Proceedings of the 13th Conference on Language Resources and Evaluation (LREC 2022), pages 3461–3470
Marseille, 20-25 June 2022

STAPI: An Automatic Scraper for Extracting Iterative Title-Text Structure from Web Documents

Nan Zhang[1], Shomir Wilson[1], Prasenjit Mitra[1,2]
[1]College of Information Sciences and Technology, The Pennsylvania State University, USA
[2]L3S Research Center, Germany
{njz5124, shomir, pmitra}@psu.edu

Abstract

Formal documents often are organized into sections of text, each with a title, and extracting this structure remains an under-explored aspect of natural language processing. This iterative title-text structure is valuable data for building models for headline generation and section title generation, but there is no corpus that contains web documents annotated with titles and prose texts. Therefore, we propose the first title-text dataset on web documents that incorporates a wide variety of domains to facilitate downstream training. We also introduce **STAPI** (**S**ection **T**itle **A**nd **P**rose text **I**dentifier), a two-step system for labeling section titles and prose text in HTML documents. To filter out unrelated content like document footers, its first step involves a filter that reads HTML documents and proposes a set of textual candidates. In the second step, a typographic classifier takes the candidates from the filter and categorizes each one into one of the three pre-defined classes (title, prose text, and miscellany). We show that STAPI significantly outperforms two baseline models in terms of title-text identification. We release our dataset along with a web application to facilitate supervised and semi-supervised training in this domain.

Keywords: Iterative Title-Text Dataset, Title-Text Identification, Automatic Scraping, Information Extraction

1. Introduction

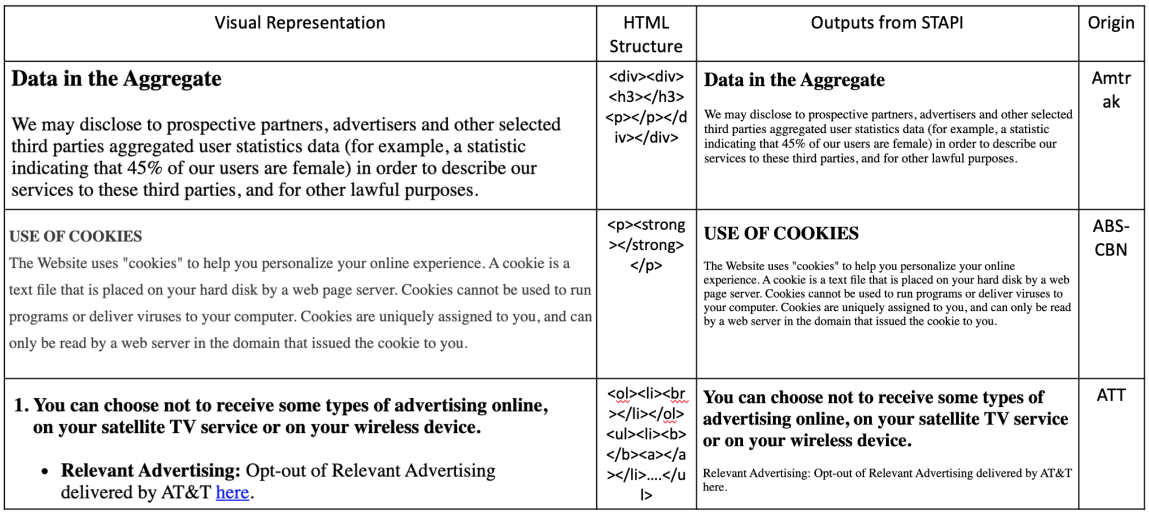
Many documents (*e.g.*, news articles, how-to guides, books, privacy policies, terms of service documents, and others) adopt an iterative title-text structure. They often separate their contents into multiple sections, and each section comes with a title. Sometimes a section is further broken into a hierarchy of subsections. The first column of Figure 1 displays typical snippets of title-text structure. This iterative title-text structure is potentially valuable for natural language generation tasks such as headline and section title generation, and question answering. For headline and section title generation, it provides labels for whether each text segment is a title or prose text in the original document. Therefore, once a large collection of documents is identified, it can facilitate rapid construction of a large-scale section title dataset. For question answering and dialogue systems, the iterative title-text structure may help a model find the target section, thus improving the model performance and reducing the search space. This structure may also facilitate topic extraction and sentiment analysis by leveraging the titles in a document.

However, extracting this iterative title-text structure from web documents is not well addressed in previous research. First of all, there is no corpus that contains web documents annotated with titles and prose texts. Moreover, our analysis about an existing approach (Gopinath et al., 2018) shows two cases where existing algorithms run into difficulties: (1) when documents do not conform to a universal HTML writing style, and (2) when irrelevant text segments like headers and footers that are not related to our title-text extraction task exist. As shown in Figure 1, visually similar title-text representations are backed up by completely different HTML structures due to diverse writing styles. Additionally, many documents may have contents like headers and footers. For example, a navigation bar can be a document header and is not relevant to the title-text structure. When the title-text extraction task incorporates these irrelevant text pieces into its input space, the original title-text structure will be disorganized. Therefore, discarding them during HTML parsing facilitates the extraction of the title-text structure.

In this paper, we introduce **STAPI**[1], which stands for **S**ection **T**itle **A**nd **P**rose text **I**dentifier. STAPI comes with a dataset about annotated web documents and a software pipeline that can be trained on our dataset for labeling section titles and prose texts in HTML documents. This system is expected to extract textual contents from HTML and benefit downstream tasks like headline generation, and the third column of Figure 1 displays its sample outputs. Adopting STAPI as the back-end model, we also develop a web application to demonstrate its functionality[2]. The key contributions of this work are summarized as follows:

- We release a dataset with labeled titles and prose texts, which is the first collection of web documents that contains title-text annotations.
- We create a novel pipeline and a web application for extracting iterative title-text structure from web documents, which mitigates the two difficulties (inconsistency of HTML writing styles and incorporation of irrelevant text segments) identified from existing literature (Gopinath et al., 2018).

[1]https://github.com/ZN1010/STAPI
[2]https://structure-extractor.ist.psu.edu



Visual Representation	HTML Structure	Outputs from STAPI	Origin
Data in the Aggregate We may disclose to prospective partners, advertisers and other selected third parties aggregated user statistics data (for example, a statistic indicating that 45% of our users are female) in order to describe our services to these third parties, and for other lawful purposes.	<div><div><h3></h3><p></p></div></div>	**Data in the Aggregate** We may disclose to prospective partners, advertisers and other selected third parties aggregated user statistics data (for example, a statistic indicating that 45% of our users are female) in order to describe our services to these third parties, and for other lawful purposes.	Amtrak
USE OF COOKIES The Website uses "cookies" to help you personalize your online experience. A cookie is a text file that is placed on your hard disk by a web page server. Cookies cannot be used to run programs or deliver viruses to your computer. Cookies are uniquely assigned to you, and can only be read by a web server in the domain that issued the cookie to you.	<p><strong></strong></p>	**USE OF COOKIES** The Website uses "cookies" to help you personalize your online experience. A cookie is a text file that is placed on your hard disk by a web page server. Cookies cannot be used to run programs or deliver viruses to your computer. Cookies are uniquely assigned to you, and can only be read by a web server in the domain that issued the cookie to you.	ABS-CBN
1. You can choose not to receive some types of advertising online, on your satellite TV service or on your wireless device. • **Relevant Advertising:** Opt-out of Relevant Advertising delivered by AT&T here.	<ol><li> </li></ol><ul><li><b></b><a></a></li>....</ul>	**You can choose not to receive some types of advertising online, on your satellite TV service or on your wireless device.** Relevant Advertising: Opt-out of Relevant Advertising delivered by AT&T here.	ATT

Figure 1: Excerpts from the website privacy policies of Amtrak, ABS-CBN, and ATT (Gopinath et al., 2018) for presenting three examples of visually similar title-text representations with different HTML structures. The left column shows the original visual representation of each example through Google Chrome browser. Each example has the title on top followed by the section paragraph. The two columns in the middle display the simplified version of HTML structure (HTML tags) for each example and outputs from STAPI. The right column shows the origin of each example.

- Our experimental results demonstrate that our proposed pipeline is more effective than baseline models in reducing irrelevant text segments and identifying title-text structure.

2. Related Work

We review current literature to see how STAPI can benefit headline and section title generation. Then, we review existing approaches for structure extraction from input documents and broadly categorize this domain into two folds: extracting structure-related information from HTML documents and from other formats such as PDFs and document images. Although this paper concentrates on extracting textual structure from HTML, we also review methods for other markup languages.

2.1. Applications of Document Structure Extraction

We review representative papers of section title and headline generation for the sake of motivation of our work. Gehrmann et al. (2019) developed a pipeline for section title generation in low-resource environment (limited training data with labels). Their pipeline incorporates a selector that chooses the most salient sentence and a compressor that adopts a Semi-Markov Conditional Random Field (Sarawagi and Cohen, 2004) to leverage word representations such as BERT (Devlin et al., 2019). Specifically, their selector was trained on the CNN/DailyMail dataset (Hermann et al., 2015). Field et al. (2020) designed various transformer decoders to achieve section title generation. Without using news-related data, their dataset consists of articles from the English Language Wikipedia. Their dataset is not publicly available. Kiyono et al. (2018) have addressed headline generation by mitigating three of the bottlenecks of previous approaches: (1) redundant generation, (2) missing important phrases, and (3) incorporation of irrelevant entities. They used the Gigaword corpus (Graff et al., 2003), which is a news-related dataset. Jin et al. (2020) proposed a novel model that generates style-specific headlines. Their model could generate humorous, romantic, and clickbait headlines. To train their model, they compiled a rich source dataset by combining the New York Times (Sandhaus, 2008) and CNN/DailyMail. Matsumaru et al. (2020) concentrated on improving the truthfulness of automatically generated headlines. Truthfulness is determined by checking whether a generated headline contains information outside of its corresponding section. They built a binary classifier to predict entailment relationship between headlines and the corresponding sections and manually annotated the entailment relationships of instances in the Gigaword corpus to train this binary classifier.

To the best of our knowledge, most headline and section title generators utilize news-related datasets like the CNN/DailyMail and Gigaword corpus. These datasets are built by collecting articles from a few large news sites such as CNN, and articles from a single site typically have consistent HTML structures. For example, we find that paragraphs (prose texts) in CNN articles share a class name called “zn-body__paragraph”. Field et al. (2020) use articles from Wikipedia that is outside of news domain and also has consistent HTML structure. In other words, extracting title-text structure from one site (or a small number of sites) is straightforward, because people just need to check every platform individually and look for its specific identifiers of section titles and prose paragraphs. However, when researchers want to train a section title generator on a dataset that contains web documents from countless platforms, they may find a hard time looking for a desired dataset, because each platform adopts unique HTML writing styles and it is costly to find structure identifiers for every platform. For instance, datasets on privacy policy such as Wilson et al. (2016) usually have documents collected from innumerable sites, since every company has its own privacy policy. Because training section title generators like Gopinath et al. (2020) requires documents collected from various sites, a competitive structure extraction tool that works for documents from any site is highly demanded. The purpose of such tool is to scale up the structure extraction work that humans can achieve. We hence propose STAPI, a fully automated tool, to locate the section titles and prose texts in web files from any sites.

2.2. Structure Extraction from HTML

Extracting iterative title-text structure in web documents is an underexplored aspect of natural language processing (NLP). Existing libraries like Beautiful Soup (Richardson, 2007) can capture the markup that visually distinguishes titles from prose, but they cannot identify which text is title and which text is prose due to inconsistently used HTML tags as shown in Figure 1. As an early piece of work of adopting machine learning algorithms to extract information from web documents, Freitag (1998) took structural and other information of the web document as input to build an extensible token-oriented feature set. This method can be adapted to tasks like identifying the title of a web page. Flesca et al. (2004) reviewed rule-based tools called wrappers (Adelberg, 1998; Baumgartner et al., 2001; Muslea et al., 2001; Crescenzi et al., 2001; Kushmerick, 2000; Laender et al., 2002) which automatically extract information from web documents, and Dalvi et al. (2011) improved wrappers by allowing them to be trained on noisy training data. Unlike rule-based systems, STAPI is flexible enough to handle web documents with different writing styles. Algur and Hiremath (2006) extracted data records (a set of data that represent a meaningful entity in a web document) from HTML based on visual clues. Similarly, YesuRaju and KiranSree (2013) proposed VIPS (**VI**sion based **P**age **S**egmentation) algorithm that transforms a web page into a visual block tree for extracting structure-related information. Visual clues are important for labeling section titles and prose paragraphs, so we build features for STAPI with the consideration of the visual appearance of each text segment.

More recently, García-Plaza et al. (2016) adopted fuzzy logic to leverage HTML markup for web page clustering. They designed 31 independent rules based on HTML markup, so their approach is still a rule-based one. Our approach also builds features based on HTML markup, but we do not have the performance of their model on title-text classification to compare with STAPI, since their approach is for document clustering task.

As mentioned in the previous section, Gopinath et al. (2018) proposed ASDUS, which stands for **A**utomatic **S**egment **D**etection using **U**nsupervised and **S**upervised learning. ASDUS is the only tool that serves as a direct competitor of STAPI. Therefore, we select it as our baseline model for further comparison. We also analyze its system output incorporated in its dataset. We find that it assigns textual content appearing before the first title of a document under an "uncategorized" title. 107 (out of 303) documents have this "uncategorized" title. The content under the "uncategorized" title contains a mix of legitimate text segments (either section title or prose paragraph) and irrelevant text segments (*e.g.*, document header). The incorporation of irrelevant text pieces in ASDUS is a shortcoming. Additionally, ASDUS is prone to errors (*e.g.*, inability to handle the anchor element in HTML) when parsing different HTML writing styles in the dataset, and its dataset does not come with labels (*e.g.*, whether a piece of text is a section title or a prose paragraph). We ameliorate all these shortcomings of ASDUS by annotating its dataset and introducing a more effective pipeline. STAPI is novel, because there does not exist a relevant pipeline that comes with annotations and adopts machine learning algorithms to filter out irrelevant text segments. With better design of features, STAPI is shown to be more effective than ASDUS (discussed in Section 5.4) in reducing irrelevant text segments and identifying title-text structure.

2.3. Structure Extraction from Other Document Formats

Other researchers have worked on structure extraction from formats besides HTML. Bentabet et al. (2020) extracted the table of contents of any searchable PDF files using the character-level convolutional neural network (Zhang et al., 2015). This deep neural network takes a sequence of characters as input, so they cannot learn from features corresponding to the visual appearance of each text segment. Therefore, we do not pursue this model, because we want our model to learn not only character-level features but also visual clues of every text segment (humans may also rely heavily on visual clues when they do title-text classification).

For scanned documents, Yang et al. (2017), Lee et al. (2019), and Barman et al. (2020) extracted semantic structure from document images by performing a pixel-wise segmentation task. Although we can also use the same method by converting web documents to scanned ones, we believe it is costly and unnecessary. So we decide to work on raw HTML documents directly. Aydin (2021) applied optical character recognition (OCR) operations to first extract textual data from document images. Then, naive Bayes (Webb, 2010) was utilized to perform document classification based on the extracted texts. Performing OCR on a document and then applying image processing techniques are also costly operations. Dang and Nguyen (2021) introduced a novel deep neural network for information extraction on the 2D character-grid embedding of a document, and their model captured the textual and spatial relations between 2D elements. Inspired by their design, our feature set incorporates spatial information between text segments.

People have done structure detection as a part of other things. Kuropiatnyk and Shinkarenko (2020) processed digital documents structure (documents represented by "doc" or "docx" files format) using templates, so their method is not flexible enough like STAPI that can handle various HTML writing styles. Bentabet et al. (2019) generated table of contents and designed 28 hand-crafted features for title detection on searchable PDF documents. Some features that we use for STAPI are also from their feature set, but our model fits better with web documents, because we complement their idea with HTML-specific features such as tag name. Mezghanni and Gargouri (2016) recognized the semantic structure of Arabic electronic documents including tiles and the headings of chapters. However, their approach cannot be used on our dataset, because it builds many language-specific rules.

Many models analyzed in this subsection are based on deep learning, so we conduct experiments between STAPI and a deep neural network-based model to show the effectiveness of our approach.

3. Methods

STAPI works in two steps. Its first step involves a filter that identifies relevant parts from the input text, and its second step involves a typographic classifier. The output of the filter will be the input of the typographic classifier. Figure 2 displays its workflow.

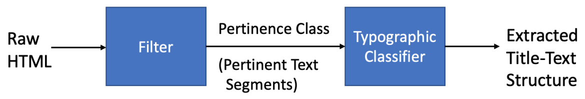


Figure 2: System diagram of STAPI.

3.1. Filter

We train a filter using XGBoost (Chen and Guestrin, 2016) to filter out irrelevant text segments in web documents like document header and footer, since our focus is on the main text in a document. The filter serves as an HTML parser. It first discards HTML elements that do not present a clear separation of textual sections such as *table*, *image*, and *button*. The iterative title-text structure does not apply in these HTML elements, and we do not lose any of the main text in the document by doing so. Then, the filter examines all the remaining HTML tags and keeps those that contain texts. We remove unreadable characters and extra whitespaces from them. A binary classifier decides which text to filter out (*irrelevance* class) and which text to keep (*pertinence* class). Since the number of items in each class is unbalanced, we adopt the SMOTE (Chawla et al., 2002) to upsample the number of training instances of the minority class. We favor oversampling over downsampling, because we do not want to lose any training data and the existing literature shows that oversampling can provide optimal results (Mohammed et al., 2020).

Our filter uses *number of words*, *relative position*, *tag name*, *parent tag name*, *next tag name in the list*, *ID of the tag*, and *average sentence encoding* as features. The average sentence encoding is a numerical representation of every text candidate. To compute it, we break each candidate into sentences, tokenize each sentence, compute sentence encoding outputs using a pre-trained BERT model ("bert-base-cased" from Transformers library of Hugging Face), and average them. We believe irrelevant text segments are more likely to be semantically different than the rest, so we adopt average sentence encoding. The average sentence encoding for each candidate has a fixed length of 768, and it is concatenated with the rest of the features to form a feature set as the input to our XGBoost model. We apply one-hot encoding to all the categorical features.

We design the feature set based on our literature review in the previous section. Tag name, next tag name in the list, and ID of the tag correspond to the visual clues of text segments. Number of words is morphological characteristic and can be treated as the syntactic feature. Relative position encodes spatial information, and average sentence encoding is the semantic feature.

3.2. Typographic Classifier

The typographic classifier takes the output from the filter as input and performs structure extraction. It tries to classify every candidate proposed by the filter into three classes: title, prose text, and miscellany. The miscellany class is a minority class and is designed for collecting all texts that are not necessarily parts of any section in semantics but are still related to the document discourse. For example, a piece of metadata like *"Last updated at Jan 29, 2021"* in a privacy policy document belongs to the miscellany class. Links that navigate within the page (*e.g.*, *"Back to top"*) are also in this class. (Note that the navigation link for redirecting to another page belong to the irrelevance class in the previous step and should be thrown away.) Instead of simply viewing the miscellany class as the irrelevance class in the previous step, we design the miscellany class to keep as many relevant texts as possible, since it may become useful for future study (*e.g.*, extracting the revision time of a web page). Under the goal of keeping as much training data as possible, we also use SMOTE to upsample the number of training instances for the classes of title and miscellany.

Type	# Documents	# Irrelevance	# Pertinence	# Title	# Prose Text	# Miscellany
Privacy Policy	144	18770	12500	2838	9233	429
Terms of Service	99	7554	9476	2142	7099	235
Miscellaneous Topics	48	5426	2363	506	1648	209

Table 1: Detailed statistics of our proposed dataset

The features that the typographic classifier uses are *number of words*, *ratio of current text length to next text length in the list*, *number of punctuation symbols*, *tag name*, *next tag name in the list*, and *average sentence encoding*. We apply the same techniques as used in the filter to encode categorical features and calculate average sentence encoding. We train the typographic classifier using XGBoost, and it generates a sanitized version of the raw HTML document that contains predicted labels of each relevant text segment. Notice that the number of punctuation symbols and the ratio of current text length to next text length in the list are morphological characteristics.

4. Dataset

As mentioned in Section 2.2, there does not exist a dataset that contains annotated web documents with the label for whether each text segment is a title or prose text. Thus, we build the first dataset for extracting the iterative title-text structure from web documents. Our dataset is based on the HTML files collected by Gopinath et al. (2018). There are three types of web documents: privacy policy, terms of service, and miscellaneous topics. The miscellaneous topics cover *news*, *sports*, *botany*, *web design*, *photography*, *data science*, *cookie policies*, *HTML*, *history*, *migraine*, *dataset*, *technical documentation*, *shoes*, *grammar*, *kids stories*, and *cricket* domains. Table 1 summarizes the key statistics of our dataset after annotation. Within a specific document type, the sum of title, prose text, and miscellany classes should equal the number of instances in the pertinence class. We see that the prose text class is always the majority class across all document types. On the other hand, the pertinence category is the majority class in terms of service documents, but it is the minority class overall. Our dataset incorporates 18 domains with a focus on privacy policies.

We annotated each web document in our dataset manually by adding a label (about title, prose text and miscellany classes) to every relevant text piece that we see through the Google Chrome browser. As elaborated above, the miscellany class contains texts such as *"Back to top"* and *"Last updated at Jan 29, 2021"*. The first author of this paper completed all the annotation job to ensure consistency, and the whole annotation process took more than 43 consecutive hours, which is a long time just for 291 web documents. We relied on visual clues through the browser to distinguish section titles from prose paragraphs. We group all the labeled text segments together and call them the gold standard. Note that this annotation work is for labels that the typographic classifier uses. In other words, the annotator only labeled the pertinence class (pertinent text segments) to improve work efficiency. Our filter will compare its preprocessed text segments with the gold standard, and those with no label assigned will belong to the irrelevance class. We deleted documents that are duplicate or have lots of hidden contents. For example, text segments are not displayed by the browser if they have "display:none" as the style attribute. When a web document has few hidden segments, the annotator still labeled them. But the web document was discarded if it contains so many hidden segments that the annotator determined the extremely high cost of locating them. Moreover, when the whole content of a web document belongs to a table, depending on the difficulty and reasonableness, the annotator either deleted the table tag in the original document to extract the content from the table or removed the entire document from our dataset. The reason is that SPAPI is not designed for labeling text segments in tables. We noted the cases when we had to modify the original web document. Whenever there are ambiguous cases, the annotator followed one rule: titles must differentiate themselves from prose text by having different visual or syntactic clues. For instance, when prose paragraphs are not numbered while the other text segments are bullet points, the annotator will assign those bullet points as titles even if all text segments share the same visual style (other than being numbered or not).

For reproducibility, we release our dataset along with the modification note and time spent annotating each document. We believe the time spent on each document is an indicator of the complexity of the document, so we keep it for future research. Specifically, we sometimes spent more than 35 minutes annotating a single web document due to its lengthy structure, which demonstrates the high costs of data annotation for HTML title-text extraction.

5. Experiments

In this section, we first discuss the construction of our baseline models. We then identify a limitation of STAPI and possibly all existing applications that involve scraping texts from HTML documents. Finally, we conduct four kinds of experiments to showcase the effectiveness of STAPI: (1) train and test STAPI on its processed text segments to check STAPI's capability to

	Weighted Precision	Weighted Recall	Macro Precision	Macro Recall
Filter	0.933	0.933	0.930	0.932
Typographic Classifier	0.968	0.967	0.893	0.872

Table 2: Precision and recall for both filter and typographic classifier of STAPI

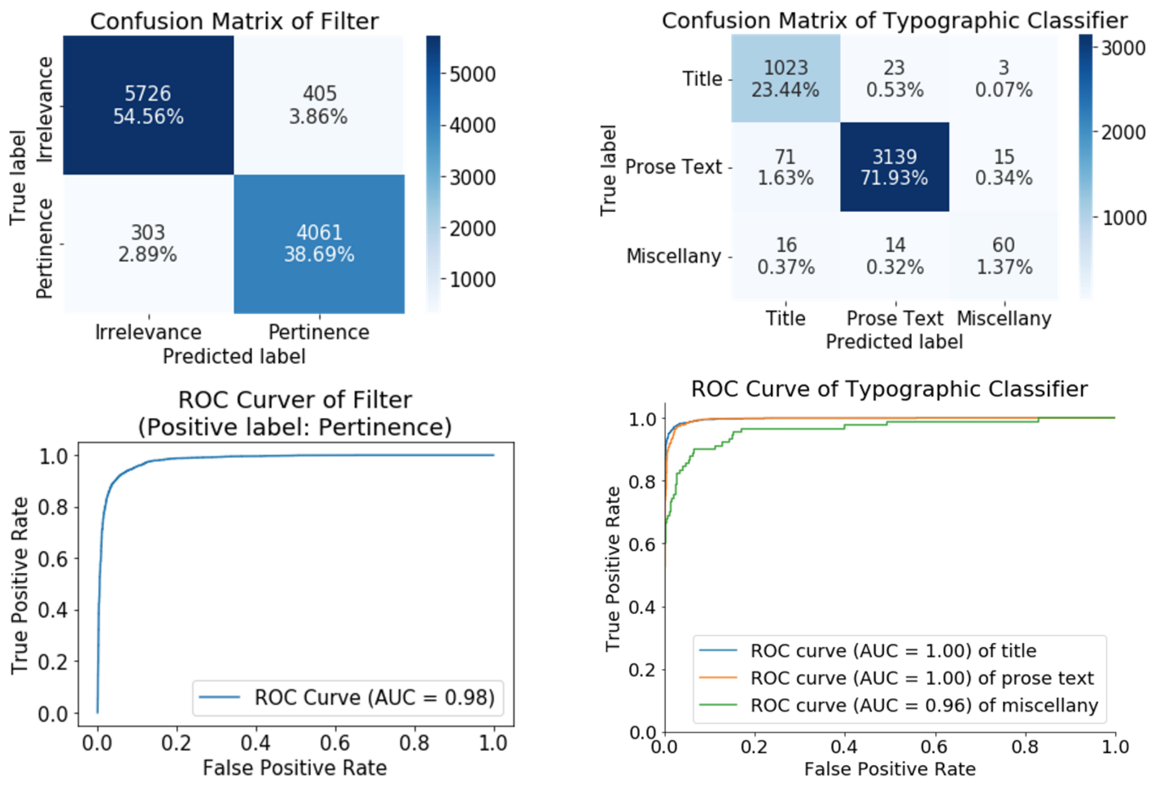


Figure 3: Confusion matrices and ROC curves for both filter and typographic classifier

predict unseen data, (2) run a baseline model to examine the necessity of STAPI's filter, (3) compare STAPI with two other baselines to see how each model performs on the title-text classification task, and (4) evaluate the importance of each feature via an ablation study.

5.1. Baseline Models

Program jusText (Pomikálek, 2011) is a competitive tool for parsing HTML documents and claims to remove boilerplate content. We select it as a baseline model to test capability of the filter of STAPI. Because jusText is purely based on heuristic, we run it directly on our test set.

Two other baseline models come from ASDUS (Gopinath et al., 2018): a domain independent model (**DI** pipeline) using K-means clustering, and a domain dependent model (**DD** pipeline) using a feed forward neural network. Because some static files of ASDUS are unavailable, we cannot run its source code. Instead, we implement both DI and DD models by ourselves based on the paper and source code of ASDUS. For the DI pipeline, scikit-learn (Pedregosa et al., 2011) is utilized to realize the K-means clustering algorithm. The system builds 8 features to capture the syntactic aspect of each text piece, and clustering is performed over these eight manually designed features. For each potential title identified in clustering, the system calculates an overlap score between its lemmatized form and the lemmatized form of the next text segment. Finally, any potential title that has an overlap score above 75% will be classified as a section title.

For the DD pipeline, a deep neural network is adopted to conduct supervised learning. We train this pipeline using our annotated data. The pipeline uses text from titles and from prose paragraphs to build two separate word embedding models via gensim (Rehurek and Sojka, 2010). It then computes two semantic relatedness scores and concatenates them with the string length of the corresponding text segment to form a feature set. We utilize scikit-learn to realize our own version of the DD pipeline. Although DD claims to classify a text piece into "title", "prose", or "unrelated", it does not define the "unrelated" class. Thus, we only consider "title" and "prose" classes when we train DD.

5.2. Limitation

We test the parsing capability of STAPI under all writing styles in our dataset. As shown in Figure 1, some HTML authors may prefer to place a text in a deeply nested organization by nesting HTML elements (*e.g.*,

Model	Privacy Policy	Terms of Service	Miscellaneous Topics	All Types Combined
jusText	63.0%	61.3%	63.7%	66.8%
Filter of STAPI	82.8%	93.2%	78.6%	90.5%

Table 3: The model performance in terms of gold standard coverage

Model	Privacy Policy	Terms of Service	Miscellaneous Topics	All Types Combined
DI	0.797	0.800	0.733	0.773
DD	0.854	0.884	0.831	0.869
STAPI	0.965	0.977	0.923	0.976

Table 4: Weighted F1 scores of models across different document types (domains)

the example from Amtrak), which poses a challenge to the HTML parsing. Others may prefer a flatter organization (*e.g.*, the examples from ABS-CBN and ATT). Since the filter of STAPI compares a raw document with its corresponding gold standard to generate labels of each text segment (described in Table 4), we set the text in the gold standard as target and compute the percentage of gold standard coverage to see how well STAPI can scrape text from raw HTML files. As a result, we find six documents where more than 50% of their gold standards are not picked up by STAPI. In other words, for these six documents, more than half of their original title-text structures will be lost in training or testing if we incorporate them in the dataset. STAPI successfully picks up 95.0% of the gold standard on average per document after removing the six documents, but this coverage shrinks to 93.7% if we maintain all the web documents in our dataset.

Analyzing the HTML source code of these six documents, we conclude that the mismatch between the visual appearance of a text segment and its corresponding source code is the reason why STAPI cannot match more than half of the gold standard in these six documents. Within a web page, a text segment that people see may be backed up by several HTML elements. For example, the example from ATT in Figure 1 has a text segment that is backed up by bold and anchor elements. Therefore, an ideal system needs to figure out the number of HTML elements to be concatenated and the time for concatenation. Since this element concatenation problem is related to HTML parsing and is not the major concern of this paper, we decide to remove these six documents from training or testing and leave this problem for future study.

5.3. Setup

We choose to perform four types of experiments. We randomly select 60% of documents for training, 20% of documents for validation, and the rest 20% for the testing set. The first experiment involves training and testing STAPI on its processed text segments from all three document types. We analyze the performance of both filter and typographic classifier. After we tune hyperparameters on the validation set, the filter ends up with a learning rate of 0.1, and the typographic classifier has a learning rate of 0.65.

The second experiment is to compare the filter of STAPI with jusText across different document types (domains) of our dataset. We seek to showcase the necessity of proposing a novel method for removing irrelevant text segments in HTML. To achieve this, we calculate the percentage of gold standard coverage for both models. The learning rate of the filter has values of 0.3, 0.25, 0.3, and 0.1 for the second, third, fourth, and fifth column of Table 3.

The third experiment tests the title-text classification capability of STAPI, DI, and DD. We seek to check how the typographic classifier of STAPI performs against DI and DD. According to the source code of ASDUS, both DI and DD adopt generic parsers to process raw HTML documents. In particular, they discard more HTML tags than STAPI does. For example, anchor tags, form tags, and list-related tags are discarded in the baselines but not in STAPI. Therefore, when different pipelines parse HTML differently, each pipeline will end up with different training and testing sets. Since HTML parsing is not the point of interest of ASDUS and the previous experiment already examines the parsing performance of STAPI, we train and test all the pipelines on the same data parsed by STAPI. In this experiment, we throw away the miscellany class in our dataset when we compare performance across different systems. The reason is that the definition of miscellany class is not defined in our baselines. We also select the weighted F1 score as the evaluation metric, because it takes label imbalance into account. As for hyperparameters, the learning rate of the typographic classifier has values of 0.3, 0.7, 0.3, and 0.3 for the second, third, fourth, and fifth column of Table 4. The number of clusters of DI is set to 2 for all the columns, and DD has (h1=4, h2=8), (h1=4, h2=8), (h1=16, h2=2), (h1=4, h2=8), and (h1=24, h2=48) for the second, third, fourth, and fifth column of Table 4.

Finally, we conduct an ablation study to showcase the different contributions of features adopted by the filter and the typographic classifier. In this study, we drop a feature and re-train our model to see how the weighted F1 score gets affected.

Dropped Feature	**Filter**	**Typographic Classifier**
Number of words	0.928	0.965
Relative position	0.932	-
Tag name	0.929	0.957
Parent tag name	0.929	-
Next tag name in the list	0.924	0.967
ID of the tag	0.932	-
Average sentence encoding	0.920	0.938
Ratio of the current text length to next text length in the list	-	0.951
Number of punctuation symbols	-	0.966
None	0.933	0.967

Table 5: Ablation study for the performance (weighted F1 score) of filter and typographic classifier

5.4. Results

The effectiveness of STAPI is showed by evaluating the performance of the filter and the typographic classifier. Table 2 summarizes their precision and recall, and Figure 3 shows their confusion matrices and ROC curves. The filter achieves 0.93 on both weighted precision and recall, and the typographic classifier achieves 0.97 on both of these metrics, which demonstrates the strong performance of STAPI. Note that the typographic classifier is low on macro recall (0.87). As indicated in the confusion matrix, macro recall assigns the minority (miscellany class) with more weight and we choose to incorporate it at the cost of a slightly lower recall value, because the AUC score of the miscellany class is still high (0.96).

Table 3 summarizes the performance of each model in terms of gold standard coverage. The filter of SATPI significantly outperforms jusText in every document type. Although there are tools available for extracting relevant text content from HTML, the performance of the filter demonstrates the necessity of designing a novel content filtering model as a part of STAPI. Both filter and jusText achieve relatively lower coverage on the documents with miscellaneous topics, which indicates the potential of mitigating the HTML element concatenation problem discussed in Section 5.2.

In Table 4, the typographic classifier of STAPI is compared against DI and DD for title-text classification. It is the best model in terms of weighted F1 score. DD relies heavily on semantic features while DI relies purely on syntactic features. In addition to a wide variety of techniques such as upsampling, the typographic classifier constructs features on both semantic and syntactic aspects. This is one of the reasons why it outperforms the baselines. Its high F1 score on all three document types combined (0.976) demonstrates the capability of STAPI as a reasonably domain-agnostic system. The DD pipeline leads DI in every column because of the advantage of supervised learning when labels are available. The superiority of STAPI and DD over DI showcases that conducting unsupervised learning for title-text classification on web documents is a suboptimal approach and some level of supervision is required for more competitive performance. Such finding reinforces our motivation and the utility of proposing an annotated dataset for extracting the iterative title-text structure from HTML documents.

In our ablation study, the importance of each feature can be treated as the performance drop that the feature causes. The more important a feature is, the more performance drop it can cause. In Table 5, the average sentence encoding is the most important feature for both filter and typographic classifier, which demonstrates the recent success of pre-trained transformer models. For the filter, next tag name in the list is also important, so we can infer that the filter we build is trying to learn the decision boundary of headers and footers. Relative position and ID of a tag are the least important ones for the filter. As for the typographic classifier, it is shown that ratio of the current text length to next text length in the list is an important feature, because a title usually has much shorter length than its following paragraph. On the other hand, number of punctuation symbols is the least important feature, which indicates that many titles have similar number of punctuation symbols as prose texts in our dataset.

6. Conclusion

We propose a novel scraper called STAPI for extracting the iterative title-text structure from web documents. STAPI mitigates the bottlenecks of previous approaches by adopting a filter and a typographic classifier. We show that STAPI is more effective at title-text classification in comparison to two existing models. We release our implementation along with our annotations and a web application to facilitate research of structure extraction from HTML.

Future research should study the element concatenation problem in web documents. In particular, when the visual appearance of a text segment is backed up by several HTML tags, a model needs to efficiently locate the corresponding HTML tags and concatenate them. Such a model will enhance the parsing ability of any HTML parsers and improve the gold standard coverage of our title-text classification pipeline. In addition, being able to identify the hierarchy of section titles (*e.g.*, titles and subtitles) will also be valuable.

7. Bibliographical References

Adelberg, B. (1998). Nodose—a tool for semi-automatically extracting structured and semistructured data from text documents. *SIGMOD Rec.*, 27(2):283–294, June.

Algur, S. P. and Hiremath, P. (2006). Extraction of flat and nested data records from web pages. In *Proceedings of the fifth Australasian conference on Data mining and analystics-Volume 61*, pages 163–168. Citeseer.

Aydin, O. (2021). Classification of documents extracted from images with optical character recognition methods.

Barman, R., Ehrmann, M., Clematide, S., Oliveira, S. A., and Kaplan, F. (2020). Combining visual and textual features for semantic segmentation of historical newspapers. *arXiv preprint arXiv:2002.06144*.

Baumgartner, R., Flesca, S., and Gottlob, G. (2001). Visual web information extraction with lixto. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, page 119–128, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Bentabet, N.-I., Juge, R., and Ferradans, S. (2019). Table-of-contents generation on contemporary documents.

Bentabet, N.-I., Juge, R., Maarouf, I., Valsamou, D., and Ferradans, S. (2020). Automatic table-of-contents generation for efficient information access. *SN Computer Science*, 1, 08.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June.

Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

Crescenzi, V., Mecca, G., and Merialdo, P. (2001). Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*.

Dalvi, N., Kumar, R., and Soliman, M. (2011). Automatic wrappers for large scale web extraction. *Proc. VLDB Endow.*, 4(4):219–230, January.

Dang, T.-A. N. and Nguyen, D.-T. (2021). End-to-end information extraction by character-level embedding and multi-stage attentional u-net. *arXiv preprint arXiv:2106.00952*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Field, A., Rothe, S., Baumgartner, S., Yu, C., and Ittycheriah, A. (2020). A generative approach to titling and clustering Wikipedia sections. In *Proceedings of the Fourth Workshop on Neural Generation and Translation*, pages 79–87, Online, July. Association for Computational Linguistics.

Flesca, S., Manco, G., Masciari, E., Rende, E., and Tagarelli, A. (2004). Web wrapper induction: A brief survey. *AI Commun.*, 17(2):57–61, April.

Freitag, D. (1998). Information extraction from html: Application of a general machine learning approach. In *AAAI/IAAI*, pages 517–523.

García-Plaza, A. P., Fresno, V., Unanue, R. M., and Zubiaga, A. (2016). Using fuzzy logic to leverage html markup for web page representation. *IEEE Transactions on Fuzzy Systems*, 25(4):919–933.

Gehrmann, S., Layne, S., and Dernoncourt, F. (2019). Improving human text comprehension through semi-Markov CRF-based neural section title generation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1677–1688, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Gopinath, A. A. M., Wilson, S., and Sadeh, N. (2018). Supervised and unsupervised methods for robust separation of section titles and prose text in web documents. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 850–855.

Gopinath, A. A. M., Kumar, V. B., Wilson, S., and Sadeh, N. (2020). Automatic section title generation to improve the readability of privacy policies.

Graff, D., Kong, J., Chen, K., and Maeda, K. (2003). English gigaword. *Linguistic Data Consortium, Philadelphia*, 4(1):34.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In C. Cortes, et al., editors, *Advances in Neural Information Processing Systems*, volume 28, pages 1693–1701. Curran Associates, Inc.

Jin, D., Jin, Z., Zhou, J. T., Orii, L., and Szolovits, P. (2020). Hooks in the headline: Learning to generate headlines with controlled styles. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5082–5093, Online, July. Association for Computational Linguistics.

Kiyono, S., Takase, S., Suzuki, J., Okazaki, N., Inui, K., and Nagata, M. (2018). Reducing odd generation from neural headline generation. In *Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation*.

Kuropiatnyk, O. and Shinkarenko, V. (2020). Text borrowings detection system for natural language structured digital documents. 04.

Kushmerick, N. (2000). Wrapper induction: Efficiency and expressiveness. *Artif. Intell.*, 118(1–2):15–68, April.

Laender, A. H. F., Ribeiro-Neto, B., Silva, A., and Teixeira, J. S. (2002). A brief survey of web data extraction tools. *SIGMOD Rec.*, 31:84–93.

Lee, J., Hayashi, H., Ohyama, W., and Uchida, S. (2019). Page segmentation using a convolutional neural network with trainable co-occurrence features. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1023–1028. IEEE.

Matsumaru, K., Takase, S., and Okazaki, N. (2020). Improving truthfulness of headline generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1335–1346.

Mezghanni, I. B. and Gargouri, F. (2016). Detecting hidden structures from arabic electronic documents: Application to the legal field. In *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 75–81.

Mohammed, R., Rawashdeh, J., and Abdullah, M. (2020). Machine learning with oversampling and undersampling techniques: Overview study and experimental results. In *2020 11th International Conference on Information and Communication Systems (ICICS)*, pages 243–248.

Muslea, I., Minton, S., and Knoblock, C. A. (2001). Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1–2):93–114, March.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Pomikálek, J. (2011). *Removing boilerplate and duplicate content from web corpora*. Ph.D. thesis, Masarykova univerzita, Fakulta informatiky.

Rehurek, R. and Sojka, P. (2010). Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 workshop on new challenges for NLP frameworks*. Citeseer.

Richardson, L. (2007). Beautiful soup documentation. *April*.

Sandhaus, E. (2008). The new york times annotated corpus. *Linguistic Data Consortium, Philadelphia*, 6(12):e26752.

Sarawagi, S. and Cohen, W. W. (2004). Semi-markov conditional random fields for information extraction. *Advances in neural information processing systems*, 17:1185–1192.

Webb, G. I., (2010). *Naïve Bayes*, pages 713–714. Springer US, Boston, MA.

Wilson, S., Schaub, F., Dara, A. A., Liu, F., Cherivirala, S., Giovanni Leon, P., Schaarup Andersen, M., Zimmeck, S., Sathyendra, K. M., Russell, N. C., Norton, T. B., Hovy, E., Reidenberg, J., and Sadeh, N. (2016). The creation and analysis of a website privacy policy corpus. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1330–1340, Berlin, Germany, August. Association for Computational Linguistics.

Yang, X., Yumer, E., Asente, P., Kraley, M., Kifer, D., and Lee Giles, C. (2017). Learning to extract semantic structure from documents using multimodal fully convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July.

YesuRaju, P. and KiranSree, P. (2013). A language independent web data extraction using vision based page segmentation algorithm. *arXiv preprint arXiv:1310.6637*.

Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. In C. Cortes, et al., editors, *Advances in Neural Information Processing Systems*, volume 28, pages 649–657. Curran Associates, Inc.