Capturing Logical Structure of Visually Structured Documents with Multimodal Transition Parser

Yuta Koreeda[12] and Christopher D. Manning[2]
[1]Hitachi America Ltd, Santa Clara, CA, USA
[2]Stanford University, Stanford, CA, USA
{koreeda, manning}@stanford.edu

Abstract

While many NLP pipelines assume raw, clean texts, many texts we encounter in the wild, including a vast majority of legal documents, are not so clean, with many of them being visually structured documents (VSDs) such as PDFs. Conventional preprocessing tools for VSDs mainly focused on word segmentation and coarse layout analysis, whereas fine-grained logical structure analysis (such as identifying paragraph boundaries and their hierarchies) of VSDs is underexplored. To that end, we proposed to formulate the task as prediction of *transition labels* between text fragments that maps the fragments to a tree, and developed a feature-based machine learning system that fuses visual, textual and semantic cues. Our system is easily customizable to different types of VSDs and it significantly outperformed baselines in identifying different structures in VSDs. For example, our system obtained a paragraph boundary detection F1 score of 0.953 which is significantly better than a popular PDF-to-text tool with an F1 score of 0.739.

1 Introduction

Despite recent motivation to utilize NLP for wider range of real world applications, most NLP papers, tasks and pipelines assume raw, clean texts. However, many texts we encounter in the wild, including a vast majority of legal documents (e.g., contracts and legal codes), are not so clean, with many of them being visually structured documents (VSDs) such as PDFs. For example, of 7.3 million text documents found in Panama Papers (which arguably approximates the distribution of data one would see in the wild), approximately 30% were PDFs[1]. Good preprocessing of VSDs is crucial in order to apply recent advances in NLP to real world applications.

Thus far, the most micro and macro extremes of VSD preprocessing have been extensively studied, such as word segmentation and layout analysis (detecting figures, body texts, etc.; Soto and Yoo, 2019; Stahl et al., 2018), respectively. While these two lines of studies allow extracting a sequence of words in the body of a document, neither of them accounts for local, logical structures such as paragraph boundaries and their hierarchies.

These structures convey important information in any domain, but they are particulary important in the legal domain. For example, Figure 1(1) shows raw text extracted from a non-disclosure agreement (NDA) in PDF format. An information extraction (IE) system must be aware of the hierarchical structure to successfully identify target information (e.g., extracting "definition of confidential information" requires understanding of hierarchy as in Figure 1(2)). Furthermore, we must utilize the logical structures to remove debris that has slipped through layout analysis ("Page 1 of 5" in this case) and other structural artifacts (such as semicolons and section numbers) for a generic NLP pipeline to work properly.

Yet, such logical structure analysis is difficult. Even the best PDF-to-text tool with a word-related error rate as low as 1.0% suffers from 17.0% newline detection error (Bast and Korzen, 2017) that is arguably the easiest form of logical structure analysis.

The goal of this study is to develop a fine-grained logical structure analysis system for VSDs. We propose a transition parser-like formulation of logical structure analysis, where we predict a *transition label* between each consecutive pair of text fragments (e.g., two fragments are in a same paragraph, or in different paragraphs of different hierarchies). Based on such formulation, we developed a feature-based machine learning system that fuses multimodal cues: visual (such as indentation and line spacing), textual (such as section num-

[1]Calculated from Obermaier et al. (2016) by regarding their *emails*, *PDFs* and *text documents* as the denominator.

Natural Legal Language Processing Workshop 2021, pages 144–154

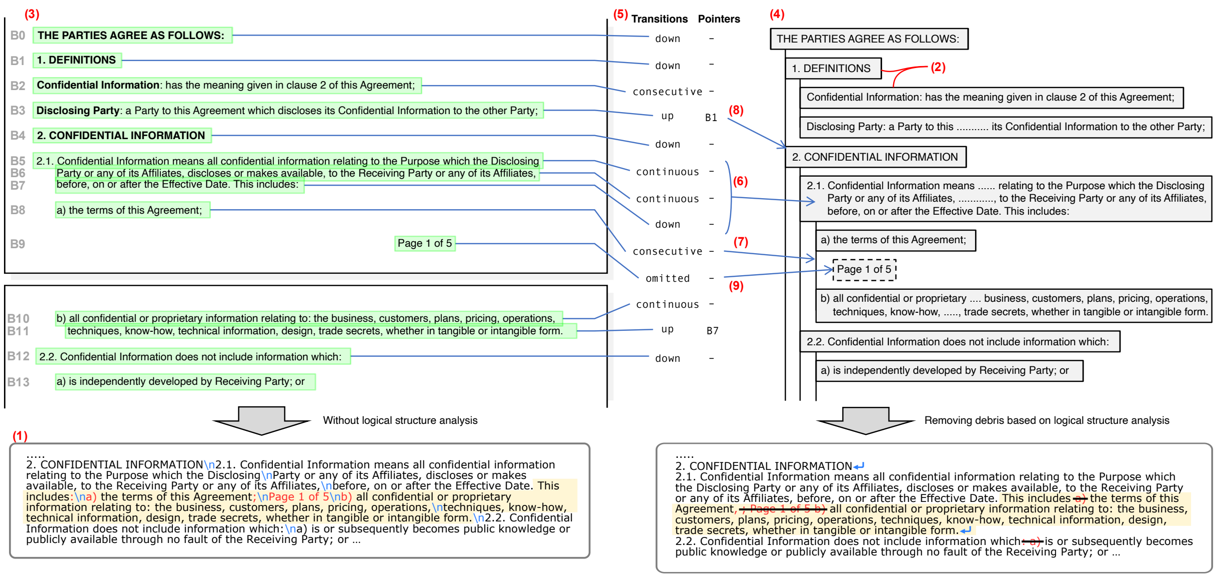


Figure 1: Overview of the logical structure analysis for VSDs and its formulation.

bering and punctuation), and semantic (such as language model coherence) cues. Finally, we show that our system is easily customizable to different types of VSDs and that it significantly outperforms baselines in identifying different structures in VSDs. For example, our system obtained a paragraph boundary detection F1 score of 0.953 that is significantly better than PDFMiner[2], a popular PDF-to-text tool, with an F1 score of 0.739. We open-sourced our system and dataset[3].

2 Problem Setting and Our Formulation

In this study, we concentrate on logical structure analysis of VSDs. The input is a sequence of text blocks (Figure 1(3)) that can be obtained by utilizing existing coarse layout analysis and word-level preprocessing tools. We aim to extract paragraphs and identify their relationships. This is equivalent to creating a tree with each block as a node (Figure 1(4)).

We propose to formulate this tree generation problem as identification of a *transition label* between each consecutive pair of blocks (Figure 1(5)) that defines their relationship in the tree. We define the transition $trans_i$ between i-th block (hereafter b_i) and b_{i+1} as one of the following:

`continuous` b_i and b_{i+1} are continuous in a single paragraph (Figure 1(6)).

`consecutive` b_{i+1} is the start of a new paragraph at the same level as b_i (Figure 1(7)).

`down` b_{i+1} is the start of a new paragraph that is a child (a lower level) of the paragraph that b_i belongs to (Figure 1(6)).

`up` b_{i+1} is the start of a new paragraph that is in a higher level than the paragraph that b_i belongs to (Figure 1(8)).

`omitted` i-th block is debris and omitted (Figure 1(9)). $trans_{i-1}$ is carried over to the relationship between b_{i-1} and b_{i+1}.

While `down` is well-defined (because we assume a tree), `up` can be ambiguous as to how many levels we should raise. To that end, we also introduce a *pointer* to each `up` block, which points at b_j whose level b_i belongs to ($ptr_i = b_j$, where $j < i$; Figure 1(8)).

3 Dataset

In this study, we target four types of VSDs in different file formats and languages:

Contract$^{pdf}_{en}$ English NDAs in PDF format.

Law$^{pdf}_{en}$ English executive orders from local authorities.

Contract$^{txt}_{en}$ English NDAs in visually structured plain text format.

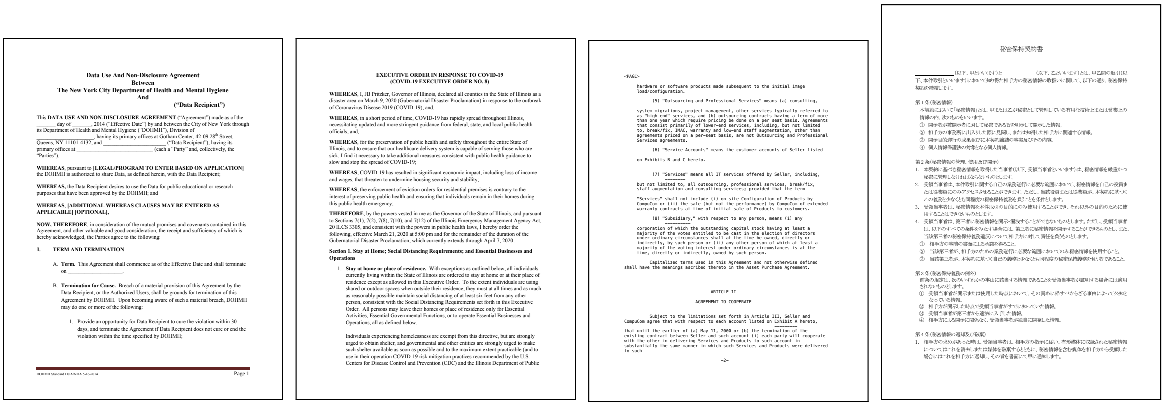
Contract$^{pdf}_{ja}$ Japanese NDAs in PDF format.

Examples of each type of VSDs are shown in Figure 2.

For PDFs, we downloaded PDFs from Google.com search result. Since our focus is not on coarse layout analysis or word-level preprocessing, we selected single column documents

[2] https://euske.github.io/pdfminer/

[3] https://github.com/stanfordnlp/pdf-struct



(a) Contract$_{en}^{pdf}$ (b) Law$_{en}^{pdf}$ (c) Contract$_{en}^{txt}$ (d) Contract$_{ja}^{pdf}$

Figure 2: Examples of VSDs in our dataset[4]

and extracted blocks with an existing software. Specifically, we utilized PDFMiner and extracted each `LTTextLine`, which roughly corresponds to each line of text, as a block. We merged multiple `LTTextLines` where `LTTextLines` are vertically overlapping.

For plain texts, we searched documents filed at EDGAR[5]. We simply used each non-blank line of a plain text as a block.

We annotated all documents by hand. We describe more details of the data collection and annotation in Appendix A.1.

The data statistics are given in Table 1. While the number of documents is somewhat limited, we note that each document comes with many text blocks and evaluations were stable. Furthermore, it was enough to reliably show the difference between our system and baselines in our experiments.

	Contract$_{en}^{pdf}$	Law$_{en}^{pdf}$	Contract$_{en}^{txt}$	Contract$_{ja}^{pdf}$
Format	PDF	PDF	Text	PDF
Language	English	English	English	Japanese
#Documents	40	40	22	40
#Text blocks	137.9	165.9	142.0	73.7
Max. depth	3.4	3.9	3.1	3.0
#`continuous`	95.4 (68%)	110.6 (67%)	109.9 (77%)	33.9 (44%)
#`consecutive`	20.3 (17%)	30.8 (20%)	15.3 (12%)	14.9 (20%)
#`up`	8.5 (6%)	7.1 (4%)	4.8 (3%)	11.0 (15%)
#`down`	9.4 (6%)	9.9 (6%)	4.6 (3%)	12.1 (17%)
#`omitted`	4.4 (3%)	7.6 (3%)	7.4 (4%)	1.8 (2%)

A number in the second set of rows indicates an average count over documents. A percentage represents an average ratio of each label.

Table 1: Dataset information

4 Proposed System

4.1 Transition Parser

In this work, we propose to employ handcrafted features and a machine learning-based classifier as the transition parser. This strategy is more suited to our task than utilizing deep learning because (1) we can incorporate visual, textual and semantic cues, and (2) it only requires a small number of training data which is critical in the legal domain where most data is proprietary.

For each block, our parser extracts features from a context of four blocks and performs multi-class classification over the five transition labels. Since `omitted` changes targets of transition, we also omit `omitted` blocks in feature extraction. For $trans_i \neq$ `omitted`, we extract features from $[b_{i-1}, b_i, b_j, b_{j+1}]$ where b_j is the first block after b_i with $trans_j \neq$ `omitted`. For $trans_i =$ `omitted`, we extract features from $[b_{i-1}, b_i, b_{i+1}, b_{i+2}]$. At test time, since we need to know the presence of `omitted` before feature extraction, we run a first pass of predictions to identify blocks with `omitted`, then use that information to dynamically extract features to identify other labels.

Our system can be customized to different types of documents by modifying the features. We have designed a feature set for each document type by visually inspecting the training dataset (Table 2). For Contract$_{en}^{txt}$, we regarded space characters as horizontal spacing and blank lines as vertical spacing, which allowed us to define features that are

[4](a) http://www.astho.org/Programs/Infectious-Disease/Healthcare-Associated-Infections/Electronic-Health-Records/Toolkit/Data-Use-Agreement-New-York-City/, (b) https://www2.illinois.gov/IISNews/21288-Gov._Pritzker_Stay_at_Home_Order.pdf, (c) https://www.sec.gov/Archives/edgar/data/86115/0000930661/0000930661-99-001321-index.htm, and (d) http://www.septima.co.jp/contracts/27_himitsuhoji.pdf

[5]https://www.sec.gov/edgar.shtml

Description		Blocks	Document type		
			Contract$_{en}^{pdf}$ /Law$_{en}^{pdf}$	Contract$_{en}^{txt}$	Contract$_{fa}^{pdf}$
Visual features					
V1	Indentation (up, down or same)	1-2, 2-3	✓	✓	✓
V2	Indentation after erasing numbering	1-2, 2-3		✓	
V3	Centered	2, 3	✓	✓	✓
V4	Line break before right margin*	1, 2	✓	✓	✓
V5	Page change	1-2, 2-3	✓		✓
V6	Within top 15% of a page	2	✓		✓
V7	Within bottom 15% of a page	2	✓		✓
V8	Larger line spacing*	1-2, 2-3	✓	✓	✓
V9	Justified with spaces in middle	2, 3	✓	✓	✓
V10	Similar text in a similar position*	2	✓		✓
V11	Emphasis by spaces between characters	1, 2			✓
V12	Emphasis by parentheses	1, 2			✓
Textual features					
T1	Numbering transition*	2	✓	✓	✓
T2	Punctuated	1, 2	✓	✓	✓
T3	List start (`/[-;:,]$/`)	1, 2	✓	✓	✓
T4	List elements (`/(;|,|and|or)$/`)	2	✓	✓	
T5	Page number (strict)	1, 2, 3	✓	✓	✓
T6	Page number (tolerant)	1, 2, 3	✓	✓	✓
T7	Starts with "whereas"	3	✓	✓	
T8	Starts with "now, therefore"	3	✓	✓	
T9	Dictionary-like (includes ":" & not V4)	2, 3	✓	✓	
T10	All capital	2, 3	✓	✓	
T11	Contiguous blank field (underbars)	1-2, 2-3	✓	✓	✓
T12	Horizontal line ("*-=#%_+" only)	1, 2, 3		✓	
Semantic features					
S1	Language model coherence*	1-2-3	✓	✓	✓

The "Blocks" columns list blocks used to extract features for $trans_2$ (e.g. "1-2, 2-3" means $[b_{i-1}, b_i]$ and $[b_i, b_{i+1}]$ are used to extract two sets of features). Features with a similar intended functionality are assigned the same feature name and implementations may vary for different document types. *: Explained in detail in Section 4.1.

Table 2: List of features for each feature extractor

analogous to those for PDFs.

While readers can reference our open-sourced code for the concrete implementation, we will discuss some of the features that have important implementation details. For a target block b_i:

Numbering transition (T1) A categorical feature that itself is a heuristic transition parser. It identifies a numbering in each block and keeps a memory of the largest numberings by their types (i.e., its alphanumeric type and styling, such as `IV.` and `(a)`). It outputs (1) `continuous` if no numbering is found, (2) `consecutive` if the numbering in b_{i+1} is contiguous to the numbering in b_i, (3) `up` if not consecutive and there is a corresponding number in the memory, and (4) `down` if it is none of above and it is the first number in its numbering type. For example, B0 in Figure 1 is `down` as `1.` is the first numbering type that it sees and "1" will be added to the memory. B1 and B2 are `continuous` as no numbering is found and B3 is `consecutive` as a number "2" is found in the same type as `1.`. B4 is `down` as it contains a new numbering type.

Language model coherence (S1) To determine if b_i should be classified as `omitted`, it utilizes language model to classify whether it is more natural to have b_i or b_{i+1} after b_{i-1}. Specifically, we use GPT-2 (Radford et al., 2019) to calculate language model loss $\ell(i, i-1)$ for b_i given b_{i-1} as a context (i.e., fed into the model but not used in the loss calculation). We then calculate $\ell(i, i-1) - \ell(i+1, i-1)$ as the feature. If it is more coherent to have b_i after b_{i-1}, $\ell(i, i-1)$ will be smaller than $\ell(i+1, i-1)$ and the feature value will be negative. We also utilize $\ell(i+1, i) - \ell(i+1, i-1)$.

Similar text in similar position (V10) Headers and footers tend to appear in similar positions across different pages with similar texts. For example, a contract may have the contract's title on every pages at the same position. This feature is 1 if there exists a block b_j such that blocks' overlapping area is larger than 50% of their bounding box (treating as if they are on the same page), and their edit distance is small[6].

Line break before right margin (V4) A Boolean feature that is 0 if the block spans to the right margin and 1 otherwise (i.e., breaks before the right margin). To distinguish the body and the margin of the document, we

[6] $d(b_i, b_j)/\max(len(b_i), len(b_j)) < 0.1$, where d gives the Levenshtein distance and len gives the length of text.

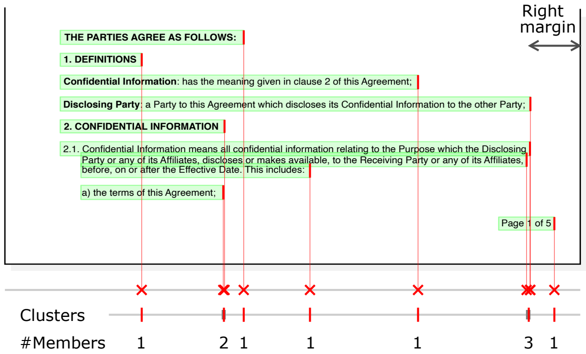


Figure 3: A sketch of how we determine right margin. We apply 1D clustering on the right positions of the blocks and choose the rightmost cluster with at least a user-defined number of members. If we choose to have a minimum of two members, the right margin would be the cluster with three members.

```python
class PDFFeatureExtractor(BaseFeatureExtractor):
    def __init__(self, text_boxes):
        bboxes = np.array(
            [tb.bbox for tb in text_boxes])
        page_top = bboxes[:, 3].max()
        page_bottom = bboxes[:, 1].min()
        self.header_thresh = \
            page_top - 0.15 * (page_top - page_bottom)
        ...

    @single_input_feature([1])
    def header_region(self, tb):
        return tb.bbox[3] > self.header_thresh

    @pairwise_feature([(0, 1), (1, 2)])
    def page_change(self, tb1, tb2):
        if tb1 is None or tb2 is None:
            return True
        return tb1.page != tb2.page

    @pointer_feature()
    def pointer_left_aligned(
            self, head_tb, tb1, tb2, tb3):
        return self.left_aligned(tb1)

    ...
```

Figure 4: The Python implmenetation of a feature extractor

apply 1D clustering[7] on the right positions of the blocks and extract the rightmost cluster with minimum members of six per page (to ignore headers/footers) as the right margin (Figure 3). This margin information is used in other features (V3, V6 and V7).

Larger line spacing (V8) A Boolean feature that is 0 if line spacing is *normal* and 1 otherwise. To determine the *normal* line spacing, we apply 1D clustering on line spacings and pick a cluster with the largest number of members.

4.2 Pointer Identification

We also implement the pointer identification with handcrafted features and a machine learning-based classifier. Since a down transition creates a new level that a block can point back to, we extract all pairs of $[b_j, b_i]$ ($b_j \in \mathcal{C}_i$) with $trans_i = \texttt{up}$, $trans_j = \texttt{down}$ and $j < i$. We then extract features from $[b_j, b_i]$ and train a binary classifier to predict $p\,(ptr_i = b_j | b_j, b_i)$. In training, we use ground truth down labels to extract candidates $\mathcal{C}_i$. At test time, we aggregate $\mathcal{C}_i$ from predicted transition labels and predict the pointer by $ptr_i = \text{argmax}_{b_j \in \mathcal{C}_i}\, p\,(ptr_i = b_j | b_j, b_i)$.

While our pointer points at a block with down (b_j), it is sometimes important to extract features from the first block in the paragraph that b_j belongs to, which we will hereafter refer as $b_{head(j)}$. Using $b_{head(j)}$, we extract the following features from $[b_j, b_i]$:

Consecutive numbering Boolean features on whether a numbering in b_i is contiguous to a numbering in b_j and $b_{head(j)}$, respectively.

Indentation Categorical features on whether indentation gets larger, smaller or stays the same from b_j to b_i and from $b_{head(j)}$ to b_{i+1}, respectively.

Left aligned Binary features on whether b_j, b_{i+1} and $b_{head(j)}$ are left aligned, respectively.

Transition counts We count numbers of blocks $\{b_k\}_{j<k<i}$ with down and with up, respectively. We use these two numbers along with their difference as features. This is based on an intuition that a closer block with down tends to be more important.

Pointer features are also customizable, but we used the same features[8] for all the document types.

While we call our system a "transition parser", we do not employ a stack and instead employ the graph-based parser-like formulation for the pointer identification. We selected this strategy because of the recent success of graph-based parsers (Dozat and Manning, 2017; Zhang et al., 2019).

5 Implementation and Customization

In this section, we briefly describe the implementation of our system that allows easy customization to different types of VSDs. Our system employs modular and customizable design and is implemented in Python. A user may implement a new feature extractor simply by writing a new fea-

[7] We utilized a naïve 1D clustering, where it greedily adds elements from a sorted list to a cluster while the maximum difference of the elements is within a user-defined threshold.

[8] More precisely, the pointer features are implemented slightly different for different document types, such as numbering being modified to Japanese for Contract$^{pdf}_{ja}$, but they are intended to have similar functionalities.

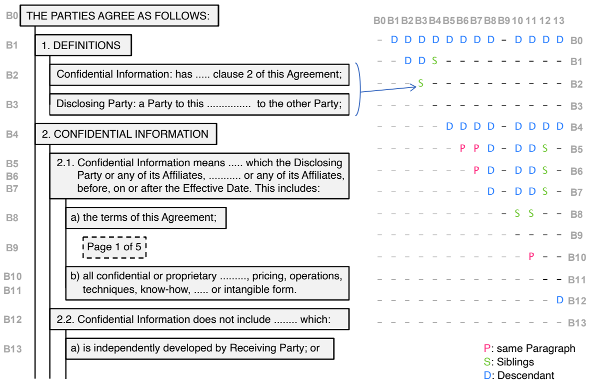


Figure 5: Evaluation from IE perspective. For each of ground truth and predicted trees, we extract a relationship matrix (right) that describes all the pairwise relationships and calculate F1 scores/accuracy by comparing the matrices.

ture extractor class where each feature is implemented as its class function (Figure 4). For example, `@single_input_feature([1])` denotes that the subsequent function should be applied to the second block of each context (thus corresponding to feature V6). Likewise, the features for pointer identification can be implemented by marking a function with `@pointer_feature()`, which takes a candidate block b_j (`tb1`), a target block b_i (`tb2`), the block next to the target block b_{i+1} (`tb3`) and $b_{head(j)}$ (`head_tb`) as an input.

A feature extractor object is instantiated for each document where all feature functions are automatically aggregated to produce the feature vector. A new feature extractor can inherit from an existing feature extractor (e.g., feature extractors for Contract$^{pdf}_{en}$ and Contract$^{pdf}_{ja}$ both inherit from a base PDF feature extractor), which makes it easy to reuse implementations.

6 Experiments

6.1 Evaluation Metrics

While we do report transition prediction accuracy, it is not a true task metric since it is rooted on our formulation of the task. Looking back at our initial motivation in Section 1, we introduce two sets of evaluation metrics.

The first set of metrics is rooted on IE perspective. For IE, it is important to identify ancestor-descendant and sibling relationships because it allows, for example, identifying a subject (in an ancestral block) and its objects (a decendant block and its siblings). Thus, we evaluate F1 scores for identifying pairs of blocks in (1) same paragraph, (2) sibling, and (3) ancestor-descendant relationships, respectively (Figure 5). Note that we do not include cousin blocks in the sibling relationship, because it is not clear whether cousin blocks have any meaningful information in the context of IE.

We use the second set of metrics to evaluate a system's efficacy as a preprocessing tool for more general NLP pipelines. We evaluate paragraph boundary identification metrics since paragraph boundaries can be used to determine appropriate chunks of text to be fed into the NLP pipelines. We also report accuracy for removing debris with `omitted`.

We used five-folds cross validation for the evaluation.

6.2 Baselines

We compared our system against the following baselines:

Numbering baseline (Hatsutori et al., 2017) This baseline detects numberings using a set of regular expressions and identifies dropping in hierarchy when the type of numberings has changed. Adopting Hatsutori et al. (2017) to our problem formulation, our implementation is the same as the feature "numbering transition (T1)."

Visual baseline This baseline relies purely on visual cues; i.e., indentation and line spacing. For each pair of consecutive blocks, this baseline outputs (1) `continuous` when indentation does not change and line spacing is normal (as in feature V8), (2) `consecutive` when indentation does not change and line spacing is larger than normal, (3) `down` when indentation gets larger, and (4) `up` when indentation gets smaller. On `up`, it points back at the closest block with the same indentation.

PDFMiner We use this popular open-source project to detect paragraph boundaries as in Bast and Korzen (2017). PDFMiner relies purely on geometric heuristics to detect paragraph breaks.

6.3 Implementation Details

We used Random Forest (Breiman, 2001) as the transition and pointer classifiers, which is suited for categorical features that occupy the majority of our features. We did not tune hyperparameters of the Random Forest classifier and used default values of scikit-learn (Pedregosa et al., 2011).

For language model coherence feature S1, we

Relationship			Contract$_{en}^{pdf}$			Law$_{en}^{pdf}$			Contract$_{en}^{txt}$			Contract$_{ja}^{pdf}$		
			Visual	Number	Ours	Visual	Number	Ours	Visual	Number	Ours	Visual	Number	Ours
Same paragraph	Micro	P	**0.982**	0.484	0.944	**0.891**	0.219	0.858	**0.993**	0.540	0.983	0.446	0.402	**0.973**
		R	0.683	0.947	**0.951**	0.681	**0.969**	0.957	0.708	0.917	**0.978**	0.552	**0.985**	0.966
		F	0.806	0.641	**0.947**	0.772	0.357	**0.905**	0.826	0.680	**0.980**	0.494	0.571	**0.969**
	Macro	P	**0.980**	0.644	0.955	0.906	0.328	**0.936**	**0.990**	0.595	0.969	0.481	0.478	**0.971**
		R	0.670	**0.966**	0.951	0.634	**0.974**	0.951	0.746	0.934	**0.976**	0.527	**0.985**	0.956
		F	0.782	0.736	**0.948**	0.731	0.452	**0.933**	0.847	0.687	**0.971**	0.450	0.617	**0.955**
Siblings	Micro	P	0.332	0.677	**0.841**	0.430	0.647	**0.849**	0.397	0.780	**0.784**	0.106	0.151	**0.699**
		R	0.323	**0.765**	0.736	0.283	0.504	**0.712**	0.481	**0.763**	0.723	0.506	0.571	**0.691**
		F	0.328	0.718	**0.785**	0.341	0.567	**0.774**	0.435	**0.772**	0.752	0.176	0.238	**0.695**
	Macro	P	0.443	0.678	**0.791**	0.598	0.493	**0.793**	0.482	0.677	**0.814**	0.347	0.237	**0.719**
		R	0.427	0.691	**0.751**	0.417	0.379	**0.696**	0.557	0.603	**0.758**	0.506	0.536	**0.663**
		F	0.337	0.650	**0.748**	0.410	0.385	**0.724**	0.435	0.605	**0.754**	0.292	0.283	**0.671**
Descendants	Micro	P	0.381	0.184	**0.502**	**0.627**	0.132	0.456	0.239	0.190	**0.541**	0.536	0.125	**0.577**
		R	0.123	**0.879**	0.807	0.303	**0.881**	0.858	0.048	**0.888**	0.771	0.340	0.580	**0.826**
		F	0.186	0.304	**0.619**	0.409	0.229	**0.596**	0.080	0.313	**0.635**	0.416	0.205	**0.679**
	Macro	P	0.295	0.242	**0.655**	0.438	0.173	**0.581**	0.193	0.269	**0.639**	0.462	0.122	**0.737**
		R	0.194	**0.848**	0.798	0.314	0.764	**0.837**	0.072	**0.859**	0.735	0.358	0.519	**0.834**
		F	0.203	0.340	**0.641**	0.327	0.230	**0.617**	0.096	0.367	**0.625**	0.372	0.195	**0.739**
Accuracy	Micro		0.772	0.778	**0.914**	0.827	0.685	**0.908**	0.587	0.674	**0.828**	0.618	0.623	**0.940**
	Macro		0.686	0.679	**0.889**	0.732	0.427	**0.840**	0.571	0.580	**0.841**	0.623	0.492	**0.899**
Average F1	Micro		0.440	0.555	**0.784**	0.507	0.384	**0.758**	0.447	0.588	**0.789**	0.362	0.338	**0.781**
	Macro		0.441	0.576	**0.779**	0.489	0.356	**0.758**	0.459	0.553	**0.783**	0.372	0.365	**0.788**

"Micro": Micro-average, "Macro": Macro-average, "P": Precision, "R": Recall, "F": F1 score

Table 3: Results for evaluation on IE perspective

Criteria			Contract$_{en}^{pdf}$				Law$_{en}^{pdf}$				Contract$_{en}^{txt}$			Contract$_{ja}^{pdf}$			
			PDFMiner	Visual	Number	Ours	PDFMiner	Visual	Number	Ours	Visual	Number	Ours	PDFMiner	Visual	Number	Ours
Paragraph boundary	Micro	P	0.672	0.563	0.914	**0.958**	0.546	0.536	0.911	**0.948**	0.465	0.783	**0.955**	0.531	0.603	0.961	**0.970**
		R	0.822	**0.968**	0.700	0.948	0.858	0.916	0.637	**0.948**	**0.989**	0.637	0.945	0.850	0.663	0.627	**0.991**
		F	0.739	0.712	0.793	**0.953**	0.667	0.676	0.750	**0.948**	0.633	0.702	**0.950**	0.653	0.632	0.759	**0.980**
	Macro	P	0.698	0.598	0.921	**0.958**	0.632	0.565	0.866	**0.946**	0.527	0.840	**0.953**	0.585	0.645	0.964	**0.970**
		R	0.798	**0.964**	0.703	0.945	0.874	0.930	0.522	**0.943**	**0.984**	0.633	0.944	0.867	0.653	0.624	**0.988**
		F	0.722	0.729	0.772	**0.947**	0.703	0.692	0.620	**0.940**	0.673	0.693	**0.947**	0.661	0.627	0.745	**0.976**
Block elimination	Micro	P	—	—	—	0.969	—	—	—	0.979	—	—	0.865	—	—	—	1.000
		R	—	—	—	0.897	—	—	—	0.755	—	—	0.914	—	—	—	0.849
		F	—	—	—	0.932	—	—	—	0.852	—	—	0.889	—	—	—	0.919
	Macro	P	—	—	—	0.948	—	—	—	0.929	—	—	0.815	—	—	—	0.929
		R	—	—	—	0.906	—	—	—	0.858	—	—	0.816	—	—	—	0.866
		F	—	—	—	0.913	—	—	—	0.874	—	—	0.800	—	—	—	0.888

"Micro": Micro-average, "Macro": Macro-average, "P": Precision, "R": Recall, "F": F1 score

Table 4: Results for evaluation on preprocessing perspective

used GPT-2 medium[9] for English documents and `japanese-gpt2-medium`[10]) for Japanese documents.

6.4 Results

Structure and preprocessing evaluations are shown on Table 3 and Table 4, respectively. Our system obtained micro-average structure prediction accuracy of 0.914 for Contract$_{en}^{pdf}$, 0.908 for Law$_{en}^{pdf}$, 0.828 for Contract$_{en}^{txt}$ and 0.940 for Contract$_{ja}^{pdf}$, significantly outperforming the best baselines with 0.778, 0.827, 0.674 and 0.623, respectively. Our system performed the best with respect to F1 scores for all but one structure relationships.

[9]https://huggingface.co/gpt2

[10]https://huggingface.co/rinna/japanese-gpt2-medium

The difference was even more significant for paragraph boundary detection. For Contract$_{en}^{pdf}$, our system obtained a micro-average paragraph boundary detection F1 score of 0.953 that is significantly better than PDFMiner with an F1 score of 0.739. PDFMiner performed on par with our visual baseline and generally performed worse than our numbering baseline. This shows the importance of incorporating textual information to preprocess VSDs.

Micro-average transition label prediction accuracies were 0.951 (Contract$_{en}^{pdf}$), 0.938 (Law$_{en}^{pdf}$), 0.955 (Contract$_{en}^{txt}$) and 0.923 (Contract$_{ja}^{pdf}$).

We investigated the importance of each feature with greedy forward selection and greedy backward elimination of the features (Table 5). We can observe that our system makes a balanced use of

#	Contract$_{en}^{pdf}$				Law$_{en}^{pdf}$				Contract$_{en}^{txt}$				Contract$_{ja}^{pdf}$			
	Forward		Backward		Forward		Backward		Forward		Backward		Forward		Backward	
	All	(0.914)	All	(0.914)	All	(0.908)	All	(0.908)	All	(0.828)	All	(0.828)	All	(0.940)	All	(0.940)
1	T1, 2	(0.763)	T1, 2	(0.855)	T1, 2	(0.685)	T1, 2	(0.854)	V8, 2-3	(0.333)	T2, 2	(0.820)	T1, 2	(0.596)	T1, 2	(0.934)
2	V10, 2	(0.796)	T10, 3	(0.794)	V8, 2-3	(0.883)	V10, 2	(0.859)	T10, 2	(0.465)	V9, 2	(0.811)	V12, 2	(0.686)	V9, 2	(0.909)
3	T10, 3	(0.818)	T10, 2	(0.794)	V10, 2	(0.893)	T2, 2	(0.836)	T9, 2	(0.716)	T3, 2	(0.805)	V1, 2-3	(0.821)	V8, 2-3	(0.882)
4	T7, 3	(0.853)	V1, 2-3	(0.796)	V8, 1-2	(0.885)	V8, 2-3	(0.800)	T3, 2	(0.727)	V5	(0.785)	T2, 2	(0.813)	S1‡, 2	(0.865)
5	T10, 2	(0.813)	S1‡, 2	(0.808)	V5, 2-3	(0.858)	V1, 2-3	(0.747)	T6, 2	(0.721)	T10, 2	(0.781)	V8, 2-3	(0.887)	T2, 2	(0.856)
6	V1, 2-3	(0.844)	T8, 2-3	(0.801)	T7, 3	(0.881)	V4, 2-3	(0.716)	T4, 2	(0.723)	V8, 2-3	(0.752)	V9, 3	(0.906)	V4, 2	(0.824)
7	V4, 2	(0.868)	T2, 2	(0.774)	V1, 2-3	(0.898)	V9, 2	(0.676)	T2, 2	(0.721)	S1†, 2	(0.749)	T2, 1	(0.913)	V1, 2-3	(0.787)
8	T2, 2	(0.886)	V4, 2-3	(0.692)	V3, 2	(0.904)	S1†, 2	(0.717)	V4, 2	(0.722)	V9, 3	(0.751)	V4, 2-3	(0.926)	V9, 2	(0.799)

Numbers in parentheses show micro-average transition label prediction accuracy. The first line shows the results with all features.
†: $\ell(i, i-1) - \ell(i+1, i-1)$ variant. ‡: $\ell(i+1, i) - \ell(i+1, i-1)$ variant.

Table 5: Eight most important features chosen by greedy forward selection and backward elimination.

the visual and textual cues. "Indentation (V1)", "Larger line spacing (V8)" and "numbering hierarchy (T1)", which partially represent the baselines, were ranked high in many cases. At the same time, other features such as "all capital (T10)" and "punctuated (T2)" were also contributing significantly to the accuracy, which made our system much superior to the baselines.

The feature importance revealed that the semantic cue (S1) was no more important than other cues. We suspect that the feature (which compares whether adjacent or non-adjacent block is more likely given a context) had fallen back to mere language model with the context being ignored in some cases, possibly due to GPT-2 not being fine-tuned on the legal domain.

We also conducted a qualitative error analysis. For Contract$_{en}^{pdf}$, we found that our system was performing poorly on documents where they had bold or underlined section titles, followed by paragraphs without any indentation (predicted `continuous` instead of `down`). We believe incorporating typographic features would improve our system as implied by the success of the "all capital (T10)" feature.

For Contract$_{en}^{txt}$, we found that blocks that are all capitals or are all underbars were misclassified as `omitted`. All capital words and underbars are frequently used to denote headers and footers, but they were used as section titles and input fields in these examples. Unlike for Contract$_{en}^{pdf}$, we attribute this problem to lack of training data, as those should have been classified correctly with other features (such as T4 and T8) if the system had seen similar patterns in the training data.

Interestingly, we observed that the system tends to do better in documents that are hierarchically more complex. This may be because hierarchically complex documents tend to incorporate more cues to support humans comprehend the documents.

7 Related Works

As discussed in Section 1, previous works mainly focused on word segmentation and layout analysis, whereas fine-grained logical structure analysis of VSDs is less addressed. Nevertheless, there exist some studies that focus on similar goals.

Abreu et al. (2019) and Ferrés et al. (2018) have tried to deal with logical structure analysis by identifying specific structures in VSDs such as subheadings. However, these studies are too coarse-grained and cannot handle paragraph-level logical structure, thus they are unable to satisfy the need we have discussed in Section 1. FinSBD-3 shared task (Au et al., 2021) is more fine-grained than those works and incorporates extraction of list items. However, its main focus is not on analysis of logical structures; it has only four static levels for list hierarchies and does not consider hierarchies in non-list paragraphs.

Hatsutori et al. (2017) proposed a rule-based system that purely relies on numberings. We compared our system against it in Section 6 and showed that our system, which also incorporates textual and semantic cues, is superior to their method.

Sporleder and Lapata (2004) proposed a paragraph boundary detection method for plain texts that purely relies on textual and semantic cues. While their method is not intended for VSDs, some of their ideas could be incorporated to our work as additional features. We leave use of more advanced semantic cues for a future work.

While the goal is different, our textual features have some similarity to those used in sentence boundary detection (Gillick, 2009). Since our goal is to predict structures as well as boundaries, we employ richer textual and visual features that they do not utilize.

LayoutLM (Xu et al., 2020, 2021) incorporates multimodal self-supervised learning to utilize deep learning for form understanding. While it may alleviate the need for a large training dataset, it is not trivial to adopt the same method for logical structure analysis as text blocks would not fit onto the LayoutLM's context. Furthermore, it is easier to diagnose and to improve our system as it utilizes a combination of hand-crafted features, while deep learning systems tend to be completely black box.

8 Conclusions

We proposed a transition parser-like formulation of the logical structure analysis of VSDs and developed a feature-based machine learning system that fuses visual, textual and semantic cues. Our system significantly outperformed baselines and an existing open-source software on different types of VSDs. The experiment revealed that incorporating both the visual and textual cues is crucial in successfully conducting logical structure analysis of VSDs. As a future work, we will incorporate typographic and more advanced semantic cues.

Acknowledgements

We used computational resource of AI Bridging Cloud Infrastructure (ABCI) provided by the National Institute of Advanced Industrial Science and Technology (AIST) for the experiments.

References

Carla Abreu, Henrique Cardoso, and Eugénio Oliveira. 2019. FinDSE@FinTOC-2019 Shared Task. In *Proceedings of the Second Financial Narrative Processing Workshop*.

Willy Au, Abderrahim Ait-Azzi, and Juyeon Kang. 2021. FinSBD-2021: The 3rd Shared Task on Structure Boundary Detection in Unstructured Text in the Financial Domain. In *Companion Proceedings of the Web Conference 2021*.

Hannah Bast and Claudius Korzen. 2017. A Benchmark and Evaluation for Text Extraction from PDF. In *2017 ACM/IEEE Joint Conference on Digital Libraries*.

Leo Breiman. 2001. Random Forests. *Machine Learning*, 45(1):5–32.

Timothy Dozat and Christopher D. Manning. 2017. Deep Biaffine Attention for Neural Dependency Parsing. In *5th International Conference on Learning Representations*.

Daniel Ferrés, Horacio Saggion, Francesco Ronzano, and Àlex Bravo. 2018. PDFdigest: an Adaptable Layout-Aware PDF-to-XML Textual Content Extractor for Scientific Articles. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*.

Dan Gillick. 2009. Sentence Boundary Detection and the Problem with the U.S. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Yoichi Hatsutori, Katsumasa Yoshikawa, and Haruki Imai. 2017. Estimating Legal Document Structure by Considering Style Information and Table of Contents. In *New Frontiers in Artificial Intelligence*, pages 270–283. Springer International Publishing.

Frederik Obermaier, Bastian Obermayer, Vanessa Wormer, and Wolfgang Jaschensky. 2016. About the Panama Papers. *Süddeutsche Zeitung*.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. *OpenAI blog*, 1(8):9.

Carlos Soto and Shinjae Yoo. 2019. Visual Detection with Context for Document Layout Analysis. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*.

Caroline Sporleder and Mirella Lapata. 2004. Automatic Paragraph Identification: A Study across Languages and Domains. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*.

Christopher Stahl, Steven Young, Drahomira Herrmannova, Robert Patton, and Jack Wells. 2018. DeepPDF: A Deep Learning Approach to Extracting Text from PDFs. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*.

The U.S. Securities and Exchange Commission. 2018. *EDGAR® Public Dissemination Service Technical Specification*.

Yang Xu, Yiheng Xu, Tengchao Lv, Lei Cui, Furu Wei, Guoxin Wang, Yijuan Lu, Dinei Florencio, Cha Zhang, Wanxiang Che, Min Zhang, and Lidong Zhou. 2021. LayoutLMv2: Multi-modal pre-training for visually-rich document understanding.

In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*.

Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. 2020. Layoutlm: Pre-training of text and layout for document image understanding. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019. AMR Parsing as Sequence-to-Graph Transduction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

A Appendix

A.1 Details of Data Collection and Annotation

In this section, we provide supplemental information regarding the data collection and the annotation discussed in Section 3.

For PDFs, we queried Google search engines and downloaded the PDF files that the search engines returned. We used the following queries and the domains:

Contract$^{pdf}_{en}$ "“non-disclosure” agreement filetype:pdf" on seven domains from countries where English is widely spoken (US ".com", UK ".co.uk", Australia ".com.au", New Zealand ".co.nz", Singapore ".com.sg", Canada ".ca", South Africa ".co.za").

Law$^{pdf}_{en}$ "site:*.gov "order" filetype:pdf" on "google.com".

Contract$^{pdf}_{ja}$ "“秘密保持契約書” filetype:pdf" on "google.co.jp".

For the collection of Contract$^{txt}_{en}$, we first download all the documents filed at EDGAR from 1996 to 2020 in a form of daily archives[11]. We uncompressed each archive and deserialized files using regular expressions by referencing to the EDGAR specifications(The U.S. Securities and Exchange Commission, 2018), which gave us 12,851,835 filings each of which contains multiple documents. We then extracted NDA candidates from the documents by a rule-based filtering. Using meta-data obtained during the deserialization, we extracted documents whose file type starts with "EX" (denotes that it is an exhibit), its file extension is one of ".pdf", ".PDF", ".txt", ".TXT", ".html", ".HTML", ".htm" or "HTM", and its content is matched by a regular expression "(?<![a-zA-Z;"()]␣*)([Nn]on[-␣][Dd]isclosure)|(NON[-␣]DISCLOSURE)".

We then randomly selected documents that fulfill following criteria:

- it is an NDA or an executive order,
- it has embedded texts (for PDFs),
- it is a single column document, and
- a similar document is not yet in the dataset.

The last criterion mainly targets contracts from same organizations and executive orders from same authorities. It ensures that we get a wide variety of documents in our dataset.

The datasets were annotated by one of the authors. We did not employ majority vote to improve annotation consistency, because labels can be easily determined by a brief inspection of the document.

[11] `https://www.sec.gov/Archives/edgar/Oldloads/`