Statutory Legal Reasoning: Challenging Natural Language Systems with Understanding Prescriptive Rules

Anonymous authors

Abstract

Legislation can be viewed as a body of prescriptive rules expressed in natural language. The application of legislation to facts of a case we refer to as statutory reasoning, where those facts are also expressed in natural language. Computational statutory reasoning is distinct from existing work in machine reading, in that much of the information needed for deciding a case is declared exactly once (a law), while much of the information needed in machine reading tends to be learned through distributional language statistics. To measure the performance of natural language understanding approaches on statutory reasoning, we introduce a benchmark dataset, together with a legal-domain text corpus. Current models exhibit low out-of-the-box performance on our challenge task, whether or not they have been fine-tuned to the legal domain. In addition, we provide a Prolog-based system, able to fully solve the task, but requiring human construction. This suggests the task of statutory reasoning as an interesting real-world problem space, and motivates the development of models able to utilize prescriptive rules specified in natural language.

1. Introduction

Early artificial intelligence research focused on highly-performant, narrow-domain reasoning models, for instance in health [Ledley and Lusted, 1959, Shortliffe and Buchanan, 1975, Miller et al., 1982] and law [McCarty, 1976, Hellawell, 1980]. Such expert systems relied on hand-crafted inference rules and domain knowledge, expressed and stored with the formalisms provided by databases [Feigenbaum, 1992]. The main bottleneck of this approach is that experts are slow in building such knowledge bases and exhibit imperfect recall, which motivated research into models for automatic information extraction (e.g. Lafferty et al. [2001]). Systems for large-scale automatic knowledge base construction have improved (e.g. Etzioni et al. [2008], Mitchell et al. [2018]), as well as systems for sentence level semantic parsing [Zhang et al., 2019]. Among others, this effort has led to question-answering systems for games [Ferrucci et al., 2010] and, more recently, for science exams [Friedland et al., 2004, Gunning et al., 2010, Clark et al., 2019]. The challenges include extracting ungrounded
knowledge from semi-structured sources, e.g. textbooks, and connecting high-performance symbolic solvers with large-scale language models.

In parallel, models have begun to consider task definitions like machine reading [Rajpurkar et al., 2018] and natural language inference [Wang et al., 2019] as not requiring the use of explicit structure. Instead, the problem is cast as one of mapping inputs to high-dimensional, dense representations that implicitly encode meaning [Devlin et al., 2018, Raffel et al., 2019], and are employed in building classifiers or text decoders, bypassing classic approaches to symbolic inference.

This work is concerned with the problem of how to reason about an example situation, a case, based on complex rules provided in natural language. Statutory reasoning [Yoshioka et al., 2018, Zhong et al., 2019] is an excellent instance of this problem. In addition to the reasoning aspect, we are motivated by the lack of contemporary systems to suggest legal opinions: while there exist tools to aid lawyers in retrieving relevant documents for a given case, we are unaware of any strong capabilities in automatic statutory reasoning.

Our contributions, summarized in Figure 2, include a novel dataset based on US tax law, together with test cases (Section 2). Decades-old work in expert systems could solve problems of the sort we construct here, based on manually derived rules: we replicate that approach here as a Prolog-based system that achieves 100% accuracy on our examples (Section 3). Our results demonstrate that straight-forward application of contemporary NLU models is not sufficient for our challenge examples (Section 5), whether or not they were adapted to the legal domain (Section 4). This is meant to provoke the question of whether we should be concerned with: (a) improving methods in semantic parsing in order to replace manual transduction into symbolic form; or (b) improving machine reading methods in order to avoid explicit symbolic solvers. We view this work as part of the conversation including recent work in multi-hop inference [Yang et al., 2018], where our task is more domain specific but potentially more challenging.
2. Dataset

Here, we describe our main contribution, the StAtutory Reasoning Assessment dataset (SARA): a set of rules extracted from the statutes of the US Internal Revenue Code, together with a set of natural language questions which may only be answered correctly by referring to the rules. Section 8.1 of the Appendix provides some background on Title 26 of the United States Code, which was drawn from to build this dataset.

Statutes and test cases We selected sections that cover tax on individuals (§1), marriage and other legal statuses (§2, 7703), dependents (§152), tax exemptions and deductions (§63, 68, 151) and employment (§3301, 3306) as the basis of our set of rules. We simplified the sections to (1) remove highly specific sections (e.g. those concerning the employment of sailors) to keep the statutes to a manageable size, and (2) ensure that the sections only refer to sections from the selected subset. For ease of comparison with the original statutes,
we kept the original numbering and lettering, with no adjustment for removed sections. For example, there is a section 63(d) and a section 63(f), but no section 63(e). We assumed that any taxable year starts and ends at the same time as the corresponding calendar year.

For each subsection extracted from the statutes, we manually created two paragraphs in natural language describing a case, one where the statute applies, and one where it does not. These snippets are meant to test a system’s understanding of the statutes, as illustrated in Figure 1. For the purposes of machine learning, the cases were split up into 176 train and 100 test samples, such that (1) each pair of positive and negative cases belongs to the same split, and (2) each section is split between train and test in the same proportions as the overall split.

In addition, since tax legislation makes it possible to predict how much tax a person owes, we created a set of 100 cases where the task is to predict how much tax someone owes. Those cases were created by randomly mixing and matching pairs of cases from the first set of cases, and resolving inconsistencies manually. Those cases are no longer a binary prediction task, but a task of predicting an integer. The prediction results from taking into account the entirety of the statutes, and involves some basic arithmetic. The 100 cases were randomly split into 80 training and 20 test samples.

Because the statutes were simplified, the answers to the cases are not those that would be obtained with the current version of the IRC. In particular, some of the IRC counterparts of the statutes in our dataset have been repealed, amended, or adjusted to reflect inflation.

**Key features of the corpus** While the corpus is based on a simplification of the Internal Revenue Code, care was taken to retain prominent features of US law. We note that the present task is only one aspect of legal reasoning, which in general involves many more modes of reasoning, in particular interpreting regulations and prior judicial decisions.

*Reasoning with time.* The timing of events (marriage, retirement, income...) is highly relevant to determining whether certain sections apply, as tax is paid yearly. In addition, some sections require counting days, as in §7703(b)(1):

\[
\text{a household which constitutes for more than one-half of the taxable year the principal place of abode of a child}
\]

or taking into account the absolute point in time as in §63(c)(7):

\[
\text{In the case of a taxable year beginning after December 31, 2017, and before January 1, 2026-}
\]

*Exceptions and substitutions.* Typically, each section of the IRC starts by defining a general case and then enumerates a number of exceptions to the rule. Additionally, some rules involve applying a rule after substituting terms. For example, §63(f)(3):

\[
\text{In the case of an individual who is not married and is not a surviving spouse, paragraphs (1) and (2) shall be applied by substituting “$750” for “$600”}
\]

*Numerical reasoning.* Computing tax owed requires knowledge of the basic arithmetic operations of adding, subtracting, multiplying, dividing, rounding and comparing numbers. Moreover, the operation to be used needs to be parsed out of natural text, as in §1(c)(2):
$3,315, plus 28% of the excess over $22,100 if the taxable income is over $22,100 but not over $53,500

Cross-references. Each section of the IRC will typically reference other sections. Figure 4 in the Appendix shows how this feature was preserved in our dataset. There are explicit references within the same section, as in §7703(b)(1):

an individual who is married (within the meaning of subsection (a)) and who files a separate return

explicit references to another section, as in §3301:

There is hereby imposed on every employer (as defined in section 3306(a)) for each calendar year an excise tax

and implicit references, as in §151(a), where “taxable income” is defined in §63:

In the case of an individual, the exemptions provided by this section shall be allowed as deductions in computing taxable income.

Common sense knowledge. Four concepts, other than time, are left undefined in our statutes: (1) kinship, (2) the fact that a marriage ends if either spouse dies, (3) if an event has not ended, then it is ongoing; if an event has no start, it has been true at any time before it ends; and some events are instantaneous (e.g. payments), (4) a person’s gross income is the sum of all income and payments received by that person.

3. Prolog solver

It has been shown previously that subsets of statutes can be expressed in first-order logic, as described in Section 6. As a reaffirmation of this, and as a topline for our task, we have manually translated the statutes into Prolog rules and the cases into Prolog facts, such that each case can be answered correctly by a single Prolog query. The Prolog rules were developed based on the statutes, meaning that the Prolog code clearly reflects the semantics of the textual form, as in Gunning et al. [2010]. This is primarily meant as a proof that a carefully crafted reasoning engine, with perfect natural language understanding, can solve this dataset. There certainly are other ways of representing this given set of statutes and cases. The point of this dataset is not to design a better Prolog system, but to help the development of language understanding models capable of reasoning. We report the detail of our design choices in Section 8.2 of the Appendix.

Statutes Each subsection of the statutes was translated with a single rule, true if the section applies, false otherwise. In addition, subsections define slots that may be filled and reused in other subsections. For example, §7703(b)(1) mentions a “household” which is also referred to in §7703(b)(2). To solve this coreference problem, any term appearing in a subsection and relevant across subsections is turned into an argument of the Prolog rule. The corresponding variable may then be bound during the execution of a rule, and reused in a rule executed later. Unfilled slots correspond to unbound variables.

To check whether a given subsection applies, the Prolog system needs to rely on certain predicates, which directly reflect the facts contained in the natural language descriptions
of the cases. For instance, how do we translate *Alice and Bob got married on January 24th, 1993* into code usable by Prolog? We rely on a set of 57 predicates, following neo-davidsonian semantics [Davidson, 1967, Castañeda, 1967, Parsons, 1990]. The level of detail of these predicates is based on the granularity of the statutes themselves. Anything the statutes do not define, and which is typically expressed with a single word, is potentially such a predicate: marriage, residing somewhere, someone paying someone else, etc. The example above is translated in Figure 3.

**Cases** The natural language description of each case was manually translated into the basic facts mentioned in the previous section. The question was translated into a Prolog query or goal. For instance, *Does section 7703(b)(3) apply to Alice maintaining her home for the year 2018?* translates to

\[
s7703_b_3(alice,home,2018).
\]

and *How much tax does Alice have to pay in 2017?* translates to

\[
\]

**4. Legal NLP**

We trained two existing models on legal corpora to adapt them to the legal domain.

**Text corpus** We curated a corpus consisting solely of freely-available tax law documents with 147M tokens. The first half of this corpus is drawn from cas [2019], a project of Harvard’s Law Library that scanned and OCR’ed many of the library’s case-law reporters, making the text available upon request to researchers. The main challenge in using this resource is that it contains 1.7M U.S. federal cases, only a small percentage of which are on tax law (as opposed to criminal law, breach of contract, bankruptcy, etc.). Classifying cases by area is a non-trivial problem [Soh et al., 2019], and tax-law cases are litigated in many different courts. We used the heuristic of classifying a case as being tax-law if it met one of the following criteria: the Commissioner of Internal Revenue was a party; the case was decided by the U.S. Tax Court; or, the case was decided by any other federal court, other than a trade tribunal, with the United States as a party, and with the word *tax* appearing in the first 400 words of the case’s written opinion.

The second half of this corpus consists of IRS private letter rulings and unpublished U.S. Tax Court cases. IRS private letter rulings are similar to cases, in that they apply tax law to one taxpayer’s facts; they differ from cases in that they are written by IRS attorneys (not judges), have less precedential authority than cases, and redact names to protect taxpayer privacy. Unpublished U.S. Tax Court cases are viewed by the judges writing them as less important than those worthy of publication. These were downloaded as PDFs from the IRS and Tax Court websites, OCR’ed with tesseract if needed, and otherwise cleaned.

**Tax vectors** Before training a word2vec model [Mikolov et al., 2013], on this corpus, we did two tax-specific preprocessing steps to ensure that semantic units remained together. First, we put underscores between multi-token collocations that are tax terms of art, defined in either the tax code, Treasury regulations, or a leading tax-law dictionary. Thus, “surviving spouse” became the single token “surviving_spouse”. Second, we turned all tax
code sections and Treasury regulations into a single token, stripped of references to subsections, subparagraphs, and subclauses. Thus, “Treas. Reg. § 1.162-21(b)(1)(iv)” became the single token “sec_1.162.21”. The vectors were trained at 500 dimensions using skip-gram with negative sampling. A window size of 15 was found to maximize performance on twelve human-constructed analogy tasks.

**Legal BERT**  We performed further training of BERT [Devlin et al., 2018], on a portion of the full case.law corpus, including both state and federal cases. We did not limit the training to tax cases. Rather, the only cases excluded were those under 400 characters (which tend to be summary orders with little semantic content) and those before 1970 (when judicial writing styles had become recognizably modern). We randomly selected a subset of the remaining cases, and broke all selected cases into chunks of exactly 510 tokens, which is the most BERT’s architecture can handle. Any remaining tokens in a selected case were discarded. Using solely the masked language model task (i.e. not next sentence prediction), starting from Bert-Base-Cased, we trained on 900M tokens.

The resulting Legal BERT has the exact same architecture as Bert-Base-Cased but parameters better attuned to legal tasks. We applied both models to the natural language questions and answers in the corpus we introduce in this paper. While Bert-Base-Cased had a perplexity of 14.4, Legal BERT had a perplexity of just 2.7, suggesting that the further training on 900M tokens made the model much better adapted to legal queries.

We also probed how this further training impacted ability to handle fine-tuning on downstream tasks. The downstream task we chose was identifying legal terms in case texts. For this task, we defined legal terms as any tokens or multi-token collocations that are defined in Black’s Law Dictionary, the premier legal dictionary. We split the legal terms into training/dev/test splits. We put a 4-layer fully-connected MLP on top of both Bert-Base-Cased and Legal BERT, where the training objective was B-I-O tagging of tokens in 510-token sequences. We trained both on a set of 200M tokens randomly selected from case.law cases not previously seen by the model and not containing any of the legal terms in dev or test, with the training legal terms tagged using string comparisons. We then tested both fine-tuned models’ ability to identify legal terms from the test split in case law. The model based on Bert-Base-Cased achieved F1 = 0.35, whereas Legal BERT achieved F1 = 0.44. As a baseline, two trained lawyers given the same task on three 510-token sequences each achieved F1 = 0.26. These results indicate that Legal BERT is much better adapted to the legal domain than Bert-Base-Cased.

5. Experiments

**BERT-based models**  A given binary question \(q\) mentions the subsection in question (as in Figure 1). We extract \(s\), the text of the subsection in question, from the statutes, following Algorithm 2. In \(q\), we replace Does section XYZ apply with Does this apply. We feed the string “\([CLS] + s + [SEP] + q + c + [SEP]\)” where “+” is string concatenation, to BERT [Devlin et al., 2018]. Let \(r\) be the vector representation of the token [CLS] in the final layer. The answer to a binary question is predicted as \(g(\theta_1 \cdot r)\) where \(\theta_1\) is a learnable parameter and \(g\) is the sigmoid function. For numerical questions, all statutes have to be taken into account, which would not fit into BERT’s length limit. We encode “\([CLS] all [SEP] + q + c + [SEP]\)” into \(r\) and predict the answer as \(\mu + \sigma \theta_2 \cdot r\) where \(\theta_2\)
is a learned parameter, and $\mu$ and $\sigma$ are the mean and standard deviation of the numerical answers on the training set, respectively.

The loss function used is

$$L = \sum_{i \in I_1} y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i) + \sum_{i \in I_2} \frac{1}{\sigma^2}||\hat{y}_i - y_i||^2$$

where $I_1$ (resp. $I_2$) is the set of binary (resp. numerical) questions, $y_i$ is the ground truth output, and $\hat{y}_i$ is the model’s output.

We use Adam [Kingma and Ba, 2014] with a linear warmup schedule for the learning rate. We freeze BERT’s parameters, and experiment with unfreezing BERT’s top layer. We select the final model based on early stopping with 10% of the training examples randomly set aside at the beginning of training as a dev set. We select the best performing model for binary questions and for numerical questions separately. For hyperparameter search, we randomly sample 50 sets of hyperparameters from the space described in Table 2 in the Appendix. In our ablation experiments, we drop either the statute, or the context and the statute, in which case we predict the answer from BERT’s representation for “[CLS] + $c$ + [SEP] + $q$ + [SEP]” or “[CLS] + $q$ + [SEP]”, whichever is relevant.

**Table 1**: Test set scores. “question” models have access to the question only, “context” to the context and question, and “statute” to the statutes, context and question. We report the 90% confidence interval.

<table>
<thead>
<tr>
<th>Model</th>
<th>Features</th>
<th>Inputs</th>
<th>Acc.</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>-</td>
<td>-</td>
<td>50 ± 8.3</td>
<td>240 ± 405</td>
</tr>
<tr>
<td>BERT-based</td>
<td>BERT</td>
<td>question</td>
<td>51 ± 8.3</td>
<td>241 ± 407</td>
</tr>
<tr>
<td></td>
<td></td>
<td>context</td>
<td>50 ± 8.3</td>
<td>240 ± 404</td>
</tr>
<tr>
<td></td>
<td></td>
<td>statutes</td>
<td>53 ± 8.3</td>
<td>240 ± 404</td>
</tr>
<tr>
<td></td>
<td>+ unfreeze</td>
<td>statutes</td>
<td>51 ± 8.3</td>
<td>241 ± 405</td>
</tr>
<tr>
<td>Legal BERT</td>
<td>question</td>
<td>49 ± 8.3</td>
<td>240 ± 406</td>
<td></td>
</tr>
<tr>
<td></td>
<td>context</td>
<td>50 ± 8.3</td>
<td>239 ± 403</td>
<td></td>
</tr>
<tr>
<td></td>
<td>statutes</td>
<td>46 ± 8.3</td>
<td>239 ± 403</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ unfreeze</td>
<td>statutes</td>
<td>48 ± 8.3</td>
<td>240 ± 404</td>
</tr>
<tr>
<td>relational</td>
<td>tax vectors</td>
<td>question</td>
<td>52 ± 8.3</td>
<td>237 ± 400</td>
</tr>
<tr>
<td>neural</td>
<td></td>
<td>context</td>
<td>51 ± 8.3</td>
<td>227 ± 383</td>
</tr>
<tr>
<td></td>
<td></td>
<td>statutes</td>
<td>51 ± 8.3</td>
<td>239 ± 403</td>
</tr>
<tr>
<td>word2vec</td>
<td>question</td>
<td>50 ± 8.3</td>
<td>241 ± 406</td>
<td></td>
</tr>
<tr>
<td></td>
<td>context</td>
<td>49 ± 8.3</td>
<td>241 ± 406</td>
<td></td>
</tr>
<tr>
<td></td>
<td>statutes</td>
<td>50 ± 8.3</td>
<td>241 ± 406</td>
<td></td>
</tr>
<tr>
<td>relational</td>
<td>tax vectors</td>
<td>question</td>
<td>51 ± 8.3</td>
<td>236 ± 399</td>
</tr>
<tr>
<td>non-neural</td>
<td></td>
<td>context</td>
<td>51 ± 8.3</td>
<td>237 ± 400</td>
</tr>
<tr>
<td></td>
<td></td>
<td>statutes</td>
<td>50 ± 8.3</td>
<td>237 ± 396</td>
</tr>
<tr>
<td>word2vec</td>
<td>question</td>
<td>50 ± 8.3</td>
<td>241 ± 406</td>
<td></td>
</tr>
<tr>
<td></td>
<td>context</td>
<td>49 ± 8.3</td>
<td>241 ± 406</td>
<td></td>
</tr>
<tr>
<td></td>
<td>statutes</td>
<td>49 ± 8.3</td>
<td>241 ± 406</td>
<td></td>
</tr>
</tbody>
</table>

**Relational models** We use Arora et al. [2016] to embed strings into vectors, with smoothing parameter equal to $10^{-3}$. We use either tax vectors described in Section 4 or word2vec vectors [Mikolov et al., 2013]. We estimate unigram counts from the corpus used to build the tax vectors, or the training set, whichever is relevant. For a given context $c$ and question $q$, we retrieve relevant subsection $s$ as above. Using Arora et al. [2016], $s$ is mapped to vector $v_s$, and $(c, q)$ to $v_{c+q}$. Let $r = [v_s, v_{q+c}, v_{s-v_{c+q}}, v_s \odot v_{c+q}]$ where $[a, b]$ is the concatenation of $a$ and $b$, $|.|$ is the element-wise absolute value, and $\odot$ is the element-wise product. The
answer is predicted as \( g(\theta_1 \cdot f(r)) \) or \( \mu + \sigma \theta_2 \cdot f(r) \), as above, where \( f \) is a feed-forward neural network. We use batch normalization between each layer of the neural network [Ioffe and Szegedy, 2015]. This approach is modeled after relational networks [Santoro et al., 2017]. As above, we perform ablation experiments, where we drop the statute, or the context and the statute, in which case \( r \) is replaced by \( v_{c+q} \) or \( v_q \). We also experiment with \( f \) being the identity function (no neural network). We sample 150 sets of hyperparameters from the space described in Table 3 in the Appendix. Training is otherwise done as above, but without the warmup schedule.

Results

We report the accuracy on the binary questions from the test set, as well as the mean squared error (MSE) on the numerical questions of the test set. Results are described in Table 1. As a check, we swapped in the RTE dataset from the SuperGLUE benchmark, and achieved 73.6% accuracy on the dev set, close to numbers reported in Wang et al. [2019]. The baseline predicts either the majority class, or the average tax (estimated on the train set). BERT was trained on Wikipedia, which contains snippets of law text. See article United States Code and links therefrom, especially Internal Revenue Code. Overall, models perform comparably to the baseline, independent of the underlying method. Performance remains mostly unchanged when dropping the statutes or statutes and context, meaning that models are not utilizing the statutes to reason. Adapting BERT or word vectors to the legal domain has no noticeable effect. This suggests the limiting factor is in the reasoning.

6. Related work

Clark et al. [2019] describe a decade-long effort to answer science exam questions stated in natural language, based on descriptive knowledge stated in natural language. The final system relies on a variety of NLP and specialized reasoning techniques. In contrast, our work involves prescriptive knowledge. The task introduced by Weston et al. [2015] requires some amount of reasoning, and some amount of common sense knowledge. Contrary to the present work, the types of question in the train and test sets are highly related, and the vocabulary overlap is quite high.

There have been several efforts to translate law statutes into expert systems. TAXMAN [McCarty, 1976] focuses on corporate reorganization law, and is able to classify a case into three different legal types of reorganization, following a theorem-proving approach. Sergot et al. [1986] translate the major part of the British Nationality Act 1981 into around 150 rules in micro-Prolog, proving the suitability of Prolog logic to express and apply legislation. Closest to our work is Sherman [1987], who manually translated part of Canada’s Income Tax Act into a Prolog program. To our knowledge, the projects cited did not include a dataset or task that the programs were applied to. Other works have similarly described the formalization of law statutes into rule-based systems [Hellawell, 1980, Satoh et al., 2010, Fungwachararakorn and Satoh, 2018, Khan et al., 2016].

Yoshioka et al. [2018] introduce a dataset of Japanese statute law and its English translation, together with questions collected from the Japanese bar exam. To tackle these two tasks, Kim et al. [2019] investigate heuristic-based and machine learning-based methods. A similar dataset based on the Chinese bar exam was released by Zhong et al. [2019]. Many papers explore case-based reasoning for law, with expert systems [Popp and Schlink, 1974, vdL Gardner, 1983], human annotations [Bruninghaus and Ashley, 2003] or automatic an-
notations [Ashley and Brüninghaus, 2009] as well as transformer-based methods [Rabelo et al., 2019]. Some datasets are concerned with very specific tasks, as in tagging in contracts [Chalkidis and Androutsopoulos, 2017], classifying clauses [Chalkidis et al., 2018], and classification of documents [Chalkidis et al., 2019] or single paragraphs [Biagioli et al., 2005]. Ravichander et al. [2019] have released a dataset of questions about privacy policies, elicited from turkers and answered by legal experts.

Text that describes procedural knowledge, which is knowledge needed to perform a task, is similar to statutes. Zhang et al. [2012] published a dataset of how-to instructions, with human annotations defining key attributes (actee, purpose...) and models to automatically extract the attributes. Similarly, Chowdhury et al. [2020] describe a dataset of human-elicited procedural knowledge, and Wambsganß and Fromm [2019] automatically detect repair instructions from posts on an automotive forum. Branavan et al. [2012] use information from a game manual to improve the performance of an agent playing the game.

7. Conclusion

We introduce a resource of law statutes and a challenge dataset of hand-curated rules and cases. We also contribute a fully symbolic approach based on Prolog, able to represent the rules and solve the challenge task. This hand-built solver contrasts with our baselines based on current NLP approaches, even when we adapt them to the legal domain.

The intersection between natural language processing and the legal domain is a growing area of research [Rissland et al., 2003, Ashley and Brüninghaus, 2009, Chalkidis et al., 2018, Eidelman, 2019, Kim et al., 2019], but with few large-scale systematic resources. Thus, in addition to the exciting challenge posed by statutory reasoning, we also intend this paper to be a contribution to legal-domain natural language processing.

Given the poor out-of-the-box performance of otherwise very powerful models, this dataset raises the question of what the most promising direction of research would be. There are at least two strategies open to the community: automatic extraction of knowledge graphs from text with the same accuracy as we did for our Prolog solver [Viet et al., 2017]; and further improvements to machine reading. We hope our resource provides (1) a benchmark for a challenging aspect of reasoning as well as for legal NLP, and (2) legal-domain NLP models useful for the research community. Future work includes further expansion of the challenge set, and consideration of more sophisticated, explicitly multi-step inference models that may be trained to understand legislation on the fly.

References


8. Appendix

8.1 US Code

Title 26 of the United States Code \(^1\) (called the IRC) contains rules and definitions for the imposition and calculation of taxes. The IRC is subdivided into sections, which in general,

\(^1\). https://uscode.house.gov/browse.xhtml
define one or more terms. For instance, section 3306 defines the terms employment, employer and wages for purposes of the federal unemployment tax. Each section and its subsections may be cast as a predicate whose truth value can be checked against a state of the world. For instance, subsection 7703(a)(2):

\[
\textit{an individual legally separated from his spouse under a decree of divorce or of separate maintenance shall not be considered as married}
\]

can be checked given an individual.

The structure of §7703 (“section 7703”) is typical for the rest of the IRC [Sherman, 1987]: a general rule (§7703(a)) is followed by a number of exceptions (§7703(b)). Certain roles are implicitly filled: §7703(a)(1) and (b)(3) mention a “spouse”, which must exist since the “individual” is married. In a similar way, roles which have been filled earlier in the section may be referred to later on. For instance, “household” is mentioned for the first time in §7703(b)(1), then again in §7703(b)(2) and in §7703(b)(3). These roles are a major feature of the IRC statutes, and can be cast as arguments of the predicates that correspond to the subsections. Resolving them correctly is a key point in successfully applying the law.

The IRC also contains hypotheticals, as in §7703(b)(1):

\[
\textit{with respect to whom such individual is entitled to a deduction for the taxable year under section 151 (or would be so entitled but for section 152(e))}
\]

Overall, the IRC can be framed as a set of predicates formulated in human language. The ambiguity, incompleteness and vagueness of human language makes it particularly challenging for a computer-based system to determine whether a subsection applies, and to identify and fill the roles mentioned. This makes the IRC an excellent corpus to build systems that reason with rules specified in natural language and have clear language understanding capabilities.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch size</td>
<td>{32, 64, 128}</td>
</tr>
<tr>
<td>learning rate</td>
<td>{5e-5, 2e-5, 1e-5, 5e-6, 2e-6}</td>
</tr>
<tr>
<td>number of epochs</td>
<td>100</td>
</tr>
<tr>
<td>warmup</td>
<td>{.1, .3, .5, .7, .9}</td>
</tr>
</tbody>
</table>

Table 2: Hyperparameters used for the BERT-based models (Section 5). Warmup refers to the fraction of epochs during which the learning rate is linearly increased to its hyperparameter value.

8.2 Design choices

As described in Section 2, the law has certain prominent features which required specific design choices in the Prolog engine.

\textit{Reasoning with time.} US tax law generally disregards time units smaller than a day. We decided to use strings in the “Year-Month-Day” format. For instance “July 4, 1776” is
Table 3: Hyperparameters used for the relational models (Section 5).

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch size</td>
<td>{16, 32, 64, 128, 256}</td>
</tr>
<tr>
<td>learning rate</td>
<td>{5e-1, 2e-1, 1e-1, 5e-2, 2e-2, 1e-2, 5e-3, 2e-3, 1e-3}</td>
</tr>
<tr>
<td>number of epochs</td>
<td>500</td>
</tr>
<tr>
<td>number of layers</td>
<td>{2, 3, 4}</td>
</tr>
<tr>
<td>number of units</td>
<td>{32, 64, 128, 256, 512, 1024, 2048}</td>
</tr>
</tbody>
</table>

Figure 4: References between sections. An arrow indicates a reference from start section to end section, either explicit or implicit.

represented as “1776-07-04”. Years are represented as integers. From there, it is straightforward to implement helper functions that compute whether one day is before the other, or that compute the first day of the year.

Exceptions and substitutions. Exceptions to rules were dealt with by employing Prolog’s negation as failure feature. For instance, the entity \( T \) satisfies §2(a) if it satisfies §2(a)(1) and fails to satisfy §2(a)(2):

\[
\text{s2\_a}(T,S,H,D,Y) :- \text{s2\_a\_1}(T,S,P,H,D,Y), \neg \text{s2\_a\_2}(T,S,P,Y).
\]

This works in conjunction with our closed-world assumption that anything not mentioned in the case description is false.

Common sense knowledge. The concepts mentioned in Section 2 were translated into Prolog functions in a `utils` file. Some implicitly defined words could have motivated building skolem predicates. For example, “marriage” implies the existence of a “spouse”. However, such implicit definitions are bound to legal definitions: to find the spouse of \( X \), we need to determine whether \( X \) is married. In other words, the value of most skolem functions is a slot in a legal definition, making systematic skolemization redundant.
8.3 Algorithm to extract relevant subsection

**Algorithm 1**: Algorithm used to tag each line in the statutes with the sections and subsections that they belong to (called “keys”).

<table>
<thead>
<tr>
<th>Data: Statutes represented as a list of lines statutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result: Keys identifying which subsection each line belongs to all_keys</td>
</tr>
</tbody>
</table>

```python
# KEYS=[
    ['a','b','c','d','e','f','g'],
    [str(ii) for ii in range(1,22)],
    ['A','B','C','D','E','F','G','H'],
    ['i','ii','iii','iv','v'],
    ['I','II']
]

# all_keys,current_keys=[],[]
for s in statutes:
    if s.startswith('Section '):
        current_keys=[(s.lstrip('Section ').split('.')[0],-1)]
    elif s.startswith('('):
        subsection_id=s.lstrip('(').split(')')[0]
        key_level=0
        while subsection_id not in KEYS[key_level]:
            key_level+=1
            cutoff_index=0
        while True:
            if cutoff_index==len(current_keys):
                break
            if current_keys[cutoff_index][1]≥key_level:
                break
            cutoff_index+=1
        current_keys.append((subsection_id,key_level))
        all_keys.append(list(current_keys))

for ii,s in enumerate(statutes):
    if ii==0:
        continue
    if not s.startswith(')'):
        if statutes[ii-1].endswith(', '):
            all_keys[ii].pop()

for ii, in enumerate(all_keys):
    all_keys[ii]=[x[0] for x in all_keys[ii]]
```
Algorithm 2: Algorithm used to extract a string representation of relevant statutes.

**Data:** String question and list all_keys as computed in Algorithm 1

**Result:** String representing relevant statutes relevant_statute

1. if question.lower().startswith('how much tax'):
   2.    relevant_statute='all'
   3.    continue
4. if 'section' in question:
   5.    section_key='section '    
   6. elif 'sec.' in question:
   7.    section_key='sec.'
   8.    relevant_section=question.split(section_key)[1].split(' ')[0].strip('?')
   9.    keys=relevant_section.replace('(',' ').replace(')',').split(' ')
10. relevant_statute="
11. for k,s in zip(all_keys,statutes):
12.    l=min(len(k),len(keys))
13.    if k[:l]==keys[:l]:
14.       relevant_statute=relevant_statute+' '+s
15.    relevant_statute=relevant_statute.strip(' ')