, these visual features alone are not sufficient. See *Table 8*.

7. CONCLUSION

We have developed a language independent pipeline to extract web content. Our pipeline collects data, labels examples, trains support vector classifier, and evaluates learned model in an automated manner. Our learning algorithm can achieve perfect labeling when trained on a single website, even for websites with multiple different templates. By analyzing features, we have found that some of our features—tag path, CSS selectors—contributed to the near perfect classification results in many websites, but they also fail in some cases. CSS visual properties work particularly well across

most websites. This reinforces the intuitive reasoning about how a person identifies the content in a website. Tag path and CSS selectors are technical details of presenting HTML documents, which often matches with human intuitive reasoning but may break in times. See *Table 2*.

8. FUTURE WORK

Sun et al.[7] showed the interesting idea of density sum in his paper. Density sum performs very well in Chaos dataset (F1 score: 96.15%) It would be interesting to see how combination of density sum and CSS features perform. But we need to refactor our code extensively to incorporate density sum, so we were not able do it in the current time frame.

9. REFERENCES

[1] Diffbot: Extract content from standard page types: articles/blog posts, front pages, image and product pages. `http://www.diffbot.com/`.

[2] Phantomjs is a headless webkit scriptable with a javascript api. `http://www.phantomjs.org/`.

[3] Readability turns any web page into a clean view for reading now or later on your computer, smartphone, or tablet. `http://www.readability.com/`.

[4] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.

[5] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. In *ACM SIGMOD Record*, pages 84–93, 2002.

[6] G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, and L. E. Moser. Extracting data records from the web using tag path clustering. In *Proceedings of the 18th international conference on World wide web*, pages 981–990, April 2009.

[7] F. Sun, D. Song, and L. Liao. Dom based content extraction via text density. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 245–254. ACM, 2011.