Fonduer: Knowledge Base Construction from Richly Formatted Data

Sen Wu, **Luke Hsiao**, **Xiao Cheng**, **Braden Hancock**, **Theodoros Rekatsinas**[*], **Philip Levis**, and **Christopher Ré**

Stanford University

[*]University of Wisconsin-Madison

Abstract

We focus on knowledge base construction (KBC) from richly formatted data. In contrast to KBC from text or tabular data, KBC from richly formatted data aims to extract relations conveyed jointly via textual, structural, tabular, and visual expressions. We introduce Fonduer, a machine-learning-based KBC system for richly formatted data. Fonduer presents a new data model that accounts for three challenging characteristics of richly formatted data: (1) prevalent document-level relations, (2) multimodality, and (3) data variety. Fonduer uses a new deep-learning model to automatically capture the representation (i.e., features) needed to learn how to extract relations from richly formatted data. Finally, Fonduer provides a new programming model that enables users to convert domain expertise, based on multiple modalities of information, to meaningful signals of supervision for training a KBC system. Fonduer-based KBC systems are in production for a range of use cases, including at a major online retailer. We compare Fonduer against state-of-the-art KBC approaches in four different domains. We show that Fonduer achieves an average improvement of 41 F1 points on the quality of the output knowledge base—and in some cases produces up to $1.87\times$ the number of correct entries—compared to expert-curated public knowledge bases. We also conduct a user study to assess the usability of Fonduer's new programming model. We show that after using Fonduer for only 30 minutes, non-domain experts are able to design KBC systems that achieve on average 23 F1 points higher quality than traditional machine-learning-based KBC approaches.

1 INTRODUCTION

Knowledge base construction (KBC) is the process of populating a database with information from data such as text, tables, images, or video. Extensive efforts have been made to build large, high-quality knowledge bases (KBs), such as Freebase [5], YAGO [38], IBM Watson [6, 10], PharmGKB [17], and Google Knowledge Graph [37]. Traditionally, KBC solutions have focused on relation extraction from unstructured text [23, 27, 36, 44]. These KBC systems already support a broad range of downstream applications such as information retrieval, question answering, medical diagnosis, and data visualization. However, troves of information remain untapped in *richly formatted* data, where relations

and attributes are expressed via combinations of textual, structural, tabular, and visual cues. In these scenarios, the semantics of the data are significantly affected by the organization and layout of the document. Examples of richly formatted data include webpages, business reports, product specifications, and scientific literature. We use the following example to demonstrate KBC from richly formatted data.

Example 1.1 (HasCollectorCurrent)

We highlight the ELECTRONICS domain. We are given a collection of transistor datasheets (like the one shown in Figure 1), and we want to build a KB of their maximum collector currents. [1] The output KB can power a tool that verifies that transistors do not exceed their maximum ratings in a circuit. Figure 1 shows how relevant information is located in both the document header and table cells and how their relationship is expressed using semantics from multiple modalities.

The heterogeneity of signals in richly formatted data poses a major challenge for existing KBC systems. The above example shows how KBC systems that focus on text data—and adjacent textual contexts such as sentences or paragraphs—can miss important information due to this breadth of signals in richly formatted data. We review the major challenges of KBC from richly formatted data.

Challenges—KBC on richly formatted data poses a number of challenges beyond those present with unstructured data: (1) accommodating *prevalent document-level relations*, (2) capturing the *multimodality* of information in the input data, and (3) addressing the tremendous *data variety*.

Prevalent Document-Level Relations: We define the *context* of a relation as the scope information that needs to be considered when extracting the relation. Context can range from a single sentence to a whole document. KBC systems typically limit the context to a few sentences or a single table, assuming that relations are expressed relatively locally. However, for richly formatted data, many relations rely on information from throughout a document to be extracted.

Example 1.2 (Document-Level Relations)

In Figure 1, transistor parts are located in the document header (boxed in blue), and the collector current value is in a table cell (boxed in green). Moreover, the interpretation of some numerical values depends on their units reported in another table column (e.g., 200 mA).

Limiting the context scope to a single sentence or table misses many potential relations—up to 97% in the ELECTRONICS application. On the other hand, considering all possible entity pairs throughout the document as candidates renders the extraction problem computationally intractable due to the combinatorial explosion of candidates.

---

[1] Transistors are semiconductor devices often used as switches or amplifiers. Their electrical specifications are published by manufacturers in datasheets.

Multimodality—Classical KBC systems model input data as unstructured text [23, 26, 36]. With richly formatted data, semantics are part of multiple modalities—textual, structural, tabular, and visual.

Example 1.3 (Multimodality)

In Figure 1, important information (e.g., the transistor names in the header) is expressed in larger, bold fonts (displayed in yellow). Furthermore, the meaning of a table entry depends on other entries with which it is visually or tabularly aligned (shown by the red arrow). For instance, the semantics of a numeric value is specified by an aligned unit.

Semantics from different modalities can vary significantly but can convey complementary information.

Data Variety—With richly formatted data, there are two primary sources of data variety: (1) format variety (e.g., file or table formatting) and (2) stylistic variety (e.g., linguistic variation).

Example 1.4 (Data Variety)

In Figure 1, numeric intervals are expressed as "−65 … 150," but other datasheets show intervals as "−65 ~ 150," or "−65 to 150." Similarly, tables can be formatted with a variety of spanning cells, header hierarchies, and layout orientations.

Data variety requires KBC systems to adopt data models that are generalizable and robust against heterogeneous input data.

Our Approach—We introduce `Fonduer`, a machine-learning-based system for KBC from richly formatted data. `Fonduer` takes as input richly formatted documents, which may be of diverse formats, including PDF, HTML, and XML. `Fonduer` parses the documents and analyzes the corresponding multimodal, document-level contexts to extract relations. The final output is a knowledge base with the relations classified to be correct. `Fonduer`'s machine-learning-based approach must tackle a series of technical challenges.

Technical Challenges: The challenges in designing `Fonduer` are:

1. Reasoning about relation candidates that are manifested in heterogeneous formats (e.g., text and tables) and span an entire document requires `Fonduer`'s machine-learning model to analyze heterogeneous, document-level context. While deep-learning models such as recurrent neural networks [2] are effective with sentence- or paragraph-level context [22], they fall short with document-level context, such as context that span both textual and visual features (e.g., information conveyed via fonts or alignment) [21]. Developing such models is an open challenge and active area of research [21].

2. The heterogeneity of contexts in richly formatted data magnifies the need for large amounts of training data. Manual annotation is prohibitively expensive, especially when domain expertise is required. At the same time, human-curated KBs, which can be used to generate training data, may exhibit low coverage or

not exist altogether. Alternatively, weak supervision sources can be used to programmatically create large training sets, but it is often unclear how to consistently apply these sources to richly formatted data. Whereas patterns in unstructured data can be identified based on text alone, expressing patterns consistently across different modalities in richly formatted data is challenging.

3. Considering candidates across an entire document leads to a combinatorial explosion of possible candidates, and thus random variables, which need to be considered during learning and inference. This leads to a fundamental tension between building a practical KBC system and learning accurate models that exhibit high recall. In addition, the combinatorial explosion of possible candidates results in a large class imbalance, where the number of "True" candidates is much smaller than the number of "False" candidates. Therefore, techniques that prune candidates to balance running time and endto- end quality are required.

Technical Contributions: Our main contributions are as follows:

1. To account for the breadth of signals in richly formatted data, we design a new *data model* that preserves structural and semantic information across different data modalities. The role of Fonduer's data model is twofold: (a) to allow users to specify multimodal domain knowledge that Fonduer leverages to automate the KBC process over richly formatted data, and (b) to provide Fonduer's machine-learning model with the necessary representation to reason about document-wide context (see Section 3).

2. We empirically show that existing deep-learning models [46] tailored for text information extraction (such as long short-term memory (LSTM) networks [18]) struggle to capture the multimodality of richly formatted data. We introduce a multimodal LSTM network that combines textual context with universal features that correspond to structural and visual properties of the input documents. These features are inherently captured by Fonduer's data model and are generated automatically (see Section 4.2). We also introduce a series of data layout optimizations to ensure the scalability of Fonduer to millions of document-wide candidates (see Appendix C).

3. Fonduer introduces a programming model in which no development cycles are spent on feature engineering. Users only need to specify *candidates*, the potential entries in the target KB, and provide lightweight *supervision* rules which capture a user's domain knowledge and programmatically label subsets of candidates, which are used for training Fonduer's deep-learning model (see Section 4.3). We conduct a user study to evaluate Fonduer's programming model. We find that when working with richly formatted data, users utilize the semantics from multiple modalities of the data, including both structural and textual information in the document. Our study demonstrates that given 30 minutes, Fonduer's programming model allows users to attain F1 scores that are 23 points higher than supervision via manual labeling candidates (see Section 6).

Summary of Results— Fonduer-based systems are in production in a range of academic and industrial uses cases, including a major online retailer. Fonduer introduces several advancements over prior KBC systems (see Appendix 7): (1) In contrast to prior systems that focus on adjacent textual data, Fonduer can extract document-level relations expressed in diverse formats, ranging from textual to tabular formats; (2) Fonduer reasons about multimodal context, i.e., both textual and visual characteristics of the input documents, to extract more accurate relations; (3) In contrast to prior KBC systems that rely heavily on feature engineering to achieve high quality [34], Fonduer obviates the need for feature engineering by extending a bidirectional LSTM—the de facto deep-learning standard in natural language processing [24]—to obtain a representation needed to automate relation extraction from richly formatted data. We evaluate Fonduer in four real-world applications of richly formatted information extraction and show that Fonduer enables users to build high-quality KBs, achieving an average improvement of 41 F1 points over state-of-the-art KBC systems.

2 BACKGROUND

We review the concepts and terminology used in the next sections.

2.1 Knowledge Base Construction

The input to a KBC system is a collection of documents. The output of the system is a relational database containing facts extracted from the input and stored in an appropriate schema. To describe the KBC process, we adopt the standard terminology from the KBC community. There are four types of objects that play integral roles in KBC systems: (1) *entities*, (2) *relations*, (3) *mentions* of entities, and (4) *relation mentions*.

An **entity** e in a knowledge base corresponds to a distinct real-world person, place, or object. Entities can be grouped into different **entity types** $T_1, T_2, \ldots, T_n$. Entities also participate in relationships. A relationship between n entities is represented as an n-ary **relation** R ($e_1, e_2, \ldots, e_n$) and is described by a **schema** $S_R(T_1, T_2, \ldots, T_n)$ where $e_i \in T_i$. A **mention** m is a span of text that refers to an entity. A relation mention **candidate** (referred to as a candidate in this paper) is an n-ary tuple c = ($m_1, m_2, \ldots, m_n$) that represents a potential instance of a relation R($e_1, e_2, \ldots, e_n$). A candidate classified as true is called a **relation mention**, denoted by r_R.

Example 2.1 (KBC)—Consider the HasCollectorCurrent task in Figure 1. Fonduer takes a corpus of transistor datasheets as input and constructs a KB containing the (Transistor Part, Current) binary **relation** as output. Parts like SMBT3904 and Currents like 200mA are **entities**. The spans of text that read "SMBT3904" and "200" (boxed in blue and green, respectively) are **mentions** of those two entities, and together they form a **candidate**. If the evidence in the document suggests that these two mentions are related, then the output KB will include the **relation mention** (SMBT3904, 200mA) of the HasCollectorCurrent relation.

The KBC problem is defined as follows:

Definition 2.2 (Knowledge Base Construction)—Given a set of documents D and a KB schema $S_R(T_1, T_2, \ldots, T_n)$, where each T_i corresponds to an entity type, extract a set of relation mentions r_R from D, which populate the schema's relational tables.

Like other machine-learning-based KBC systems [7, 36], `Fonduer` converts KBC to a statistical learning and inference problem: each candidate is assigned a Boolean random variable that can take the value "True" if the corresponding relation mention is correct, or "False" otherwise. In machine-learning-based KBC systems, each candidate is associated with certain features that provide evidence for the value that the corresponding random variable should take. Machine-learning-based KBC systems use machine learning to maximize the probability of correctly classifying candidates, given their features and ground truth examples.

2.2 Recurrent Neural Networks

The machine-learning model we use in `Fonduer` is based on a recurrent neural network (RNN). RNNs have obtained state-of-the-art results in many natural-language processing (NLP) tasks, including information extraction [15, 16, 43]. RNNs take sequential data as input. For each element in the input sequence, the information from previous inputs can affect the network output for the current element. For sequential data $\{x_1, \ldots, x_T\}$, the structure of an RNN is mathematically described as:

$$h_t = f(x_t, h_{t-1}), \quad y = g(\{h_1, \ldots, h_T\})$$

where h_t is the *hidden state* for element t, and y is the *representation* generated by the sequence of hidden states $\{h_1, \ldots, h_T\}$. Functions f and g are nonlinear transformations. For RNNs, we have that $f = \tanh(W_h x_t + U_h h_{t-1} + b_h)$ where W_h, U_h are parameter matrices and b_h is a vector. Function g is typically task-specific.

Long Short-term Memory—LSTM [18] networks are a special type of RNN that introduce new structures referred to as *gates*, which control the flow of information and can capture long-term dependencies. There are three types of gates: *input* gates i_t control which values are updated in a memory cell; *forget* gates f_t control which values remain in memory; and *output* gates o_t control which values in memory are used to compute the output of the cell. The final structure of an LSTM is given by:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

$$h_t = o_t \circ \tanh(c_t)$$

where c_t is the cell state vector, W, U, b are parameter matrices and a vector, σ is the sigmoid function, and $\circ$ is the Hadamard product.

Bidirectional LSTMs consist of forward and backward LSTMs. The forward LSTM f^F reads the sequence from x_1 to x_T and calculates a sequence of forward hidden states ($h_1^F, ..., h_T^F$). The backward LSTM f^B reads the sequence from x_T to x_1 and calculates a sequence of backward hidden states ($h_1^B, ..., h_T^B$). The final hidden state for the sequence is the concatenation of the forward and backward hidden states, e.g., $h_i = [h_i^F, h_i^B]$.

Attention—Previous work explored using pooling strategies to train an RNN, such as max pooling [41], which compresses the information contained in potentially long input sequences to a fixed-length internal representation by considering all parts of the input sequence impartially. This compression of information can make it difficult for RNNs to learn from long input sequences.

In recent years, the *attention* mechanism has been introduced to overcome this limitation by using a soft word-selection process that is conditioned on the global information of the sentence [2]. That is, rather than squashing all information from a source input (regardless of its length), this mechanism allows an RNN to pay more attention to the subsets of the input sequence where the most relevant information is concentrated.

`Fonduer` uses a bidirectional LSTM with attention to represent textual features of relation candidates from the documents. We extend this LSTM with features that capture other data modalities.

3 THE `Fonduer` FRAMEWORK

An overview of `Fonduer` is shown in Figure 2. `Fonduer` takes as input a collection of richly formatted documents and a collection of user inputs. It follows a machine-learning-based approach to extract relations from the input documents. The relations extracted by `Fonduer` are stored in a target knowledge base.

We introduce `Fonduer`'s data model for representing different properties of richly formatted data. We then review `Fonduer`'s data processing pipeline and describe the new programming paradigm introduced by `Fonduer` for KBC from richly formatted data.

The design of `Fonduer` was strongly guided by interactions with collaborators (see the user study in Section 6). We find that to support KBC from richly formatted data, a unified data model must:

- Serve as an abstraction for system and user interaction.

- Capture candidates that span different areas (e.g. sections of pages) and data modalities (e.g., textual and tabular data).

- Represent the formatting variety in richly formatted data sources in a unified manner.

Fonduer introduces a data model that satisfies these requirements.

3.1 Fonduer's Data Model

Fonduer's data model is a directed acyclic graph (DAG) that contains a hierarchy of contexts, whose structure reflects the intuitive hierarchy of document components. In this graph, each node is a *context* (represented as boxes in Figure 3). The root of the DAG is a *Document*, which contains *Section* contexts. Each *Section* is divided into: *Texts, Tables*, and *Figures. Texts* can contain multiple *Paragraphs; Tables* and *Figures* can contain *Captions; Tables* can also contain *Rows* and *Columns*, which are in turn made up of *Cells*. Each context ultimately breaks down into *Paragraphs* that are parsed into *Sentences*. In Figure 3, a downward edge indicates a parent-contains-child relationship. This hierarchy serves as an abstraction for both system and user interaction with the input corpus.

In addition, this data model allows us to capture candidates that come from different contexts within a document. For each context, we also store the textual contents, pointers to the parent contexts, and a wide range of attributes from each modality found in the original document. For example, standard NLP pre-processing tools are used to generate linguistic attributes, such as lemmas, parts of speech tags, named entity recognition tags, dependency paths, etc., for each *Sentence*. Structural and tabular attributes of a *Sentence*, such as tags, and row/column information, and parent attributes, can be captured by traversing its path in the data model. Visual attributes for the document are recorded by storing bounding box and page information for each word in a *Sentence*.

Example 3.1 (Data Model)—The data model representing the PDF in Figure 1 contains one *Section* with three children: a *Text* for the document header, a *Text* for the description, and a *Table* for the table itself (with 10 *Rows* and 4 *Columns*). Each *Cell* links to both a *Row* and *Column. Text*s and *Cell*s contain *Paragraphs* and *Sentences*.

Fonduer's multimodal data model unifies inputs of different formats, which addresses the data variety of richly formatted data that comes from variations in format. To construct the DAG for each document, we extract all the words in their original order. For structural and tabular information, we use tools such as Poppler[2] to convert an input file into HTML format; for visual information, such as coordinates and bounding boxes, we use a PDF printer to convert an input file into PDF format. If a conversion occurred, we associate the multimodal information in the converted file with all extracted words. We align the word sequences of the converted file with their originals by checking if both their characters and number of repeated occurrences before the current word are the same. Fonduer can recover

---

[2]https://poppler.freedesktop.org

from conversion errors by using the inherent redundancy in signals from other modalities. In addition, this DAG structure also simplifies the variation in format that comes from table formatting.

Takeaways: `Fonduer` consolidates a diverse variety of document formats, types of contexts, and modality semantics into one model in order to address variety inherent in richly formatted data. `Fonduer`'s data model serves as the formal representation of the intermediate data utilized in all future stages of the extraction process.

3.2 User Inputs and `Fonduer`'s Pipeline

The `Fonduer` processing pipeline follows three phases. We briefly describe each phase in turn and focus on the user inputs required by each phase. `Fonduer`'s internals are described in Section 4.

(1) KBC Initialization—The first phase in `Fonduer`'s pipeline is to initialize the target KB where the extracted relations will be stored. During this phase, `Fonduer` requires the user to specify a *target schema* that corresponds to the relations to be extracted. The target schema $S_R(T_1, \ldots, T_n)$ defines a relation R to be extracted from the input documents. An example of such a schema is provided below.

Example 3.2 (Relation Schema): An example SQL schema for the relation in Figure 1 is:

```
CREATE TABLE HasCollectorCurrent(
    TransistorPart varchar,
    Current varchar);
```

`Fonduer` uses the user-specified schema to initialize an empty relational database where the output KB will be stored. Furthermore, `Fonduer` iterates over its input *corpus* and transforms each document into an instance of `Fonduer`'s data model to capture the variety and multimodality of richly formatted documents.

(2) Candidate Generation—In this phase, `Fonduer` extracts relation candidates from the input documents. Here, users are required to provide two types of input functions: (1) *matchers* and (2) *throttlers*.

Matchers: To generate candidates for relation R, `Fonduer` requires that users define *matchers* for all distinct mention types in schema S_R. Matchers are how users specify what a mention looks like. In `Fonduer`, matchers are Python functions that accept a span of text as input—which has a reference to its data model—and output whether or not the match conditions are met. Matchers range from simple regular expressions to complicated functions that take into account signals across multiple modalities of the input data and can also incorporate existing methods such as named-entity recognition.

Example 3.3 (Matchers): From the HasCollectorCurrent relation in Figure 1, users define matchers for each type of the schema. A dictionary of valid transistor parts can be used as the first matcher. For maximum current, users can exploit the pattern that these values are commonly expressed as a numerical value between 100 and 995 for their second matcher.

```
# Use a dictionary to match transistor parts
def transistor_part_matcher(span):
  return 1 if span in part_dictionary else 0

# Use RegEx to extract numbers between [100, 995]
def max_current_matcher(span):
  return 1 if re.match('[1-9][0-9][0-5]', span) else 0
```

Throttlers: Users can optionally provide *throttlers*, which act as hard filtering rules to reduce the number of candidates that are materialized. Throttlers are also Python functions, but rather than accepting spans of text as input, they operate on candidates, and output whether or not a candidate meets the specified condition. Throttlers limit the number of *candidates* considered by Fonduer.

Example 3.4 (Throttler): Continuing the example shown in Figure 1, the user provides a throttler, which only keeps candidates whose Current has the word "Value" as its column header.

```
def value_in_column_header(cand):
  if 'Value' in header_ngrams(cand.current):
    return 1
  else:
    return 0
```

Given the input matchers and throttlers, Fonduer extracts relation candidates by traversing its data model representation of each document. By applying matchers to each leaf of the data model, Fonduer can generate sets of mentions for each component of the schema. The cross-product of these mentions produces candidates:

$$\text{Candidate}(\text{id}_{\text{candidate}}, \text{mention}_1, \ldots, \text{mention}_n)$$

where mentions are spans of text and contain pointers to their context in the data model of their respective document. The output of this phase is a set of candidates, C.

(3) Training a Multimodal LSTM for KBC—In this phase, Fonduer trains a multimodal LSTM network to *classify* the candidates generated during Phase 2 as "True" or "False" mentions of target relations. Fonduer's multimodal LSTM combines both visual and textual features. Recent work has also proposed the use of LSTMs for KBC but has focused only on textual data [46]. In Section 5.3.3, we experimentally demonstrate that state-of-the-art LSTMs struggle to capture the multimodal characteristics of richly formatted data, and thus, obtain poor-quality KBs.

Fonduer uses a bidirectional LSTM (reviewed in Section 2.2) to capture textual features and extends it with additional structural, tabular, and visual features captured by Fonduer's data model. The LSTM used by Fonduer is described in Section 4.2. Training in Fonduer is split into two sub-phases: (1) a multimodal featurization phase and (2) a phase where supervision data is provided by the user.

Multimodal Featurization: Here, Fonduer traverses its internal data model instance for each input document and automatically generates features that correspond to structural, tabular, and visual modalities as described in Section 4.2. These constitute a bare-bones

feature library (referred to as feature lib, below), which augments the textual features learned by the LSTM. All features are stored in a relation:

$$\text{Features}(\text{id}_{\text{candidate}}, \text{LSTM}_{\text{textual}}, \text{feature\_lib}_{\text{others}})$$

No user input is required during this step. `Fonduer` obviates the need for feature engineering and shows that incorporating multimodal information is key to achieving high-quality relation extraction.

Supervision: To train its multimodal LSTM, `Fonduer` requires that users provide some form of supervision. Collecting sufficient training data for multi-context deep-learning models is a well-established challenge. As stated by LeCun et al. [21], taking into account a context of more than a handful of words for text-based deep-learning models requires very large training corpora.

To soften the burden of traditional supervision, `Fonduer` uses a supervision paradigm referred to as *data programming* [33]. Data programming is a human-in-the-loop paradigm for training machine-learning systems. In data programming, users only need to specify lightweight functions, referred to as *labeling functions* (LFs), that programmatically assign labels to the input candidates. A detailed overview of data programming is provided in Appendix A. While existing work on data programming [32] has focused on labeling functions over textual data, `Fonduer` paves the way for specifying labeling functions over richly formatted data.

`Fonduer` requires that users specify labeling functions that label the candidates from Phase 2. Labeling functions in `Fonduer` are Python functions that take a candidate as input and assign +1 to label it as "True," −1 to label it as "False," or 0 to abstain.

Example 3.5 (Labeling Functions): Looking at the datasheet in Figure 1, users can express patterns such as having the Part and Current y-aligned on the visual rendering of the page. Similarly, users can write a rule that labels a candidate whose Current is in the same row as the word "current" as "True."

```
# Rule-based LF based on visual information
def y_axis_aligned(cand):
    return 1 if cand.part.y == cand.current.y else 0

# Rule-based LF based on tabular content
def has_current_in_row(cand):
    if 'current' in row_ngrams(cand.current):
        return 1
    else:
        return 0
```

As shown in Example 3.5, `Fonduer`'s internal data model allows users to specify labeling functions that capture supervision patterns across any modality of the data (see Section 4.3). In our user study, we find that it is common for users to write labeling functions that span multiple modalities and consider both textual and visual patterns of the input data (see Section 6).

The user-specified labeling functions, together with the candidates generated by `Fonduer`, are passed as input to Snorkel [32], a data-programming engine, which converts the noisy

labels generated by the input labeling functions to denoised labeled data used to train `Fonduer`'s multimodal LSTM model (see Appendix A).

Classification: `Fonduer` uses its trained LSTM to assign a marginal probability to each candidate. The last layer of `Fonduer`'s LSTM is a softmax classifier (described in Section 4.2) that computes the probability of a candidate being a "True" relation. In `Fonduer`, users can specify a *threshold* over the output marginal probabilities to determine which candidates will be classified as "True" (those whose marginal probability of being true exceeds the specified threshold) and which are "False" (those whose marginal probability fall beneath the threshold). This threshold depends on the requirements of the application. Applications that require critically high accuracy can set a high threshold value to ensure only candidates with a high probability of being "True" are classified as such.

As shown in Figure 2, supervision and classification are typically executed over several iterations as users develop a KBC application. This feedback loop allows users to quickly receive feedback and improve their labeling functions, and avoids the overhead of rerunning candidate extraction and materializing features (see Section 6).

3.3 `Fonduer`'s Programming Model for KBC

`Fonduer` is the first system to provide the necessary abstractions and mechanisms to enable the use of weak supervision as a means to train a KBC system for richly formatted data. Traditionally, machine-learning-based KBC focuses on feature engineering to obtain high-quality KBs. This requires that users rerun feature extraction, learning, and inference after every modification of the features used during KBC. With `Fonduer`'s machine-learning approach, features are generated automatically. This puts emphasis on (1) specifying the relation candidates and (2) providing multimodal supervision rules via labeling functions. This approach allows users to leverage multiple sources of supervision to address data variety introduced by variations in style better than traditional manual labeling [36].

`Fonduer`'s programming paradigm obviates the need for feature engineering and introduces two modes of operation for `Fonduer` applications: (1) development and (2) production. During development, labeling functions are iteratively improved, in terms of both coverage and accuracy, through error analysis as shown by the blue arrows in Figure 2. LFs are applied to a small sample of labeled candidates and evaluated by the user on their accuracy and coverage (the fraction of candidates receiving non-zero labels). To support efficient error analysis, `Fonduer` enables users to easily inspect the resulting candidates and provides a set of labeling function metrics, such as coverage, conflict, and overlap, which provide users with a rough assessment of how to improve their LFs. In practice, approximately 20 iterations are adequate for our users to generate a sufficiently tuned set of labeling functions (see Section 6). In production, the finalized LFs are applied to the entire set of candidates, and learning and inference are performed *only once* to generate the final KB.

On average, only a small number of labeling functions are needed to achieve high-quality KBC (see Section 6). For example, in the ELECTRONICS application, 16 labeling functions, on average, are sufficient to achieve an average F1 score of over 75. Furthermore, we find that

tabular and visual signals are particularly valuable forms of supervision for KBC from richly formatted data, and complementary to traditional textual signals (see Section 6).

4 KBC IN `Fonduer`

Here, we focus on the implementation of each component of `Fonduer`. In Appendix C we discuss a series of optimizations that enable `Fonduer`'s scalability to millions of candidates.

4.1 Candidate Generation

Candidate generation from richly formatted data relies on access to document-level contexts, which is provided by `Fonduer`'s data model. Due to the significantly increased context needed for KBC from richly formatted data, naïvely materializing all possible candidates is intractable as the number of candidates grows combinatorially with the number of relation arguments. This combinatorial explosion can lead to performance issues for KBC systems. For example, in the ELECTRONICS domain, just 100 documents can generate over 1M candidates. In addition, we find that the majority of these candidates do not express true relations, creating a significant class imbalance that can hinder learning performance [19].

To address this combinatorial explosion, `Fonduer` allows users to specify throttlers, in addition to matchers, to prune away excess candidates. We find that throttlers must:

- Maintain high accuracy by only filtering negative candidates.

- Seek high coverage of the candidates.

Throttlers can be viewed as a knob that allows users to trade off precision and recall and promote scalability by reducing the number of candidates to be classified during KBC.

Figure 4 shows how using throttlers affects the quality-performance tradeoff in the ELECTRONICS domain. We see that throttling significantly improves system performance. However, increased throttling does not monotonically improve quality since it hurts recall. This tradeoff captures the fundamental tension between optimizing for system performance and optimizing for end-to-end quality. When no candidates are pruned, the class imbalance resulting from many negative candidates to the relatively small number of positive candidates harms quality. Therefore, as a rule of thumb, we recommend that users apply throttlers to balance negative and positive candidates. `Fonduer` provides users with mechanisms to evaluate this balance over a small holdout set of labeled candidates.

Takeaways— `Fonduer`'s data model is necessary to perform candidate generation with richly formatted data. Pruning negative candidates via throttlers to balance negative and positive candidates not only ensures the scalability of `Fonduer` but also improves the precision of `Fonduer`'s output.

4.2 Multimodal LSTM Model

We now describe `Fonduer`'s deep-learning model in detail. `Fonduer`'s model extends a bidirectional LSTM (Bi-LSTM), the de facto deeplearning standard for NLP [24], with a simple set of dynamically generated features that capture semantics from the structural,

tabular, and visual modalities of the data model. A detailed list of these features is provided in Appendix B. In Section 5.3.2, we perform an ablation study demonstrating that non-textual features are key to obtaining high-quality KBs. We find that the quality of the output KB deteriorates up to 33 F1 points when non-textual features are removed. Figure 5 illustrates Fonduer's LSTM. We now review each component of Fonduer's LSTM.

Bidirectional LSTM with Attention—Traditionally, the primary source of signal for relation extraction comes from unstructured text. In order to understand textual signals, Fonduer uses an LSTM network to extract textual features. For mentions, Fonduer builds a Bi-LSTM to get the textual features of the mention from both directions of sentences containing the candidate. For sentence s_i containing the ith mention in the document, the textual features h_{ik} of each word w_{ik} are encoded by both forward (defined as superscript F in equations) and backward (defined as superscript B) LSTM, which summarizes information about the whole sentence with a focus on w_{ik}. This takes the structure:

$$h_{ik}^F = \text{LSTM}(h_{i(k-1)}^F, \Phi(s_i, k))$$

$$h_{ik}^B = \text{LSTM}(h_{i(k+1)}^B, \Phi(s_i, k))$$

$$h_{ik} = [h_{ik}^F, h_{ik}^B]$$

where $\Phi(s_i, k)$ is the word embedding [40], which is the representation of the semantics of the kth word in sentence s_i.

Then, the textual feature representation for a mention, t_i, is calculated by the following attention mechanism to model the importance of different words from the sentence s_i and to aggregate the feature representation of those words to form a final feature representation,

$$u_{ik} = \tanh(W_w h_{ik} + b_w)$$

$$\alpha_{ik} = \frac{\exp(u_{ik}^T u_w)}{\sum_j \exp(u_{ij}^T u_w)}$$

$$t_i = \sum_j \alpha_{ij} u_{ij}$$

where W_w, u_w, and b are parameter matrices and a vector. u_{ik} is the hidden representation of h_{ik}, and α_{ik} is to model the importance of each word in the sentence s_i. Special candidate markers (shown in red in Figure 5) are added to the sentences to draw *attention* to the

candidates themselves. Finally, the textual features of a candidate are the concatenation of its mentions' textual features $[t_1, \ldots, t_n]$.

Extended Feature Library—Features for structural, tabular, and visual modalities are generated by leveraging the data model, which preserves each modality's semantics. For each candidate, such as the candidate (SMBT3904, 200) shown in Figure 5, Fonduer locates each mention in the data model and traverses the DAG to compute features from the modality information stored in the nodes of the graph. For example, Fonduer can traverse sibling nodes to add tabular features such as featurizing a node based on the other mentions in the same row or column. Similarly, Fonduer can traverse the data model to extract structural features from tags stored while parsing the document along with the hierarchy of the document elements themselves. We review each modality:

Structural features: These provide signals intrinsic to a document's structure. These features are dynamically generated and allow Fonduer to learn from structural attributes, such as parent and sibling relationships and XML/HTML tag metadata found in the data model (shown in yellow in Figure 5). The data model also allows Fonduer to track structural distances of candidates, which helps when a candidate's mentions are visually distant, but structurally close together. Specifically, featurizing a candidate with the distance to the lowest common ancestor in the data model is a positive signal for linking table captions to table contents.

Tabular features: These are a special subset of structural features since tables are very common structures inside documents and have high information density. Table features are drawn from the grid-like representation of rows and columns stored in the data model, shown in green in Figure 5. In addition to the tabular location of mentions, Fonduer also featurizes candidates with signals such as being in the same row or column. For example, consider a table that has cells with multiple lines of text; recording that two mentions share a row captures a signal that a visual alignment feature could easily miss.

Visual features: These provide signals observed from a visual rendering of a document. In cases where tabular or structural features are noisy—including nearly all documents converted from PDF to HTML by generic tools—visual features can provide a complementary view of the dependencies among text. Visual features encode many highly predictive types of semantic information implicitly, such as position on a page, which may imply when text is a title or header. An example of this is shown in red in Figure 5.

Training—All parameters of Fonduer's LSTM are jointly trained, including the parameters of the Bi-LSTM as well as the weights of the last softmax layer that correspond to additional features.

Takeaways: To achieve high-quality KBC with richly formatted data, it is vital to have features from multiple data modalities. These features are only obtainable through traversing and accessing modality attributes stored in the data model.

4.3 Multimodal Supervision

Unlike KBC from unstructured text, KBC from richly formatted data requires supervision from multiple modalities of the data. In richly formatted data, useful patterns for KBC are more sparse and hidden in non-textual signals, which motivates the need to exploit overlap and repetition in a variety of patterns over multiple modalities. Fonduer's data model allows users to directly express correctness using textual, structural, tabular, or visual characteristics, in addition to traditional supervision sources like existing KBs. In the Electronics domain, over 70% of labeling functions written by our users are based on non-textual signals. It is acceptable for these labeling functions to be noisy and conflict with one another. Data programming theory (see Appendix A.2) shows that, with a sufficient number of labeling functions, data programming can still achieve quality comparable to using manually labeled data.

In Section 5.3.4, we find that using metadata in the Electronics domain, such as structural, tabular, and visual cues, results in a 66 F1 point increase over using textual supervision sources alone. Using both sources gives a further increase of 2 F1 points over metadata alone. We also show that supervision using information from all modalities, rather than textual information alone, results in an increase of 43 F1 points, on average, over a variety of domains. Using multiple supervision sources is crucial to achieving high-quality information extraction from richly formatted data.

Takeaways—Supervision using multiple modalities of richly formatted data is key to achieving high end-to-end quality. Like multimodal featurization, multimodal supervision is also enabled by Fonduer's data model and addresses stylistic data variety.

5 EXPERIMENTS

We evaluate Fonduer over four applications: Electronics, Advertisements, Paleontology, and Genomics—each containing several relation extraction tasks. We seek to answer: (1) how does Fonduer compare against both state-of-the-art KBC techniques and manually curated knowledge bases? and (2) how does each component of Fonduer contribute to end-to-end extraction quality?

5.1 Experimental Settings

Datasets—The datasets used for evaluation vary in size and format. Table 1 shows a summary of these datasets.

Electronics: The Electronics dataset is a collection of single bipolar transistor specification datasheets from over 20 manufacturers, downloaded from Digi-Key.[3] These documents consist primarily of tables and express relations containing domain-specific symbols. We focus on the relations between transistor part numbers and several of their electrical characteristics. *We use this dataset to evaluate* Fonduer *with respect to datasets that consist primarily of tables and numerical data.*

---

[3]https://www.digikey.com

Advertisements: The ADVERTISEMENTS dataset contains webpages that may contain evidence of human trafficking activity. These webpages may provide prices of services, locations, contact information, physical characteristics of the victims, etc. Here, we extract all attributes associated with a trafficking advertisement. The output is deployed in production and is used by law enforcement agencies. This is a heterogeneous dataset containing millions of webpages over 692 web domains in which users create customized ads, resulting in 100,000s of unique layouts. *We use this dataset to examine the robustness of* Fonduer *in the presence of significant data variety*.

Paleontology: The PALEONTOLOGY dataset is a collection of well-curated paleontology journal articles on fossils and ancient organisms. Here, we extract relations between paleontological discoveries and their corresponding physical measurements. These papers often contain tables spanning multiple pages. Thus, achieving high quality in this application requires linking content in tables to the text that references it, which can be separated by 20 pages or more in the document. *We use this dataset to test* Fonduer*'s ability to draw candidates from document-level contexts*.

Genomics: The GENOMICS dataset is a collection of open-access biomedical papers on gene-wide association studies (GWAS) from the manually curated GWAS Catalog [42]. Here, we extract relations between single-nucleotide polymorphisms and human phenotypes found to be statistically significant. This dataset is published in XML format, thus, we do not have visual representations. *We use this dataset to evaluate how well the* Fonduer *framework extracts relations from data that is published natively in a tree-based format*.

Comparison Methods—We use two different methods to evaluate the quality of Fonduer's output: the upper bound of state-of-the-art KBC systems (*Oracle*) and manually curated knowledge bases (*Existing Knowledge Bases*).

Oracle: Existing state-of-the-art information extraction (IE) methods focus on either *textual data* or *semi-structured and tabular data*. We compare Fonduer against both types of IE methods. Each IE method can be split into (1) a candidate generation stage and (2) a filtering stage, the latter of which eliminates false positive candidates. For comparison, we approximate the upper bound of quality of three state-of-the-art information extraction techniques by experimentally measuring the recall achieved in the candidate generation stage of each technique and assuming that all candidates found using a particular technique are correct. That is, we assume the filtering stage is perfect by assuming a precision of 1.0.

- **Text:** We consider IE methods over text [23, 36]. Here, candidates are extracted from individual sentences, which are pre-processed with standard NLP tools to add part-of-speech tags, linguistic parsing information, etc.

- **Table:** For tables, we use an IE method for semi-structured data [3]. Candidates are drawn from individual tables by utilizing table content and structure.

- **Ensemble:** We also implement an ensemble (proposed in [9]) as the union of candidates generated by **Text** and **Table**.

Existing Knowledge Base: We use existing knowledge bases as another comparison method. The ELECTRONICS application is compared against the transistor specifications published by Digi-Key, while GENOMICS is compared to both GWAS Central [4] and GWAS Catalog [42], which are the most comprehensive collections of GWAS data and widely-used public datasets. Knowledge bases such as these are constructed using a combination of manual entry, web aggregation, paid third-party services, and automation tools.

Fonduer Details— Fonduer is implemented in Python, with database operations being handled by PostgreSQL. All experiments are executed in Jupyter Notebooks on a machine with four CPUs (each CPU is a 14-core 2.40 GHz Xeon E5–4657L), 1 TB RAM, and 12×3TB hard drives, with the Ubuntu 14.04 operating system.

5.2 Experimental Results

5.2.1 Oracle Comparison—We compare the end-to-end quality of Fonduer to the upper bound of state-of-the-art systems. In Table 2, we see that Fonduer outperforms these upper bounds for each dataset. In ELECTRONICS, Fonduer results in a significant improvement of 71 F1 points over a text-only approach. In contrast, ADVERTISEMENTS has a higher upper bound with text than with tables, which reflects how advertisements rely more on text than the largely numerical tables found in ELECTRONICS. In the PALEONTOLOGY dataset, which depends on linking references from text to tables, the unified approach of Fonduer results in an increase of 43 F1 points over the Ensemble baseline. In GENOMICS, all candidates are cross-context, preventing both the text-only and the table-only approaches from finding any valid candidates.

5.2.2 Existing Knowledge Base Comparison—We now compare Fonduer against existing knowledge bases for ELECTRONICS and GENOMICS. No manually curated KBs are available for the other two datasets. In Table 3, we find that Fonduer achieves high coverage of the existing knowledge bases, while also correctly extracting novel relation entries with over 85% accuracy in both applications. In ELECTRONICS, Fonduer achieved 99% coverage and extracted an additional 17 correct entries not found in Digi-Key's catalog. In the GENOMICS application, we see that Fonduer provides over 80% coverage of both existing KBs and finds 1.87× and 1.42× more correct entries than GWAS Central and GWAS Catalog, respectively.

Takeaways: Fonduer achieves over 41 F1 points higher quality on average when compared against the upper bound of state-of-the-art approaches. Furthermore, Fonduer attains over 80% of the data in existing public knowledge bases while providing up to 1.87× the number of correct entries with high accuracy.

5.3 Ablation Studies

We conduct ablation studies to assess the effect of context scope, multimodal features, featurization approaches, and multimodal supervision on the quality of Fonduer. In each study, we change one component of Fonduer and hold the others constant.

5.3.1 Context Scope Study—To evaluate the importance of addressing the non-local nature of candidates in richly formatted data, we analyze how the different context scopes contribute to end-to-end quality. We limit the extracted candidates to four levels of context scope in Electronics and report the average F1 score for each. Figure 6 shows that increasing context scope can significantly improve the F1 score. Considering document context gives an additional 71 F1 points (12.8×) over sentence contexts and 47 F1 points (2.6×) over table contexts. The positive correlation between quality and context scope matches our expectations, since larger context scope is required to form candidates jointly from both table content and surrounding text. We see a smaller increase of 11 F1 points (1.2×) in quality between page and document contexts since many of the Electronics relation mentions are presented on the first page of the document.

Takeaways: Semantics can be distributed in a document or implied in its structure, thus requiring larger context scope than the traditional sentence-level contexts used in previous KBC systems.

5.3.2 Feature Ablation Study—We evaluate `Fonduer`'s multimodal features. We analyze how different features benefit information extraction from richly formatted data by comparing the effects of disabling one feature type while leaving all other types enabled, and report the average F1 scores of each configuration in Figure 7.

We find that removing a single feature set resulted in drops of 2 F1 points (no textual features in Paleontology) to 33 F1 points (no textual features in Advertisements). While it is clear in Figure 7 that each application depends on different feature types, we find that it is necessary to incorporate all feature types to achieve the highest extraction quality.

The characteristics of each dataset affect how valuable each feature type is to relation classification. The Advertisements dataset consists of webpages that often use tables to format and organize information—many relations can be found within the same cell or phrase. This heavy reliance on textual features is reflected by the drop of 33 F1 points when textual features are disabled. In Electronics, both components of the (part, attribute) tuples we extract are often isolated from other text. Hence, we see a small drop of 5 F1 points when textual features are disabled. We see a drop of 21 F1 points when structural features are disabled in the Paleontology application due to its reliance on structural features to link between formation names (found in text sections or table captions) and the table itself. Finally, we see similar decreases when disabling structural and tabular features in the Genomics application (24 and 29 F1 points, respectively). Because this dataset is published natively in XML, structural and tabular features are almost perfectly parsed, which results in similar impacts of these features.

Takeaways: It is necessary to utilize multimodal features to provide a robust, domain-agnostic description for real-world data.

5.3.3 Featurization Study—We compare `Fonduer`'s multimodal featurization with: (1) a human-tuned multimodal feature library that leverages `Fonduer`'s data model, requiring feature engineering; (2) a Bi-LSTM with attention model; this RNN considers textual

features only; (3) a machine-learning-based system for information extraction, referred to as SRV, which relies on HTML features [11]; and (4) a document-level RNN [22], which learns a representation over all available modes of information captured by `Fonduer`'s data model. We find that:

- `Fonduer`'s automatic multimodal featurization approach produces results that are comparable to manually-tuned feature representations requiring feature engineering. `Fonduer`'s neural network is able to extract relations with a quality comparable to the human-tuned approach in all datasets differing by no more than 2 F1 points (see Table 4).

- `Fonduer`'s RNN outperforms a standard, out-of-the-box Bi-LSTM significantly. The F1-score obtained by `Fonduer`'s multimodal RNN model is 1.7× to 2.2× higher than that of a typical Bi-LSTM (see Table 4).

- `Fonduer` outperforms extraction systems that leverage HTML features alone. Table 5 shows a comparison between `Fonduer` and SRV [11] in the ADVERTISEMENTS domain—the only one of our datasets with HTML documents as input. `Fonduer`'s features capture more information than SRV's HTML-based features, which only capture structural and textual information. This results in 2.3× higher quality.

- Using a document-level RNN to learn a single representation across all possible modalities results in neural networks with structures that are too large and too unique to batch effectively. This leads to slow runtime during training and poor-quality KBs. In Table 6, we compare the performance of a document-level RNN [22] and `Fonduer`'s approach of appending non-textual information in the last layer of the model. As shown `Fonduer`'s multimodal RNN obtains an F1-score that is almost 3× higher while being three orders of magnitude faster to train.

Takeaways: Direct feature engineering is unnecessary when utilizing deep learning as a basis to obtain the feature representation needed to extract relations from richly formatted data.

5.3.4 Supervision Ablation Study—We study how quality is affected when using only textual LFs, only metadata LFs, and the combination of the two sets of LFs. Textual LFs only operate on textual modality characteristics while metadata LFs operate on structural, tabular, and visual modality characteristics. Figure 8 shows that applying metadata-based LFs achieves higher quality than traditional textual-level LFs alone. The highest quality is achieved when both types of LFs are used. In ELECTRONICS, we see an increase of 66 F1 points (9.2×) when using metadata LFs and a 3 F1 point (1.04×) improvement over metadata LFs when both types are used. Because this dataset relies more heavily on distant signals, LFs that can label correctness based on column or row header content significantly improve extraction quality. The ADVERTISEMENTS application benefits equally from metadata and textual LFs. Yet, we get an increase of 20 F1 points (1.2×) when both types of LFs are applied. The PALEONTOLOGY and GENOMICS applications show more moderate increases of 40 (4.6×) and 40 (1.8×) F1 points by using both types over only textual LFs, respectively.

6 USER STUDY

Traditionally, ground truth data is created through manual annotation, crowdsourcing, or other time-consuming methods and then used as data for training a machine-learning model. In Fonduer, we use the data-programming model for users to programmatically generate training data, rather than needing to perform manual annotation—a human-in-the-loop approach. In this section we qualitatively evaluate the effectiveness of our approach compared to traditional human labeling and observe the extent to which users leverage non-textual semantics when labeling candidates.

We conducted a user study with 10 users, where each user was asked to complete the relation extraction task of extracting maximum collector-emitter voltages from the ELECTRONICS dataset. Using the same experimental settings, we compare the effectiveness of two approaches for obtaining training data: (1) manual annotations (Manual) and (2) using labeling functions (LF). We selected users with a basic knowledge of Python but no expertise in the ELECTRONICS domain. Users completed a 20 minute walk-through to familiarize themselves with the interface and procedures. To minimize the effect of cognitive fatigue and familiarity with the task, half of the users performed the task of manually annotating training data first, then the task of writing labeling functions, while the other half performed the tasks in the reverse order. We allotted 30 minutes for each task and evaluated the quality that was achieved using each approach at several checkpoints. For manual annotations, we evaluated every five minutes. We plotted the quality achieved by user's labeling functions each time the user performed an iteration of supervision and classification as part of Fonduer's iterative approach. We filtered out two outliers and report results of eight users.

In Figure 9 (left), we report the quality (F1 score) achieved by the two different approaches. The average F1 achieved using manual annotation was 0.26 while the average F1 score using labeling functions was 0.49, an improvement of 1.9×. We found with statistical significance that all users were able to achieve higher F1 scores using labeling functions than manually annotating candidates, regardless of the order in which they performed the approaches. There are two primary reasons for this trend. First, labeling functions provide a larger set of training data than manual annotations by enabling users to apply patterns they find in the data programmatically to all candidates—a natural desire they often vocalized while performing manual annotation. On average, our users manually labeled 285 candidates in the allotted time, while the labeling functions they created labeled 19,075 candidates. Users provided seven labeling functions on average. Second, labeling functions tend to allow Fonduer to learn more generic features, whereas manual annotations may not adequately cover the characteristics of the dataset as a whole. For example, labeling functions are easily applied to new data.

In addition, we found that for richly formatted data, users relied less on textual information —a primary signal in traditional KBC tasks—and more on information from other modalities, as shown in Figure 9 (right). Users utilized the semantics from multiple modalities of the richly formatted data, with 58.5% of their labeling functions using tabular information. This reflects the characteristics of the ELECTRONICS dataset, which contains

information that is primarily found in tables. In our study, the most common labeling functions in each modality were:

- **Tabular:** labeling a candidate based on the words found in the same row or column.

- **Visual:** labeling a candidate based on its placement in a document (e.g., which page it was found on).

- **Structural:** labeling a candidate based on its tag names.

- **Textual:** labeling a candidate based on the textual characteristics of the voltage mention (e.g. magnitude).

Takeaways

We found that when working with richly formatted data, users relied heavily on non-textual signals to identify candidates and weakly supervise the KBC system. Furthermore, leveraging weak supervision allowed users to create knowledge bases more effectively than traditional manual annotations alone.

7 RELATED WORK

We briefly review prior work in a few categories.

Context Scope

Existing KBC systems often restrict candidates to specific context scopes such as single sentences [23, 44] or tables [7]. Others perform KBC from richly formatted data by ensembling candidates discovered using separate extraction tasks [9, 14], which overlooks candidates composed of mentions that must be found jointly from document-level context scopes.

Multimodality

In unstructured data information extraction systems, only textual features [26] are utilized. Recognizing the need to represent layout information as well when working with richly formatted data, various additional feature libraries have been proposed. Some have relied predominantly on structural features, usually in the context of web tables [11, 30, 31, 39]. Others have built systems that rely only on visual information [13, 45]. There have been instances of visual information being used to supplement a tree-based representation of a document [8, 20], but these systems were designed for other tasks, such as document classification and page segmentation. By utilizing our deep-learning-based featurization approach, which supports all of these representations, Fonduer obviates the need to focus on feature engineering and frees the user to iterate over the supervision and learning stages of the framework.

Supervision Sources

Distant supervision is one effective way to programmatically create training data for use in machine learning. In this paradigm, facts from existing knowledge bases are paired with

unlabeled documents to create noisy or "weakly" labeled training examples [1, 25, 26, 28]. In addition to existing knowledge bases, crowdsourcing [12] and heuristics from domain experts [29] have also proven to be effective weak supervision sources. In our work, we show that by incorporating all kinds of supervision in one framework in a noise-aware way, we are able to achieve high quality in knowledge base construction. Furthermore, through our programming model, we empower users to add supervision based on intuition from any modality of data.

8 CONCLUSION

In this paper, we study how to extract information from richly formatted data. We show that the key challenges of this problem are (1) prevalent document-level relations, (2) multimodality, and (3) data variety. To address these, we propose Fonduer, the first KBC system for richly formatted information extraction. We describe Fonduer's data model, which enables users to perform candidate extraction, multimodal featurization, and multimodal supervision through a simple programming model. We evaluate Fonduer on four real-world domains and show an average improvement of 41 F1 points over the upper bound of state-of-the-art approaches. In some domains, Fonduer extracts up to $1.87\times$ the number of correct relations compared to expert-curated public knowledge bases.

Acknowledgments

References

1. Angeli G, Gupta S, Jose M, Manning CD, Ré C, Tibshirani J, Wu JY, Wu S, Zhang C. Stanford's 2014 slot filling systems. TAC KBP. 2014:695.

2. Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473. 2014

3. Barowy, DW., Gulwani, S., Hart, T., Zorn, B. ACM SIGPLAN Notices. Vol. 50. ACM; 2015. Flashrelate: extracting relational data from semi-structured spreadsheets using examples. In; p. 218-228.

4. Beck T, Hastings RK, Gollapudi S, Free RC, Brookes AJ. Gwas central: a comprehensive resource for the comparison and interrogation of genome-wide association studies. EJHG. 2014; 22(7):949–952. [PubMed: 24301061]

5. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J. SIGMOD. ACM; 2008. Freebase: a collaboratively created graph database for structuring human knowledge; p. 1247-1250.

6. Brown E, Epstein E, Murdock JW, Fin T-H. Tools and methods for building watson. IBM Research. Abgerufen am. 2013; 14:2013.

7. Carlson A, Betteridge J, Kisiel B, Settles B, Hruschka ER Jr, Mitchell TM. Toward an architecture for never-ending language learning. AAAI. 2010; 5:3.

8. Cosulschi M, Constantinescu N, Gabroveanu M. Classifcation and comparison of information structures from a web page. Annals of the University of Craiova-Mathematics and Computer Science Series. 2004; 31

9. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W. SIGKDD. ACM; 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion; p. 601-610.

10. Ferrucci D, Brown E, Chu-Carroll J, Fan J, Gondek D, Kalyanpur AA, Lally A, Murdock JW, Nyberg E, Prager J, et al. Building watson: An overview of the deepqa project. AI magazine. 2010; 31(3):59–79.

11. Freitag D. Information extraction from html: Application of a general machine learning approach. AAAI/IAAI. 1998:517–523.

12. Gao H, Barbier G, Goolsby R. Harnessing the crowdsourcing power of social media for disaster relief. IEEE Intelligent Systems. 2011; 26(3):10–14.

13. Gatterbauer W, Bohunsky P, Herzog M, Krüpl B, Pollak B. Towards domain-independent information extraction from web tables. WWW. 2007:71–80.

14. Govindaraju V, Zhang C, Ré C. Understanding tables in context using standard nlp toolkits. ACL. 2013:658–664.

15. Graves A, Liwicki M, Fernández S, Bertolami R, Bunke H, Schmidhuber J. A novel connectionist system for unconstrained handwriting recognition. IEEE transactions on pattern analysis and machine intelligence. 2009; 31(5):855–868. [PubMed: 19299860]

16. Graves, A., Mohamed, A-r, Hinton, G. Acoustics, speech and signal processing (icassp), 2013 ieee international conference on. IEEE; 2013. Speech recognition with deep recurrent neural networks; p. 6645-6649.

17. Hewett M, Oliver DE, Rubin DL, Easton KL, Stuart JM, Altman RB, Klein TE. Pharmgkb: the pharmacogenetics knowledge base. Nucleic acids research. 2002; 30(1):163–165. [PubMed: 11752281]

18. Hochreiter S, Schmidhuber J. Long short-term memory. Neural computation. 1997; 9(8):1735–1780. [PubMed: 9377276]

19. Japkowicz N, Stephen S. The class imbalance problem: A systematic study. Intelligent data analysis. 2002; 6(5):429–449.

20. Kovacevic M, Diligenti M, Gori M, Milutinovic V. Recognition of common areas in a web page using visual information: a possible application in a page classification. ICDM. 2002:250–257.

21. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015; 521(7553):436–444. [PubMed: 26017442]

22. Li J, Luong T, Jurafsky D. A hierarchical neural autoencoder for paragraphs and documents. ACL. 2015:1106–1115.

23. Madaan A, Mittal A, Mausam GR, Ramakrishnan G, Sarawagi S. Numerical relation extraction with minimal supervision. AAAI. 2016:2764–2771.

24. Manning, C. Representations for language: From word embeddings to sentence meanings. 2017. https://simons.berkeley.edu/talks/christopher-manning-2017-3-27

25. Min B, Grishman R, Wan L, Wang C, Gondek D. Distant supervision for relation extraction with an incomplete knowledge base. HLT-NAACL. 2013:777–782.

26. Mintz M, Bills S, Snow R, Jurafsky D. Distant supervision for relation extraction without labeled data. ACL. 2009:1003–1011.

27. Nakashole, N., Theobald, M., Weikum, G. WSDM. ACM; 2011. Scalable knowledge harvesting with high precision and high recall; p. 227-236.

28. Nguyen T-VT, Moschitti A. End-to-end relation extraction using distant supervision from external semantic repositories. HLT. 2011:277–282.

29. Pasupat P, Liang P. Zero-shot entity extraction from web pages. ACL (1). 2014:391–401.

30. Penn G, Hu J, Luo H, McDonald RT. Flexible web document analysis for delivery to narrow-bandwidth devices. ICDAR. 2001; 1:1074–1078.

31. Pinto D, Branstein M, Coleman R, Croft WB, King M, Li W, Wei X. Quasm: a system for question answering using semi-structured data. JCDL. 2002:46–55.

32. Ratner A, Bach SH, Ehrenberg H, Fries J, Wu S, Ré C. Snorkel: Rapid training data creation with weak supervision. VLDB. 2017; 11(3):269–282.

33. Ratner A, De Sa CM, Wu S, Selsam D, Ré C. Data programming: Creating large training sets, quickly. NIPS. 2016:3567–3575.

34. Ré C, Sadeghian AA, Shan Z, Shin J, Wang F, Wu S, Zhang C. Feature engineering for knowledge base construction. IEEE Data Engineering Bulletin. 2014

35. Settles B. Active learning literature survey. 2010. Computer Sciences Technical Report. 1648

36. Shin J, Wu S, Wang F, De Sa C, Zhang C, Ré C. Incremental knowledge base construction using deepdive. VLDB. 2015; 8(11):1310–1321.

37. Singhal A. Introducing the knowledge graph: things, not strings. Official google blog. 2012

38. Suchanek FM, Kasneci G, Weikum G. Yago: A large ontology from wikipedia and wordnet. Web Semantics: Science, Services and Agents on the World Wide Web. 2008; 6(3):203–217.

39. Tengli A, Yang Y, Ma NL. Learning table extraction from examples. COLING. 2004:987.

40. Turian J, Ratinov L, Bengio Y. Word representations: a simple and general method for semi-supervised learning. ACL. 2010:384–394.

41. Verga P, Belanger D, Strubell E, Roth B, McCallum A. Multilingual relation extraction using compositional universal schema. HLT-NAACL. 2016:886–896.

42. Welter D, MacArthur J, Morales J, Burdett T, Hall P, Junkins H, Klemm A, Flicek P, Manolio T, Hindorff L, et al. The nhgri gwas catalog, a curated resource of snp-trait associations. Nucleic acids research. 2014; 42:D1001–D1006. [PubMed: 24316577]

43. Wu Y, Schuster M, Chen Z, Le QV, Norouzi M, Macherey W, Krikun M, Cao Y, Gao Q, Macherey K, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144. 2016

44. Yahya M, Whang SE, Gupta R, Halevy A. ReNoun : Fact Extraction for Nominal Attributes. EMNLP. 2014:325–335.

45. Yang Y, Zhang H. Html page analysis based on visual cues. ICDAR. 2001:859–864.

46. Zhang Y, Chaganty A, Paranjape A, Chen D, Bolton J, Qi P, Manning CD. Stanford at tac kbp 2016: Sealing pipeline leaks and understanding chinese. TAC. 2016

A DATA PROGRAMMING

Machine-learning-based KBC systems rely heavily on ground truth data (called *training data*) to achieve high quality. Traditionally, manual annotations or incomplete KBs are used to construct training data for machine-learning-based KBC systems. However, these resources are either costly to obtain or may have limited coverage over the candidates considered during the KBC process. To address this challenge, Fonduer builds upon the newly introduced paradigm of data programming [33], which enables both domain experts and non-domain experts alike to programmatically generate large training datasets by leveraging multiple weak supervision sources and domain knowledge.

In data programming, which provides a framework for weak supervision, users provide weak supervision in the form of user-defined functions, called *labeling functions*. Each labeling function provides potentially noisy labels for a subset of the input data and are combined to create large, potentially overlapping sets of labels which can be used to train a machine-learning model. Many different weak-supervision approaches can be expressed as labeling functions. This includes strategies that use existing knowledge bases, individual annotators' labels (as in crowdsourcing), or user-defined functions that rely on domain-specific patterns and dictionaries to assign labels to the input data.

The aforementioned sources of supervision can have varying degrees of accuracy, and may conflict with each other. Data programming relies on a generative probabilistic model to estimate the accuracy of each labeling function by reasoning about the conflicts and overlap across labeling functions. The estimated labeling function accuracies are in turn used to assign a probabilistic label to each candidate. These labels are used in conjunction with a noise-aware discriminative model to train a machine-learning model for KBC.

A.1 Components of Data Programming

The main components in data programming are as follows:

Candidates

A set of candidates C to be probabilistically classified.

Labeling Functions

Labeling functions are used to programmatically provide labels for training data. A labeling function is a user-defined procedure that takes a candidate as input and outputs a label. Labels can be as simple as true or false for binary tasks, or one of many classes for more complex multiclass tasks. Since each labeling function is applied to all candidates and labeling functions are rarely perfectly accurate, there may be disagreements between them. The labeling functions provided by the user for binary classification can be more formally defined as follows. For each labeling function λ_i and $r \in C$, we have $\lambda_i : r \mapsto \{-1, 0, 1\}$ where $+1$ or -1 denotes a candidate as "True" or "False" and 0 abstains. The output of applying a set of l labeling functions to k candidates is the label matrix $\Lambda \in \{-1, 0, 1\}^{k \times l}$.

Output

Data-programming frameworks output a confidence value p for the classification for each candidate as a vector $Y \in \{p\}^k$.

To perform data programming in `Fonduer`, we rely on a data-programming engine, Snorkel [32]. Snorkel accepts candidates and labels as input and produces marginal probabilities for each candidate as output. These input and output components are stored as relational tables. Their schemas are detailed in Section 3.

A.2 Theoretical Guarantees

While data programming uses labeling functions to generate noisy training data, it theoretically achieves a learning rate similar to methods that use manually labeled data [33]. In the typical supervised-learning setup, users are required to manually label $\tilde{O}(\varepsilon^{-2})$ examples for the target model to achieve an expected loss of ε. To achieve this rate, data programming only requires the user to specify a constant number of labeling functions that does not depend on ε. Let β be the minimum coverage across labeling functions (i.e., the probability that a labeling function provides a label for an input point) and γ be the minimum reliability of labeling functions, where $\gamma = 2 \cdot a - 1$ with a denoting the accuracy of a labeling function. Then under the assumptions that (1) labeling functions are conditionally independent given the true labels of input data, (2) the number of user-

provided labeling functions is at least $\tilde{O}(\gamma^{-3}\beta^{-1})$, and (3) there are $k = \tilde{O}(\varepsilon^{-2})$ candidates, data programming achieves an expected loss ε. Despite the strict assumptions with respect to labeling functions, we find that using data programming to develop KBC systems for richly formatted data leads to high-quality KBs (across diverse real-world applications) even when some of the data-programming assumptions are not met (see Section 5.2).

B EXTENDED FEATURE LIBRARY

`Fonduer` augments a bidirectional LSTM with features from an extended feature library in order to better model the multiple modalities of richly formatted data. In addition, these extended features can provide signals drawn from large contexts since they can be calculated using `Fonduer`'s data model of the document rather than being limited to a single sentence or table. In Section 5, we find that including multimodal features is critical to achieving high-quality relation extraction. The provided extended feature library serves as a baseline example of these types of features that can be easily enhanced in the future. However, even with these baseline features, our users have been able to build high-quality knowledge bases for their applications.

The extended feature library consists of a baseline set of features from the structural, tabular, and visual modalities. Table 7 lists the details of the extended feature library. Features are represented as strings, and each feature space is then mapped into a one-dimensional bit vector for each candidate, where each bit represents whether the candidate has the corresponding feature.

C Fonduer AT SCALE

We use two optimizations to enable `Fonduer`'s scalability to millions of candidates (see Section 3.1): (1) data caching and (2) data representations that optimize data access during the KBC process. Such optimizations are standard in database systems. Nonetheless, their impact on KBC has not been studied in detail.

Each candidate to be classified by `Fonduer`'s LSTM as "True" or "False" is associated with a set of mentions (see Section 3.2). For each candidate, `Fonduer`'s multimodal featurization generates features that describe each individual mention in isolation and features that jointly describe the set of all mentions in the candidate. Since each mention can be associated with many different candidates, we cache the featurization of each mention. Caching during featurization results in a 100× speed-up on average in the Electronics domain yet only accounts for 10% of this stage's memory usage.

Recall from Section 3.3 that `Fonduer`'s programming model introduces two modes of operation: (1) development, where users iteratively improve the quality of labeling functions without executing the entire pipeline; and (2) production, where the full pipeline is executed once to produce the knowledge base. We use different data representations to implement the abstract data structures of Features and Labels (a structure that stores the output of labeling functions after applying them over the generated candidates). Implementing Features as a

list-of-lists structure minimizes runtime in both modes of operation since it accounts for sparsity. We also find that Labels implemented as a coordinate list during the development mode are optimal for fast updates. A list-of-lists implementation is used for Labels in production mode.

C.1 Data Caching

With richly formatted data, which frequently requires document-level context, thousands of candidates need to be featurized for each document. Candidate features from the extended feature library are computed at both the mention level and relation level by traversing the data model accessing modality attributes. Because each mention is part of many candidates, naïve featurization of candidates can result in the redundant computation of thousands of mention features. This pattern highlights the value of data caching when performing multimodal featurization on richly formatted data.

Traditional KBC systems that operate on single sentences of unstructured text pragmatically assume that only a small number of candidates will need to be featurized for each sentence and do not cache mention features as a result.

Example C.1 (Inefficient Featurization)

In Figure 1, the transistor part mention MMBT3904 could be matched with up to 15 different numerical values in the datasheet. Without caching, the features of the MMBT3904 would be unnecessarily recalculated 14 times, once for each candidate. In real documents 100s of feature calculations would be wasted.

In Example C.1, eliminating unnecessary feature computations can improve performance by an order of magnitude.

To optimize the feature-generation process, Fonduer implements a document-level caching scheme for mention features. The first computation of a mention feature requires traversing the data model. Then, the result is cached for fast access if the feature is needed again. All features are cached until all candidates in a document are fully featurized, after which the cache is flushed. Because Fonduer operates on documents atomically, caching a single document at a time improves performance without adding significant memory overhead. In the ELECTRONICS application, we find that caching achieves over 100× speed-up on average and in some cases even over 1000×, while only accounting for approximately 10% of the memory footprint of the featurization stage.

Takeaways—When performing feature generation from richly formatted data, caching the intermediate results can yield over 1000× improvements in featurization runtime without adding significant memory overhead.

C.2 Data Representations

The Fonduer programming model involves two modes of operation: (1) development and (2) production. In development, users iteratively improve the quality of their labeling functions through error analysis and without executing the full pipeline as in previous

techniques such as incremental KBC [36]. Once the labeling functions are finalized, the `Fonduer` pipeline is only run once in production.

In both modes of operation, `Fonduer` produces two abstract data structures (Features and Labels as described in Section 3). These data structures have three access patterns: (1) *materialization*, where the data structure is created; (2) *updates*, which include inserts, deletions, and value changes; and (3) *queries*, where users can inspect the features and labels to make informed updates to labeling functions.

Both Features and Labels can be viewed as matrices, where each row represents annotations for a candidate (see Section 3.2). Features are dynamically named during multimodal featurization but are static for the lifetime of a candidate. Labels are statically named in classification but updated during development. Typically Features are sparse: in the ELECTRONICS application, each candidate has about 100 features while the number of unique features can be more than 10M. Labels are also sparse, where the number of unique labels is the number of labeling functions.

The data representation that is implemented to store these abstract data structures can significantly affect overall system runtime. In the ELECTRONICS application, multimodal featurization accounts for 50% of end-to-end runtime, while classification accounts for 15%. We discuss two common sparse matrix representations that can be materialized in a SQL database.

- **List of lists** (LIL): Each row stores a list of (column key, value) pairs. Zero-valued pairs are omitted. An entire row can be retrieved in a single query. However, updating values requires iterating over sublists.

- **Coordinate list** (COO): Rows store (row key, column - key, value) triples. Zero-valued triples are omitted. With COO, multiple queries must be performed to fetch a row's attributes. However, updating values takes constant time.

The choice of data representation for Features and Labels reflects their different access patterns, as well as the mode of operation. During development, Features are materialized once, but frequently queried during the iterative KBC process. Labels are updated each time a user modifies labeling functions. In production, Features' access pattern remains the same. However, Labels are not updated once users have finalized their set of labeling functions.

From the access patterns in the `Fonduer` pipeline, and the characteristics of each sparse matrix representation, we find that implementing Features as an LIL minimizes runtime in production and development. Labels, however, should be implemented as COO to support fast insertions during iterative KBC and reduce runtimes for each iteration. In production, Labels can also be implemented as LIL to avoid the computation overhead of COO. In the ELECTRONICS application, we find that LIL provides $1.4\times$ speed-up over COO in production and that COO provides over $5.8\times$ speed-up over LIL when adding a new labeling function.

Takeaways

We find that Labels should be implemented as a coordinate list during development, which supports fast updates for supervision, while Features should use a list of lists, which

provides faster query times. In production, both Features and Labels should use a list-of-list representation.

D FUTURE WORK

Being able to extract information from richly formatted data enables a wide range of applications, and represents a new and interesting research direction. While we have demonstrated that Fonduer can already obtain high-quality knowledge bases in several applications, we recognize that many interesting challenges remain. We briefly discuss some of these challenges.

Data Model

One challenge in extracting information from richly formatted data comes directly at the data level—we cannot perfectly preserve all document information. Future work in parsing, OCR, and computer vision have the potential to improve the quality of Fonduer's data model for complex table structures and figures. For example, improving the granularity of Fonduer's data model to be able to identify axis titles, legends, and footnotes could provide additional signals to learn from and additional specificity for users to leverage while using the Fonduer programming model.

Deep-Learning Model

Fonduer's multimodal recurrent neural network provides a prototypical automated featurization approach that achieves high quality across several domains. However, future developments for incorporating domain-specific features could strengthen these models. In addition, it may be possible to expand our deep-learning model to perform additional tasks (e.g., identifying candidates) to simplify the Fonduer pipeline.

Programming Model

Fonduer currently exposes a Python interface to allow users to provide weak supervision. However, further research in user interfaces for weak supervision could bolster user efficiency in Fonduer. For example, allowing users to use natural language or graphical interfaces in supervision may result in improved efficiency and reduced development time through a more powerful programming model. Similarly, feedback techniques like *active learning* [35] could empower users to more quickly recognize classes of candidates that need further disambiguation with LFs.
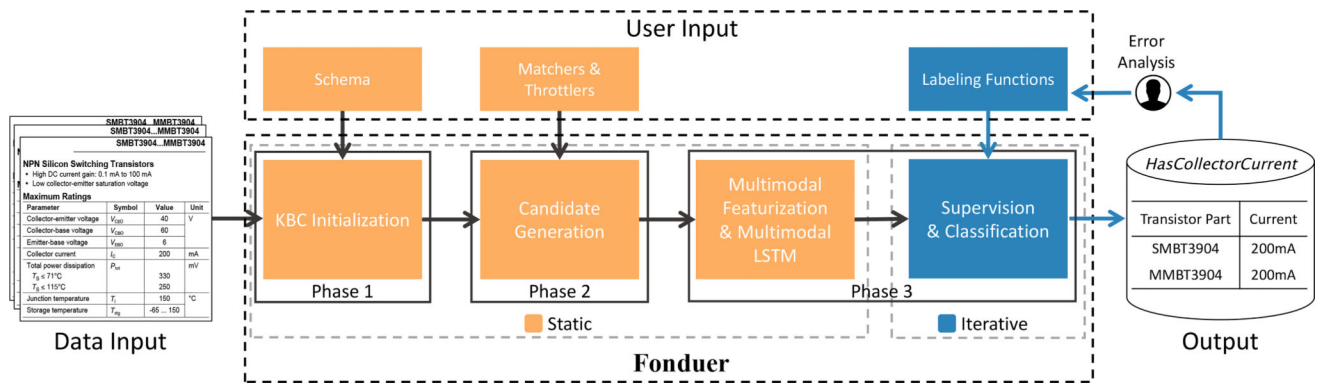
Figure 1.

A KBC task to populate relation HasCollectorCurrent (Transistor Part, Current) from transistor datasheets. Part and Current mentions are in blue and green, respectively.

Figure 2.
An overview of Fonduer KBC over richly formatted data. Given a set of richly formatted documents and a series of lightweight inputs from the user, Fonduer extracts facts and stores them in a relational database.
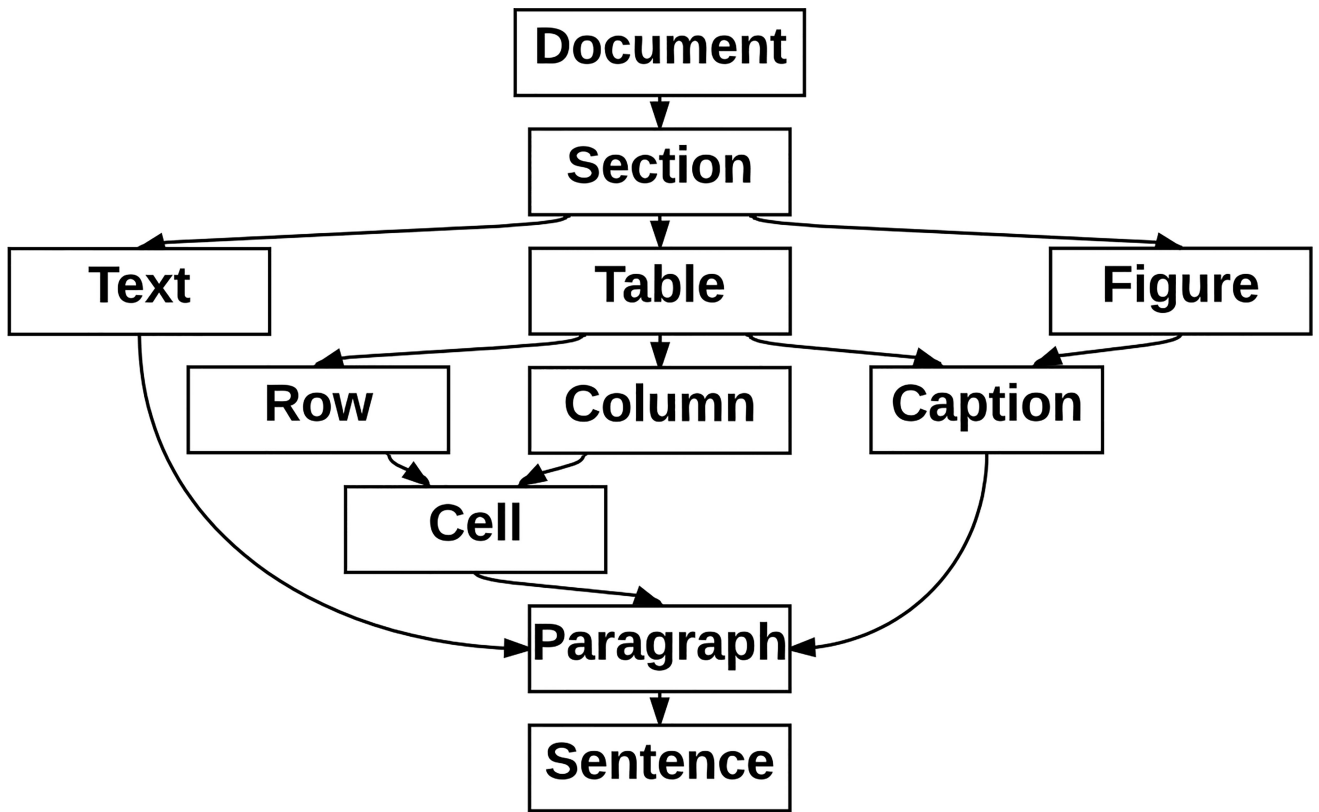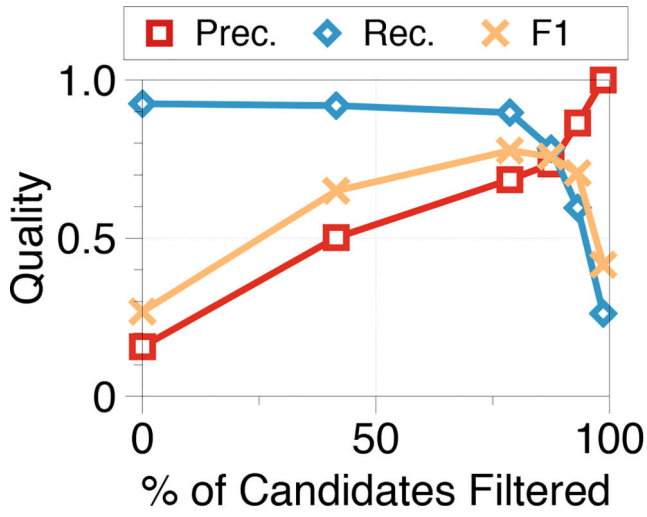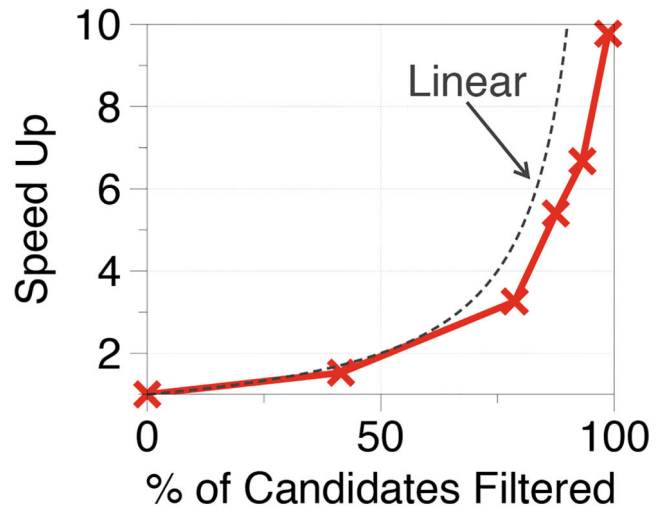
Figure 3.
Fonduer's data model.

Figure 4.
Tradeoff between (a) quality and (b) execution time when pruning the number of candidates using throttlers.
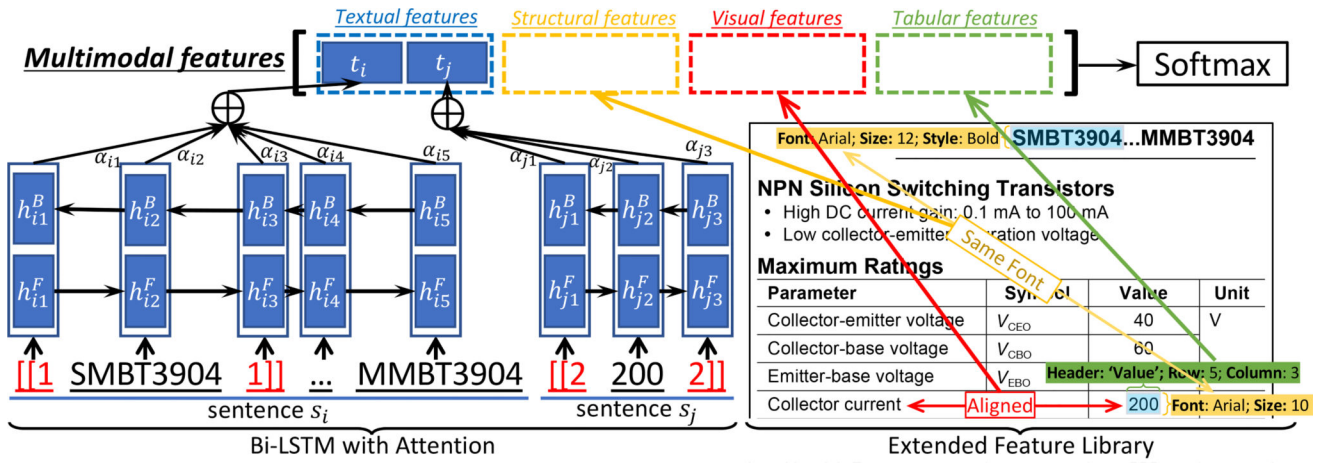
Figure 5.
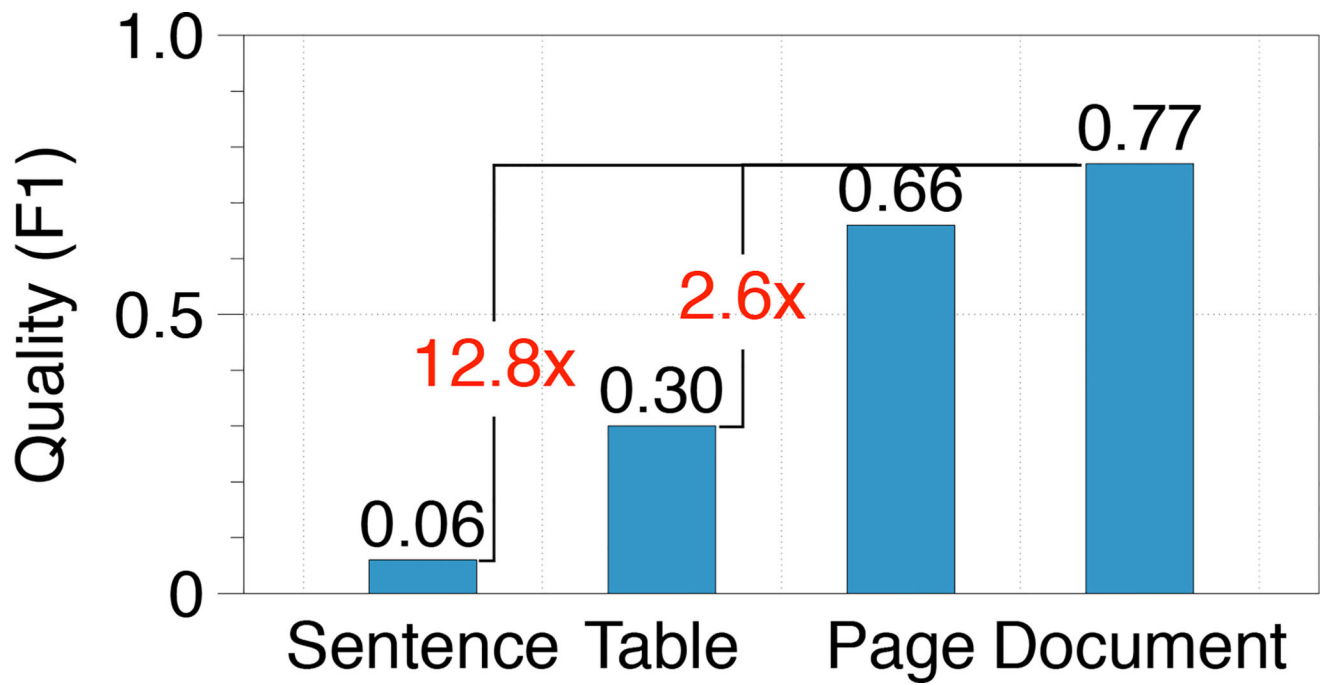An illustration of `Fonduer`'s multimodal LSTM for candidate (SMBT3904, 200) in Figure 1.
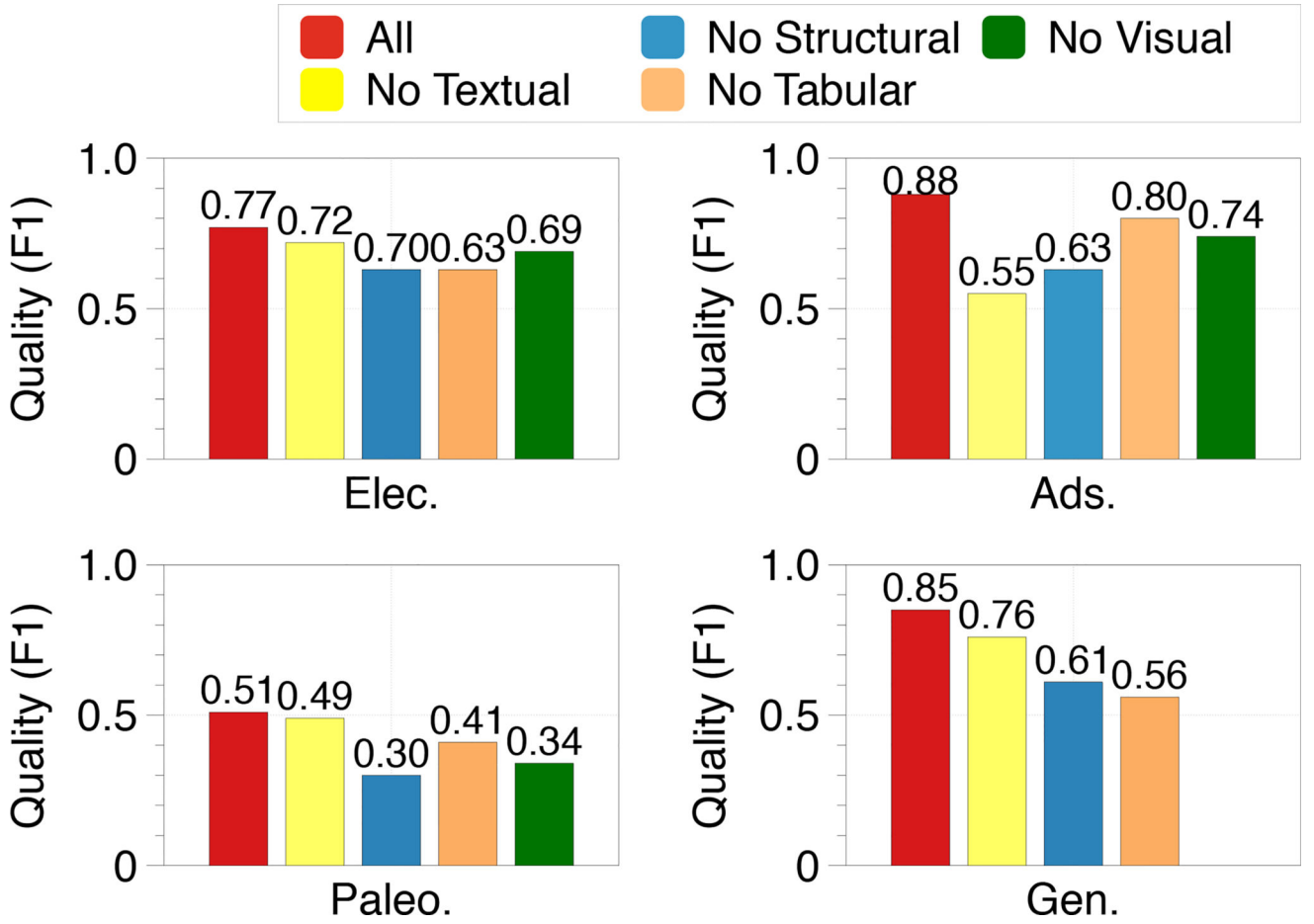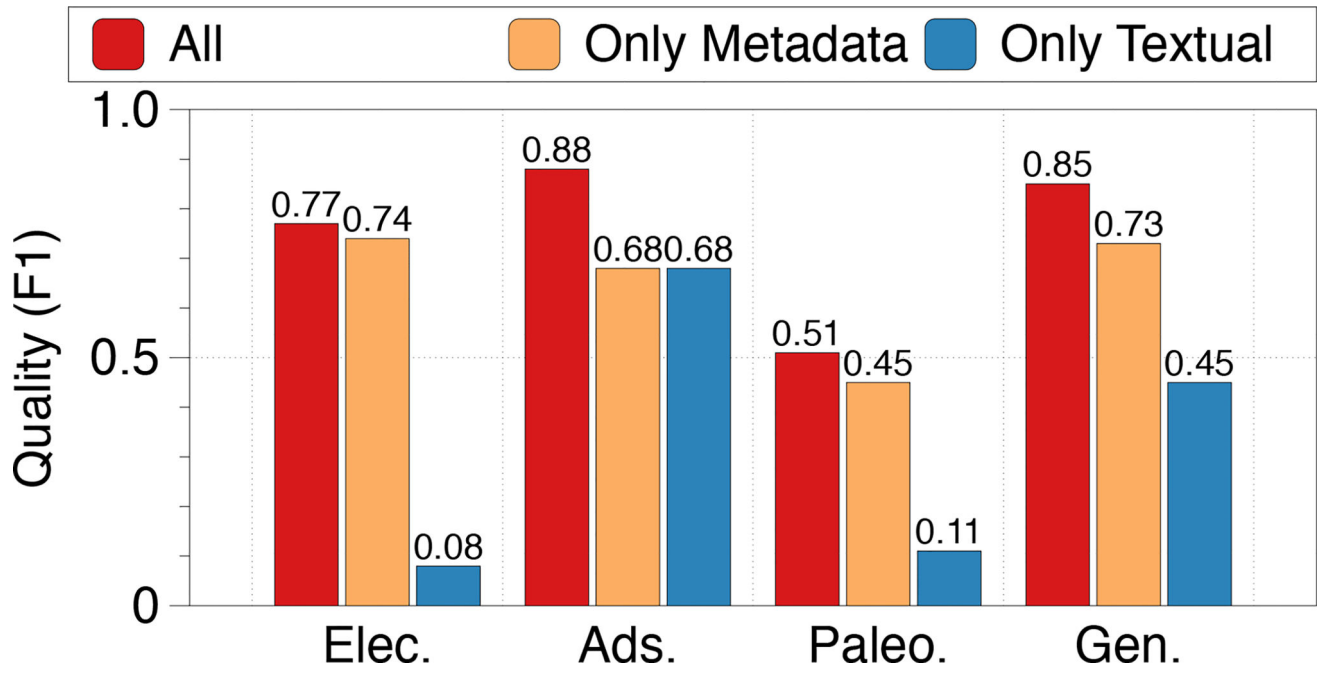
Figure 6.
Average F1 score over four relations when broadening the extraction context scope in
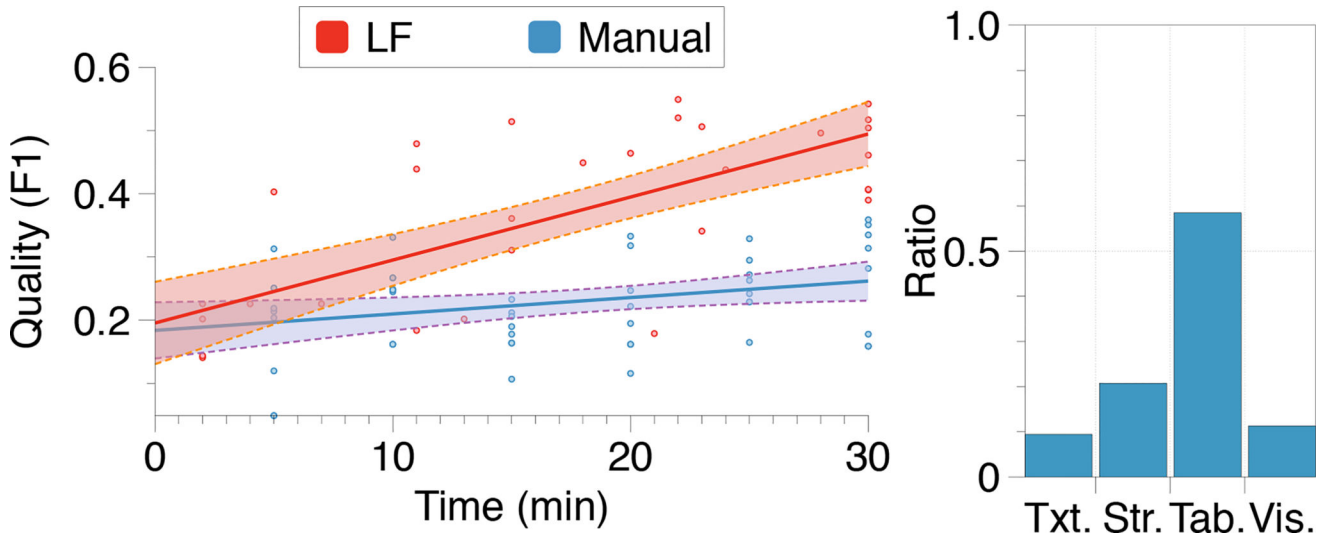ELECTRONICS.

Figure 7.
The impact of each modality in the feature library.

Figure 8.
Study of different supervision resources on quality. Metadata includes structural, tabular, and visual information.

Figure 9.
F1 quality over time with 95% confidence intervals (left). Modality distribution of user labeling functions (right).

Table 1

Summary of the datasets used in our experiments.

Dataset	Size	#Docs	#Rels	Format
Elec.	3GB	7K	4	PDF
Ads.	52GB	9.3M	4	HTML
Paleo.	95GB	0.3M	10	PDF
Gen.	1.8GB	589	4	XML

Table 2

End-to-end quality in terms of precision, recall, and F1 score for each application compared to the upper bound of state-of-the-art systems.

Sys.	Metric	Text	Table	Ensemble	Fonduer
Elec.	Prec.	1.00	1.00	1.00	0.73
	Rec.	0.03	0.20	0.21	0.81
	F1	0.06	0.40	0.42	**0.77**
Ads.	Prec.	1.00	1.00	1.00	0.87
	Rec.	0.44	0.37	0.76	0.89
	F1	0.61	0.54	0.86	**0.88**
Paleo.	Prec.	0.00	1.00	1.00	0.72
	Rec.	0.00	0.04	0.04	0.38
	F1	0.00[*]	0.08	0.08	**0.51**
Gen.	Prec.	0.00	0.00	0.00	0.89
	Rec.	0.00	0.00	0.00	0.81
	F1	0.00[#]	0.00[#]	0.00[#]	**0.85**

[*] *Text* did not find any candidates.

[#] No full tuples could be created using *Text* or *Table* alone

Table 3

End-to-end quality vs. existing knowledge bases.

System	Elec.	Gen.	
Knowledge Base	**Digi-Key**	**GWAS Central**	**GWAS Catalog**
# Entries in KB	376	3,008	4,023
# Entries in Fonduer	447	6,420	6,420
Coverage	0.99	0.82	0.80
Accuracy	0.87	0.87	0.89
# New Correct Entries	17	3,154	2,486
Increase in Correct Entries	1.05×	1.87×	1.42×

Table 4

Comparing approaches to featurization based on Fonduer's data model.

Sys.	Metric	Human-tuned	Bi-LSTM w/ Attn.	Fonduer
Elec.	Prec.	0.71	0.42	0.73
	Rec.	0.82	0.50	0.81
	F1	0.76	0.45	**0.77**
Ads.	Prec.	0.88	0.51	0.87
	Rec.	0.88	0.43	0.89
	F1	**0.88**	0.47	**0.88**
Paleo.	Prec.	0.92	0.52	0.76
	Rec.	0.37	0.15	0.38
	F1	**0.53**	0.23	0.51
Gen.	Prec.	0.92	0.66	0.89
	Rec.	0.82	0.41	0.81
	F1	**0.87**	0.47	0.85

Table 5

Comparing the features of SRV and Fonduer.

Feature Model	Precision	Recall	F1
SRV	0.72	0.34	0.39
Fonduer	0.87	0.89	0.88

Table 6

Comparing document-level RNN and Fonduer's deep-learning model on a single relation from Electronics.

Learning Model	Runtime during Training (secs/epoch)	Quality (F1)
Document-level RNN	37,421	0.26
Fonduer	48	0.65

Table 7

Features from Fonduer's feature library. Example values are drawn from the example candidate in Figure 1. Capitalized prefixes represent the feature templates and the remainder of the string represents a feature's value.

Feature Type	Arity	Description	Example Value
Structural	Unary	HTML tag of the mention	`TAG_<h1>`
Structural	Unary	HTML attributes of the mention	`HTML_ATTR_font-family:Arial`
Structural	Unary	HTML tag of the mention's parent	`PARENT_TAG_<p>`
Structural	Unary	HTML tag of the mention's previous sibling	`PREV_SIB_TAG_<td>`
Structural	Unary	HTML tag of the mention's next sibling	`NEXT_SIB_TAG_<h1>`
Structural	Unary	Position of a node among its siblings	`NODE_POS_1`
Structural	Unary	HTML class sequence of the mention's ancestors	`ANCESTOR_CLASS_<s1>`
Structural	Unary	HTML tag sequence of the mention's ancestors	`ANCESTOR_TAG_<body>_<p>`
Structural	Unary	HTML ID's of the mention's ancestors	`ANCESTOR_ID_l1`[b]
Structural	Binary	HTML tags shared between mentions on the path to the root of the document	`COMMON_ANCESTOR_<body>`
Structural	Binary	Minimum distance between two mentions to their lowest common ancestor	`LOWEST_ANCESTOR_DEPTH_1`
Tabular	Unary	N-grams in the same cell as the mention[a]	`CELL_cev`[b]
Tabular	Unary	Row number of the mention	`ROW_NUM_5`
Tabular	Unary	Column number of the mention	`COL_NUM_3`
Tabular	Unary	Number of rows the mention spans	`ROW_SPAN_1`
Tabular	Unary	Number of columns the mention spans	`COL_SPAN_1`
Tabular	Unary	Row header n-grams in the table of the mention	`ROW_HEAD_collector`
Tabular	Unary	Column header n-grams in the table of the mention	`COL_HEAD_value`
Tabular	Unary	N-grams from all Cells that are in the same row as the given mention[a]	`ROW_200_[ma]`[c]
Tabular	Unary	N-grams from all Cells that are in the same column as the given mention[a]	`COL_200_[6]`[c]
Tabular	Binary	Whether two mentions are in the same table	`SAME_TABLE`[b]
Tabular	Binary	Row number difference if two mentions are in the same table	`SAME_TABLE_ROW_DIFF_1`[b]
Tabular	Binary	Column number difference if two mentions are in the same table	`SAME_TABLE_COL_DIFF_3`[b]
Tabular	Binary	Manhattan distance between two mentions in the same table	`SAME_TABLE_MANHATTAN_DIST_10`[b]
Tabular	Binary	Whether two mentions are in the same cell	`SAME_CELL`[b]
Tabular	Binary	Word distance between mentions in the same cell	`WORD_DIFF_1`[b]
Tabular	Binary	Character distance between mentions in the same cell	`CHAR_DIFF_1`[b]
Tabular	Binary	Whether two mentions in a cell are in the same sentence	`SAME_PHRASE`[b]
Tabular	Binary	Whether two mention are in the different tables	`DIFF_TABLE`[b]

Feature Type	Arity	Description	Example Value
Tabular	Binary	Row number difference if two mentions are in different tables	`DIFF_TABLE_ROW_DIFF_4`[b]
Tabular	Binary	Column number difference if two mentions are in different tables	`DIFF_TABLE_COL_DIFF_2`[b]
Tabular	Binary	Manhattan distance between two mentions in different tables	`DIFF_TABLE_MANHATTAN_DIST_7`[b]
Visual	Unary	N-grams of all lemmas visually aligned with the mention[a]	`ALIGNED_current`
Visual	Unary	Page number of the mention	`PAGE_1`
Visual	Binary	Whether two mentions are on the same page	`SAME_PAGE`
Visual	Binary	Whether two mentions are horizontally aligned	`HORZ_ALIGNED`[b]
Visual	Binary	Whether two mentions are vertically aligned	`VERT_ALIGNED`
Visual	Binary	Whether two mentions' left bounding-box borders are vertically aligned	`VERT_ALIGNED_LEFT`[b]
Visual	Binary	Whether two mentions' right bounding-box borders are vertically aligned	`VERT_ALIGNED_RIGHT`[b]
Visual	Binary	Whether the centers of two mentions' bounding boxes are vertically aligned	`VERT_ALIGNED_CENTER`[b]

[a] All N-grams are 1-grams by default.

[b] This feature was not present in the example candidate. The values shown are example values from other documents.

[c] In this example, the mention is 200, which forms part of the feature prefix. The value is shown in square brackets.