Supervised Identification of Participant Slots in Contracts

Dan Simonson
BlackBoiler
Syracuse, NY, USA
dan.simonson@blackboiler.com

Abstract

This paper presents a technique for the identification of participant slots in English language contracts. Taking inspiration from unsupervised slot extraction techniques, the system presented here uses a supervised approach to identify terms used to refer to a genre-specific slot in novel contracts. We evaluate the system in multiple feature configurations to demonstrate that the best performing system in both genres of contracts omits the exact mention form from consideration—even though such mention forms are often the name of the slot under consideration—and is instead based solely on the dependency label and parent; in other words, a more reliable quantification of a party's role in a contract is found in what they do rather than what they are named.

1 Introduction

Contracts create a relationship between two parties, generally centered around some kind of stance object. For example, a non-disclosure agreement (NDA) often features a disclosing party, a receiving party, and some sort of confidential information; similarly, a services agreement may feature a service provider, a service recipient, and some sort of services to be provided. In both types of contracts, the agreement will refer to itself as well.

Contracts contain language that is more repetitive than those typically studied in NLP research (Simonson et al., 2019), but this is not necessarily *consistent* language. A *disclosing party* may be referred to as a "Disclosing Party," "Discloser," "Providing Party," "Company," "Owner," "Client," or any of a limitless set of possible referring expressions, including the actual company name of the *disclosing party*. Further, some of these expressions have no particular relation to either party and may be used ambiguously throughout the corpus–i.e. "Company" may refer to the disclosing party in one contract and the receiving party in another. Thus, while enumeration of such expressions may seem to present a viable heuristic for slot identification in contract language, a more robust technique is required.

We avoid neural network methods for this task for a few reasons. Results from Borchmann et al. (2020) suggest that pre-trained models trained on more general language struggle on the idiosyncrasies of contract language—at least those pre-trained on non-legal data. Further, a resource-intensive slot detection system is not scalable in the desired environment. This is a subprocess of a larger system (Simonson et al., 2020), and thus the footprint in terms of memory and hardware expense need to remain low. We also want to build a system that would provide some transparency into its decisions so users may have an opportunity to customize the system to their needs and preferences. Given that such an understanding of neural network methods remains an on-going research task, a method derived from older techniques promises better opportunities for inspection than techniques that may be more popular.

For this, we look toward more linguistically-informed methods for identifying participant slots in language generally. Schank and Abelson (1977)'s model of script learning from episodic knowledge is a sensible theoretical fit for learning participant slots from contracts. Just as scripts are presumably learned from exposure to repetitive sequences of events, contracts themselves of particular types are also repetitive series of propositions; an individual reading many contracts would readily observe patterns that would generalize into interchangeable participant slots and relationships among them. Chambers and Jurafsky (2008, 2009) present a model for doing exactly that; however, their script learning method is unsupervised, while the sort of slots identified in this work have been prescribed in advance by subject matter experts. Thus, in this model, we turn the techniques pre-

Natural Legal Language Processing Workshop 2021, pages 163–171

sented by Chambers and Jurafsky (2008) on their head to learn participant slots from supervised data.

Section (2) discusses prior work related to contract language and slot identification, particularly how this work exists at the boundary between two threads of thought on slot identification. Section (3) outlines our annotation efforts, their results, and other NLP pre-processing necessary to extract slots. Section (4) describes the mathematical underpinnings of our slot identification technique, and Section (5) enumerates how those mathematical relations are turned into slot selections. Section (6) describes the results of these experiments, and Section (7) goes into more qualitative detail in understanding the causes of the quantitative results.

2 Prior Work

Prior NLP work on contract language is extensive, including summarization (Manor and Li, 2019; Keymanesh et al., 2020) information extraction and understanding (Anish et al., 2019; Borchmann et al., 2020; Agarwal et al., 2021), as well as corpus studies looking at intrinsic properties of contracts (Curtotti and McCreath, 2011; Simonson et al., 2019) or providing new annotations over contract language (Funaki et al., 2020).

We were not able to identify any prior work specifically on participant slots in contracts. Ash et al. (2020) devised a system that identifies the obligations of parties in contracts, leveraging similar features as in this paper but with some key differences. Participants are determined based entirely on their mention form—either overtly, or through a dictionary of synonyms. Mention forms that fall outside of these are ignored, and this technique was not overtly evaluated. As their primary goal was to identify obligations and to look at large volumes of statistics about those obligations within a corpus of employment contracts, the mention form approximation is sensible to make. On the other hand, the goal of the system presented in this paper is to identify mention forms specifically. Since this is part of a user-facing contract review system, ignoring forms that could not be determined in advance would fail to meet the needs of that system; as will be discussed in Section (3.1), such unpredictable mention forms occur frequently.

The theoretical foundation for *slots* comes from Schank and Abelson (1977)'s seminal work on *scripts*—generalizations of events from prior exposure to them. The classic example is that of a "restaurant script"—an understanding of the general series of events and participants involved in visiting a restaurant. While such a conception of knowledge is intuitive, the precise nature of *how* this knowledge should be realized in NLP systems falls into roughly two threads of thought: supervised—where such knowledge is explicitly enumerated—and unsupervised—where such knowledge is inferred from a corpus of text.

While not exactly the same as scripts, some approaches attempt to model *frames* in a supervised manner—that is, rather than inferring them from natural language descriptions in corpora, projects such as *FrameNet* overtly name and describe structured frames and their participant slots (Baker et al., 1998).

However, frames are not scripts or schemas; rather, frames are the building blocks scripts and schemas, which can be thought of as networks of frames with shared participants. Even after years of annotation, the exhaustive enumeration of all frames seems unlikely. In attempting to understand language generally, a system may need to "possess many schema[s], perhaps hundreds of thousands" (Mooney and DeJong, 1985), which motivates an unsupervised approach to extracting such structures. Chambers and Jurafsky (2008) re-ignited interest in related tasks, though most community efforts have centered around performance improvements on or variants of the narrative cloze task (Jans et al., 2012; Pichotta and Mooney, 2014; Mostafazadeh et al., 2016). Other work has continued to focus on the extraction of schemas or scripts as discrete knowledge structures (Chambers and Jurafsky, 2009; Balasubramanian et al., 2013; Simonson and Davis, 2018; Weber et al., 2018).

Latently, in the unsupervised learning of such schemas, participant slots are learned through the identification of sequences of events; in supervised techniques, slots are identified overtly and must be mapped on to text by some means. However, the unique properties of contracts allow us to execute at a hybrid approach. The system described in this paper has mathematical underpinnings that are very similar to those in unsupervised work, particularly Chambers and Jurafsky (2008). However, the overt annotation of a specific genre of contract is much more similar to a frame-based supervised approach, with a subject matter expert deciding in advance what the frame is and what the slots to be labelled are. A specific frame defines each genre

of contract, composed of the relationship between the parties specified therein. Table (1) enumerates this explicitly, juxtaposing the terminology used in our own annotation and that used in FrameNet. In an NDA, that frame is FrameNet's *Exchange* while a Services agreement is a *Commercial_transaction*. These differ because FrameNet requires as a core part of a *Commercial_transaction* a *Money* slot which is not often the case in an NDA.

Table 1: Comparison between this paper's analysis of contract genres' slots and their respective FrameNet terminology.

Contract Terminology	FrameNet
NDA	Exchange
receiving party	Exchanger_1
disclosing party	Exchanger_2
confidential information	Themes
Services agreement	Commercial_transaction
service provider	Seller
service receiver	Buyer
services	Goods

3 Data

This section describes two components of our data preparation for slot extraction: the annotation which produced the system's training data (Section 3.1) and and the other NLP pre-processing steps needed to learn and extract participant slots (Section 3.2).

3.1 Annotation

Given the consistency of particular types of contracts, the annotators indicated the mention forms of each of four slots for every document. These were, for NDAs: the *receiving party*, *disclosing party*, *confidential information*, and *agreement*; for Services: *service provider*, *service receiver*, *services*, and *agreement*. Annotators indicated the values for each of these, excluding any determiners. Each defined term had to appear at least twice to be annotated. Ambiguities were noted as well, if they occurred more than once, as some contracts deviate from the defined terms from time-to-time. The annotations are case sensitive; consistent capitalization is important in leveraging the slots.

To train annotators and specify guidelines, the annotators completed two rounds of doubly-annotated documents. After that point, further annotation was completed independently by the trained annotators, according to the guidelines. Kappa scores for different slots were between 0.80 and 1.0, where strict equivalence was the standard. Most disagreements were over the scope of the defined term—e.g. including a determiner in the mention form—rather than the term itself or were due to the omission of an ambiguous mention form.

In total, the annotation team completed 712 NDAs and 214 Services agreements. Figure (1) shows in squarified treemaps the distribution of terms, where area is proportional to the token counts of a given type. Some slots vary more than others, but none are completely invariant. The smallest cells in each treemap are single counts of a particular token type. These are most often proper nouns, generally the name of the company either disclosing information or providing services. If all of these singleton mention forms were grouped as a single type, they would be the second most frequent *disclosing party* type among the NDAs and the most frequent *service provider* type among Services agreements.

3.2 Pre-Processing

For word and sentence tokenization, lemmatization, and dependency parsing, the system uses SpaCy (Honnibal and Johnson, 2015).[1] While SpaCy was not explicitly trained on contract language, for our purposes, the labels are close enough to correct—or at least are consistent enough to derive suitable results. Our observations showed most of its errors being related to either long range dependencies–e.g. relative clauses and coordination between very long VPs–or lexical items unique to contract language–e.g. "warrant" as a verb or "receiving" as part of an NP. The latter case is directly related to our target problem–it was mostly circumvented because cases where "Receiving Party" was capitalized were correctly tagged as proper nouns.

For coreference resolution, the system employs a heuristic technique. Since contract language tends to avoid 3rd person anaphora (Simonson et al., 2019), the system avoids some of the more challenging coreference problems and targets primarily noun phrases headed by common nouns and proper nouns. First, all 1st and 2nd person pronouns are joined into chains unique to each pronoun since their intrinsic deictic properties make them unambiguous. Then the technique builds chains out of all noun phrases that contain the same sequence of lemmas. This sequence is considered the *surface form* of the chain. A noun phrase, in this case, is the regular sequence of part-of-speech tags used in

[1] SpaCy 2.2.3 with en_core_web_sm, version 2.2.5.

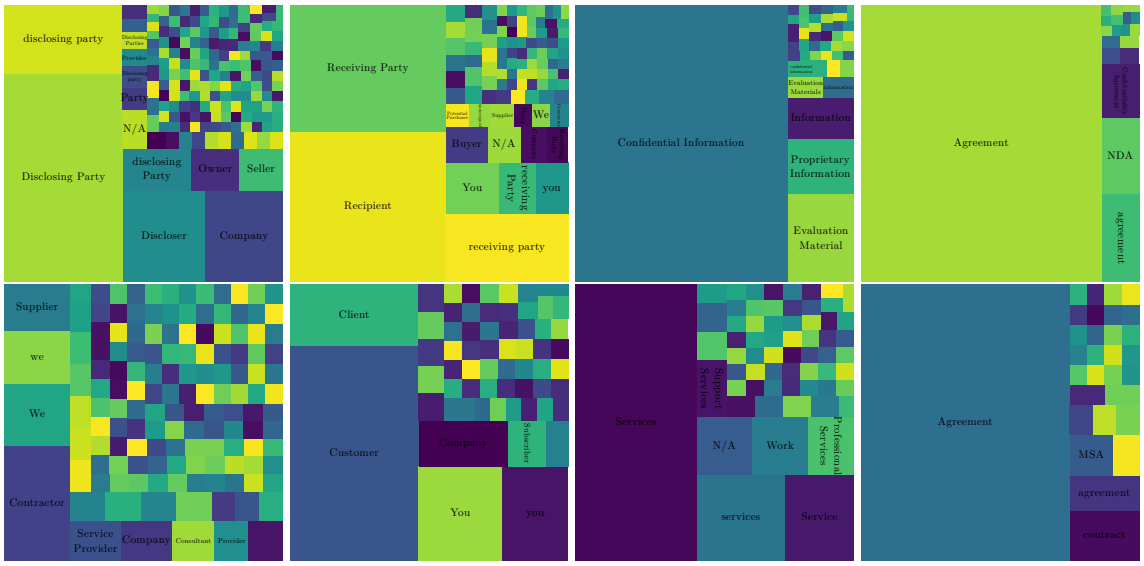

Figure 1: Squarified treemap of the contract slot annotations. The first row are NDA annotations: *disclosing party*, *receiving party*, *confidential information*, *agreement*. The second row are Services annotations: *service provider*, *service receiver*, *services*, *agreement*. Labels are not included on rare items.

Justeson and Katz (1995) but modified to exclude prepositions. A second pass is made where the coreference system checks noun phrases that were not joined with any others into a chain for subsequences of tokens that match the surface form of the chain. In these cases, the markable is joined into the chain.

4 Slot Scoring

To decide which slot a particular chain might represent requires a technique for scoring each chain C with respect to a particular slot type k. This section describes the components of a function S which does exactly that.

Rather than considering all co-referring argument pairs in a chain, as Chambers and Jurafsky (2009) did to assemble schemas, the system considers each link in a chain individually against each possible slot type. Chambers and Jurafsky (2009) were using an unsupervised technique, and thus did not have the extra information of a particular identified slot type; in other words, the slot type was implied by a strong signal between co-referring event arguments. Instead, this system associates the supervised slot class and features derived from each chain link.

During training, each chain is then represented by tuples composed of the slot type k, the mention surface form m, the dependency relation r between the mention and its parent p, which is also represented by its surface form—i.e. $\langle k, m, r, p \rangle$. In the case of a mention's immediate parent being a preposition—in a manner similar to the Stanford typed dependencies' "collapsed dependencies" (de Marneffe et al., 2006)—the relation is derived from the lemma of the preposition (e.g. "prep_in"); the parent is then the parent of the preposition.

For example, from a document where "Recipient" was annotated as *Receiving Party* and "Sensitive Documents" was annotated as *Confidential Information*, "The Recipient shall destroy all copies of the Sensitive Documents" renders three tuples:

- $\langle$*RP, Recipient, subj, destroy*$\rangle$,
- $\langle$*none, copy, dobj, destroy*$\rangle$, and
- $\langle$*DP, Sensitive Documents, prep_of, copy*$\rangle$.

The "none" class retains features to estimate independent probabilities.

Deviating from Chambers and Jurafsky (2008, 2009)'s goals for slot extraction, the system skips the constraint of looking strictly at verbs to extract such relationships. While contracts may tell stories, they are not themselves intended as narrative texts. Rather, it extracts $\langle k, m, r, p \rangle$-tuples for every mention of an entity, regardless of what r is.

In both training and slot identification, the system excludes any chains whose mention form is either "party" or "third party." This is because the meanings of these nouns are often conditioned on determiners and adjuncts with qualities that require

more sophisticated handling of the coreference information entailed within them—e.g. "each party" vs "the party." These terms rarely point specifically and unambiguously to one party or the other, and thus do not help to determine the way a slot is referred to in a given text.

From there, the system estimates probabilities as:

$$P(t) = \frac{\text{counts}(t) + \alpha}{\sum_{\tau \in T} \text{counts}(\tau) + \alpha|T|} \quad (1)$$

where $\text{counts}(t)$ is the number of times feature tuple t appeared in the corpus, T is the set of all tuples of features observed for that particular model of the type comparable to that tuple, and α is a constant smoothing parameter.

Permutations of three features determine configurations of the system—in other words, which tuples are counted and included in the set T. This paper refers to each configuration by the feature tuple of the joint probability—e.g. the model that uses the relation r and parent form p features is referred to as the $P(k, r, p)$ model, as it yields the joint probability of a particular class k with the other two features.

Pointwise mutual information (pmi) of a tuple T is defined as:

$$pmi(t) = \log \frac{P(t)}{\prod_{i=0}^{|t|} P(t_i)} \quad (2)$$

The exact contents of T and, consequentially, the probabilities multiplied in the denominator will depend on the specific probability we are using. For example, for the $P(k, r, p)$ model, pmi is realized as:

$$pmi(k, r, p) = \log \frac{P(k, r, p)}{P(k)P(r)P(p)} \quad (3)$$

To prevent rare items from being weighted highly—often a flaw of pointwise mutual information—the system forces PMI to be 0.0 for joint items with counts less than six.

When observing new items in a new document, for a given chain C, the system measures how well it fits with a particular class k using S:

$$S(k, C) = \left(\sum_{f \in C} pmi(\langle k, f_0, ..., f_n \rangle) \right) - |C| \quad (4)$$

where $|C|$ is the length of the chain C. Since the system does not know the class yet, the extracted chains are composed of tuples $\langle m, r, p \rangle$. Thus, S evaluates the pmi of each link of the chain as if it is a member of k. Subtracting the length of the chain—i.e. $-|C|$—acts to penalize longer chains.

5 Slot Selection

A score alone is not enough to decide which chains correlate with which slot. To decide which chains actually represent each slot, the system employs Algorithm (1). This is composed of roughly two parts: a score for every chain as every possible slot and a selection procedure. The first loop scores every chain's association with every class; if a score is 0.0 or less, it is omitted from further consideration. These are sorted from highest score to lowest. The second loop, in the specified reverse score order, assigns chains to particular slots—only if the chain has not yet been assigned to a slot or the slot has not yet been assigned a chain. If a slot has no assignments made to it, it is left blank.

Algorithm 1: Slot selection over a document.

```
Data: coreference chains R extracted from
      a document as a sequence of
      ⟨m, r, p⟩ tuples
Result: assignment of a chain to each slot
scores = [] for slot k ∈ slots do
    for chain C_i ∈ coreference chains R do
        if S(k, C) > 0.0 then
            scores.append(⟨S(k, C), k, i⟩)
scores.sort()
scores.reverse()
slot_selections = { }
for score tuple ⟨s, k, i⟩ in scores do
    if k ∈ slot_selections.keys then
        #skip
    else if i ∈ slot_selections.values then
        #skip
    else
        slot_selections[k] = i
    if slot_selections.keys == slots then
        break
return slot_selections
```

Figure (2) shows an example of Algorithm (1) in practice. The table on the left of Figure (2) represents the full array of scores; these are stored and sorted, resulting in the list of 3-tuples in the center of the figure. The second loop of Algorithm (1) traverses these in order, assigning each chain to

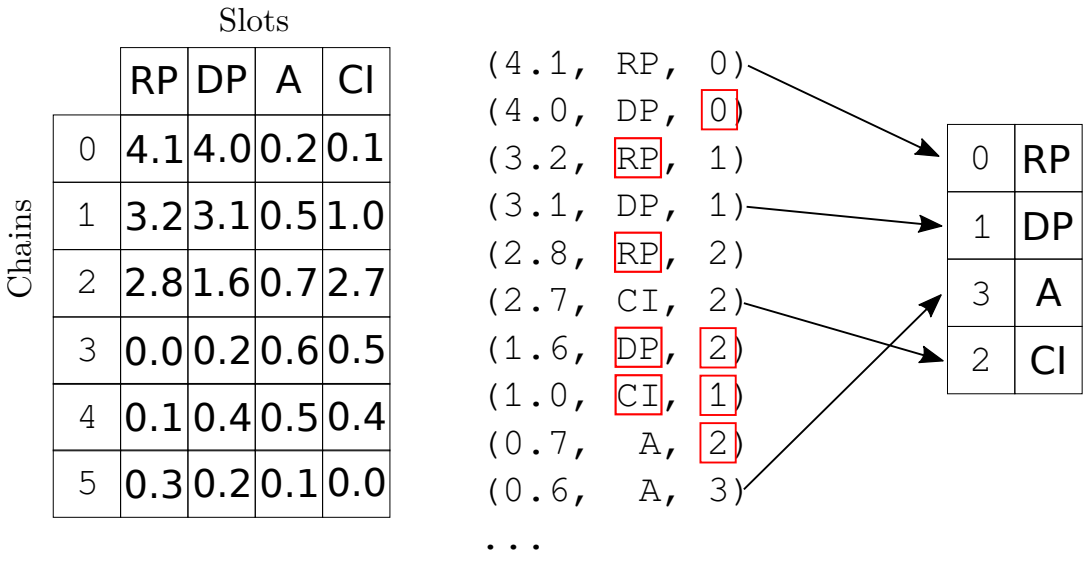

Figure 2: Illustration of an Example of Algorithm (1)

a slot if no constraints interfere. These are illustrated with boxes highlighting cause of interference. For example, the second tuple gives a high score to Chain 0 for the *disclosing party* slot; however, Chain 0 was already selected for *receiving party*, so this tuple is ignored. The next tuple—third from the top—gives a high score to Chain 1 for the *receiving party* slot, but *receiving party* has already been assigned Chain 0, so this tuple is also ignored. Next, the fourth tuple in the sequence assigns a high score for *disclosing party* to Chain 1. Nothing inhibits this, so Chain 1 is assigned to *disclosing party*. This process continues until all slots are assigned for the given document.

To clarify what has happened here, the assignment is of a chain—and specifically, of its surface form—that corresponds with the slot—and its mention form—in the document. Our annotations also correspond with such mention forms, and thus we evaluate based on whether the mention form identified by the system corresponds with the mention form annotated.

6 Results

Evaluation employs a holdout of 19 NDAs and 18 Services agreements against a total of 14 different models—7 for NDAs and 7 for Services—representing different ablations of the features considered.[2] The first ablation predicts class based on only one feature: either the mention $P(k, m)$, dependency relation label $P(k, r)$, or dependency parent $P(k, p)$. The second ablation predicts class based on pairs of features: mention and relation $P(k, m, r)$, mention and parent $P(k, m, p)$, and relation and parent $P(k, r, p)$. The third ablation considers all features jointly, yielding one model $P(k, m, r, p)$.

Table 2: Precision and recall scores for the model ablation study.

Joint prob. in PMI numer.	NDA		Services	
	Prec.	Rec.	Prec.	Rec.
$P(k, m)$	0.882	0.882	0.639	0.582
$P(k, r)$	0.300	0.039	0.167	0.051
$P(k, p)$	0.691	0.500	0.302	0.241
$P(k, m, p)$	0.900	0.829	0.623	0.544
$P(k, m, r)$	0.855	0.855	0.625	0.570
$P(k, r, p)$	0.947*	0.934*	0.806*	0.734*
$P(k, m, r, p)$	0.901	0.842	0.623	0.544

At the first level of ablation, the mention-only model ($P(k, m)$) performs best on both NDAs and Services agreements. When other features are added to the mentions, however, the mention-

[2] A reviewer pointed out the limitations of this experimental configuration, as the absence of a third dev split lowers the rigorousness of these results. The absence of this further split is, in part, a consequence of the success of the system in production. There have been too few complaints from users about the slot system to justify further improvement.

related models ($P(k, m, p)$ and $P(k, m, r)$) generally decline in performance (NDA recall, Services precision and recall) or experience only small improvement (NDA precision) in performance.

However, when the two features from the first ablation that performed the most poorly are combined, they result in the highest performance over all, across both genres of contract.

7 Analysis

The results in both genres show an interesting trend—when the values are considered independently, mention is the strongest feature in terms of performance. However, this changes in the second level, where mention being excluded and only the dependency-parent pairs of mentions (i.e. $P(k, r, p)$) results in the strongest performance. This outperforms the results in the third ablation, leading to a decline in precision and recall. These trends hold in both the NDA and Services genres of contracts, albeit to different degrees.

The relationship between a candidate chain and the rest of the contract—as embodied through its dependency labels and parent—best embodies the slots, even more than the explicit mention of the label. Though this was somewhat predictable from prior literature, it has not been explicitly tested. It is actually striking that, in a genre of text where the slot itself is so often explicitly named that performance is higher when that explicit naming is omitted, and the decision itself is only directly connected to how a particular entity participates in a contract rather than what it is called.

Among the top performing models, errors were generally concentrated in the terms used to refer to either one of the parties, especially in the NDA results.

Examining the output more closely, mention form's harm to performance can be more readily teased apart. A particularly interesting NDA, "Software Group Inc. NDA" actually expresses a four-part relationship between the State of Minnesota, the State Court Administrator's Office, a contractor, and another company Tyler Technologies. These last three are defined as "Court," "Recipient," and "Tyler" respectively. The agreement seeks to protect, primarily, "Tyler trade secret information," which our annotators marked as the term for *confidential information*, and since it revolves around Tyler's trade secrets, "Tyler" is the *disclosing party*; "Recipient," in this case, is actually the *receiving party*.

Both the $P(k, r, p)$ and $P(k, m, r, p)$ models made errors on this document, but different errors that reveal some of their flaws. The $P(k, m, r, p)$ model missed slots that included "Tyler" in the name, as that explicit mention form was unpredictable in advance. Instead, the model guessed nothing for the *disclosing party* and "information" for *confidential information*. On the other hand, the $P(k, r, p)$ model was able to get both of those slots correct; however, it incorrectly guessed the other defined term "Court" to be the *receiving party*. This is due to a use of "Court" in a very similar manner to "Recipient" and is expressed in relationships very similar to other chains in prior texts labelled *receiving party*. For example, in defining exclusions to the *confidential information*, the contract states (emphasis added):

> "Tyler trade secret information" shall not include (i) any information which was known to or readily ascertainable by proper means by the *Court or Recipient* before being disclosed to the *Court or Recipient* by Tyler; (ii) any information which is or becomes available to the general public without fault or action of the *Court or Recipient*...

The relations expressed here for "Court" are ⟨Court, prep_by, ascertainable⟩, ⟨Court, prep_to, disclosed⟩, and ⟨Court, prep_of, action⟩, all relations indicative of a *receiving party* while "Recipient", in our model, only obtains three instances of ⟨Recipient, conj, Court⟩. A more sophisticated handling of the dependency tree may rectify this—retrieving the proper semantic relationship for "Recipient." Above all, while these are errors, none of them are embarrassing errors; their causes are highly transparent given the design of the model and suggest opportunities for further improvement.

In the Services agreements, the error rate was higher for all models. This may, in part, be due to data sparsity, since far fewer Services agreements were available for annotation. However, there were differences between the slots in each genre. NDAs preferred more generic mention forms—e.g. "Discloser"—while Services agreements usually have at least one company name as one of the parties, so this hurt the $P(k, m, r, p)$ model more than in the NDAs where it labelled at least one slot wrong in all but one evaluation contract (17 of 18) because the mention contained a

proper noun. The $P(k, r, p)$ model also made some similar errors, but only in 6 of the 18 contracts. Additionally, Services agreements were more likely to have multiple mention forms for the same slot—for example, the "Apple - Developer Agreement" uses both "you" and "Apple Developer" for *service receiver*. Our algorithm is incapable of determining that both of those chains are part of the same slot—a limitation, but an intentional one, since opening up the possibility for slots to be multiple chains opens up tremendous room for error.

8 Conclusions

This paper demonstrates a technique for identifying participant slots in NDAs and Services agreements. The mention form alone obtains high performance, but performance declines as other features are added. This is likely because the specific mention form makes it harder to capture the highly common phenomenon of referring to a slot by a party name rather than the name of the role. Those features, dependency relation and dependency parent, on their own—excluding mention form—outperform mention form alone. This shows that what a participant does better reflects their role in a contract rather than the specific string used to refer to them.

There are a number of improvements that could be made to future systems. It is possible that a model that considers the disjunction of the mention form and relation-parent pairs could perform even better than those attempted here, which are strictly joint models. This would more tightly conform to Chambers and Jurafsky (2009)'s typed schemas system design. Further annotation of Services agreements would likely result in performance improvements, as well as better traversal of the parse tree, improved parsers for contract language, and architectures that allow for multiple mention form chains to be considered for a single slot to capture ambiguous cases.

Acknowledgements

Special thanks to our annotators Mariam Thomas, Krishna Shah, Jillian Watkins, and Dan Broderick, as well as Jonathan Herr, Viren Shah, and Ryan Mannion for their feedback on this work, as well as the reviewers who strengthened many of the aspects of this work as well as suggested some directions forward.

References

Arvind Agarwal, Laura Chiticariu, Poornima Chozhiyath Raman, Marina Danilevsky, Diman Ghazi, Ankush Gupta, Shanmukha Guttula, Yannis Katsis, Rajasekar Krishnamurthy, Yunyao Li, Shubham Mudgal, Vitobha Munigala, Nicholas Phan, Dhaval Sonawane, Sneha Srinivasan, Sudarshan R. Thitte, Mitesh Vasa, Ramiya Venkatachalam, Vinitha Yaski, and Huaiyu Zhu. 2021. Development of an Enterprise-Grade Contract Understanding System. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers*, pages 222–229, Online. Association for Computational Linguistics.

Preethu Rose Anish, Abhishek Sainani, Nitin Ramrakhiyani, Sachin Pawar, Girish K Palshikar, and Smita Ghaisas. 2019. Towards Disambiguating Contracts for their Successful Execution - A Case from Finance Domain. In *Proceedings of the First Workshop on Financial Technology and Natural Language Processing*, pages 8–13, Macao, China.

Elliott Ash, Jeff Jacobs, Bentley MacLeod, Suresh Naidu, and Dominik Stammbach. 2020. Unsupervised extraction of workplace rights and duties from collective bargaining agreements. In *2020 International Conference on Data Mining Workshops (ICDMW)*, pages 766–774.

Collin F Baker, Charles J Fillmore, and John B Lowe. 1998. The Berkeley Framenet Project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics.

Niranjan Balasubramanian, Stephen Solderland, Mausam, and Oren Etzioni. 2013. Generating coherent event schemas at scale. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1721–1731. Association for Computational Linguistics.

Łukasz Borchmann, Dawid Wisniewski, Andrzej Gretkowski, Izabela Kosmala, Dawid Jurkiewicz, Łukasz Szałkiewicz, Gabriela Pałka, Karol Kaczmarek, Agnieszka Kaliska, and Filip Graliński. 2020. Contract discovery: Dataset and a few-shot semantic retrieval challenge with competitive baselines. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4254–4268, Online. Association for Computational Linguistics.

Nathanael Chambers and Dan Jurafsky. 2008. Unsupervised learning of narrative event chains. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL)*, pages 789–797. Association for Computational Linguistics.

Nathanael Chambers and Dan Jurafsky. 2009. Unsupervised learning of narrative schemas and their participants. In *Proceedings of the Proceedings of the Joint*

Conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation for Natural Language Processing., pages 602–610. Association for Computational Linguistics.

Michael Curtotti and Eric C. McCreath. 2011. A Corpus of Australian Contract Language: Description, Profiling and Analysis. In *Proceedings of the 13th International Conference on Artificial Intelligence and Law*, ICAIL '11, pages 199–208, New York, NY, USA. ACM.

M. de Marneffe, B. MacCartney, and C.D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the Language Resources and Evaluation Conference of the European Language Resources Association (LREC)*. Association for Computational Linguistics.

Ruka Funaki, Yusuke Nagata, Kohei Suenaga, and Shinsuke Mori. 2020. A Contract Corpus for Recognizing Rights and Obligations. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 2045–2053, Marseille, France. European Language Resources Association.

Matthew Honnibal and Mark Johnson. 2015. An Improved Non-monotonic Transition System for Dependency Parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal. Association for Computational Linguistics.

B. Jans, S. Bethard, I. Vulić, and M.F. Moens. 2012. Skip n-grams and ranking functions for predicting script events. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 336–344. Association for Computational Linguistics.

John Justeson and Slava Katz. 1995. Technical Terminology: Some Linguistic Properties and an Algorithm for Identification in Text. *Natural Language Engineering*, 1:9–.

Moniba Keymanesh, Micha Elsner, and Srinivasan Sarthasarathy. 2020. Toward domain-guided controllable summarization of privacy policies. In *Proceedings of the Natural Legal Language Processing Workshop 2020 co-located with the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2020), Virtual Workshop, August 24, 2020*, volume 2645 of *CEUR Workshop Proceedings*, pages 18–24. CEUR-WS.org.

Laura Manor and Junyi Jessy Li. 2019. Plain English Summarization of Contracts. In *Proceedings of the Natural Legal Language Processing Workshop 2019*, pages 1–11, Minneapolis, Minnesota. Association for Computational Linguistics.

Raymond Mooney and Gerald DeJong. 1985. Learning schemata for natural language processing. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 681–687.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and evaluation framework for deeper understanding of commonsense stories. *arXiv preprint arXiv:1604.01696*.

Karl Pichotta and Raymond J Mooney. 2014. Statistical script learning with multi-argument events. In *EACL*, volume 14, pages 220–229.

Roger Schank and Robert Abelson. 1977. *Scripts, plans, goals and understanding: An inquiry into human knowledge structures.* Lawrence Erlbaum, New Jersey.

Dan Simonson, Daniel Broderick, and Jonathan Herr. 2019. The Extent of Repetition in Contract Language. In *Proceedings of the Natural Legal Language Processing Workshop 2019*, pages 21–30, Minneapolis, Minnesota. Association for Computational Linguistics.

Dan Simonson and Anthony Davis. 2018. Narrative Schema Stability in News Text. In *Proceedings of COLING 2018*, Santa Fe, NM. Association for Computational Linguistics.

Daniel E. Simonson, Jonathan Herr, Joey T. Avant, Garen P. Riedel, and Daniel P. Broderick. 2020. Systems, Methods, and Computer Program Products for Slot Normalization of Text Data. U.S. Patent and Trademark Office, US10614157.

Noah Weber, Leena Shekhar, Niranjan Balasubramanian, and Nathanael Chambers. 2018. Hierarchical Quantized Representations for Script Generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3783–3792, Brussels, Belgium. Association for Computational Linguistics.