Capturing Logical Structure of Visually Structured Documents with Multimodal Transition Parser

Yuta Koreeda$^{1,2}$ 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 particularly 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-

\footnote{\textsuperscript{1}Calculated from Obermaier et al. (2016) by regarding their emails, PDFs and text documents as the denominator.}
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 \(\text{trans}_i\) 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(8)).
- **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)). \(\text{trans}_{i-1}\) 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 (\(\text{ptr}_i = b_j\), where \(j < i\); Figure 1(8)).

3 Dataset

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

- **Contract**\(^\text{en}\) English NDAs in different file formats and languages:
- **Law**\(^\text{en}\) English executive orders from local authorities.
- **Contract**\(^\text{ja}\) Japanese NDAs in visually structured plain text 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.
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®. 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.

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 ≠ 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 ≠ 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 pdf, we regarded space characters as horizontal spacing and blank lines as vertical spacing, which allowed us to define features that are specific...

![Figure 2: Examples of VSDs in our dataset](https://example.com/figure2.png)

| Format | Contract pdf | Law pdf | Contract txt | Contract jpg |
|--------|-------------|---------|-------------|-------------|
| Language | #Documents | PDF | English | 40 | PDF | English | 40 | Text | English | 22 | PDF | Japanese | 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%) | 30.6 (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

---

4.2 ATetadata Classification

For metadata classification, we divided the documents into subsets of general contracts and legal documents. The purpose here is to demonstrate that our approach can also be applied to different types of legal documents. We use a similar setup as in the transition parsing task and achieved comparable results.
analogs 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 numbering in the memory, 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..

**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

\[ d(b_i, b_j) / \max (\text{len}(b_i), \text{len}(b_j)) < 0.1 \]

where $d$ gives the Levenshtein distance and $\text{len}$ gives the length of text.

---

### Table 2: List of features for each feature extractor

| Description                  | Blocks  | Contract$_{pdf}$ | Contract$_{txt}$ | Contract$_{pdf}$ |
|------------------------------|---------|------------------|------------------|------------------|
| Visual features              |         |                  |                  |                  |
| V1 Indentation (up, down same) | 1-2, 2-3 | ✓                | ✓                | ✓                |
| V2 Indentation after 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.
apply 1D clustering on the right positions of the blocks and extract 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.

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 \([b_j, b_j] \in C_i\) with \(trans_i = \text{down}\) and \(j < i\). We then extract features from \([b_j, b_j]\) 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 \(C_i\). At test time, we aggregate \(C_i\) from predicted transition labels and predict the pointer by \(ptr_i = \arg\max_{b_j \in C_i} p(ptr_i = b_j|b_j, b_i)\).

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 \(b_{\text{head}}(j)\). Using \(b_{\text{head}}(j)\), we extract the following features from \([b_j, b_j]\):

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

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

Left aligned Binary features on whether \(b_j\) and \(b_{\text{head}}(j)\) are left aligned, respectively.

Transition counts We count numbers of blocks \(b_k\) 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 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 feature extractor class. For example, the PDFFeatureExtractor class is designed to be subclassed and implemented as follows:

```python
class PDFFeatureExtractor(BaseFeatureExtractor):
    def __init__(self, text_boxes):
        self.page = text_boxes
        self.header_thresh = self.page[0].bbox[3] > self.header_thresh
        @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, self, tbl, tbl2):
            if tbl is None or tbl2 is None:
                return True
            return True != False
        @pointer_feature()
        def pointer_left_aligned(self, self, head_tb, tbl1, tbl2):
            return self.left_aligned(tbl1)
            ...
```

More precisely, the pointer features are implemented slightly different for different document types, such as numbering being modified to Japanese for Contract pdf, but they are intended to have similar functionalities.
The parties agree as follows:

1. Definitions

Disclosing Party: a Party to this Agreement who discloses Confidential Information to the other Party;

Confidential Information: has the meaning set forth in clause 2 of this Agreement;

Confidential Information means any data or information which the Disclosing Party treats with confidentiality

2.1. Confidential Information does not include information which:

a) is independently developed by Receiving Party; or

b) all confidential or proprietary information, pricing, operations, financial and accounting data, and related business

2.2. Confidential Information does not include information which:

a) the terms of this Agreement;

b) is independently developed by Receiving Party;

c) is or becomes known to the Receiving Party from any other

d) which:

a) is or becomes known to the Receiving Party from any other

6.1 Evaluation Metrics

Experiments

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 identified 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.
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\(^{pdf}\), 0.908 for Law\(^{pdf}\), 0.828 for Contract\(^{xml}\) and 0.940 for Contract\(^{ja}\), 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.

The difference was even more significant for paragraph boundary detection. For Contract\(^{pdf}\), 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\(^{pdf}\)), 0.938 (Law\(^{pdf}\)), 0.955 (Contract\(^{xml}\)) and 0.923 (Contract\(^{ja}\)).

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...
Larger line spacing (V8) and numbering hierarchy were used as section titles and input fields in these documents. We expected paragraphs to be distinct from the baseline features, without any indentation (predicted continuous) without any indentation. 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 Contractpdf, 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 Contracttxt, 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 Contractpdf, 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 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.
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: Pretraining 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.
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**
```
“‘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**
```
“site:*gov “order” filetype:pdf” on “google.com”.
```

**Contract**
```
“秘密保持契約書” filetype:pdf” on “google.co.jp”.
```

For the collection of Contract, we first download all the documents filed at EDGAR from 1996 to 2020 in a form of daily archives. 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.