Mindtagger: A Demonstration of Data Labeling in Knowledge Base Construction

Jaeho Shin
Stanford University
jaeho@cs.stanford.edu

Christopher Ré
Stanford University
chrismre@cs.stanford.edu

Michael Cafarella
University of Michigan
michjc@umich.edu

ABSTRACT

End-to-end knowledge base construction systems using statistical inference are enabling more people to automatically extract high-quality domain-specific information from unstructured data. As a result of deploying DeepDive framework across several domains, we found new challenges in debugging and improving such end-to-end systems to construct high-quality knowledge bases. DeepDive has an iterative development cycle in which users improve the data. To help our users, we needed to develop principles for analyzing the system's error as well as provide tooling for inspecting and labeling various data products of the system. We created guidelines for error analysis modeled after our colleagues' best practices, in which data labeling plays a critical role in every step of the analysis. To enable more productive and systematic data labeling, we created Mindtagger, a versatile tool that can be configured to support a wide range of tasks. In this demonstration, we show in detail what data labeling tasks are modeled in our error analysis guidelines and how each of them is performed using Mindtagger.

1. INTRODUCTION

End-to-end knowledge base construction (KBC) systems using statistical inference are enabling more people to automatically extract high-quality domain-specific information from unstructured data. One motivating example for our work is the MEMEX [1] project in which law enforcement officials fight human trafficking crimes by extracting information from sex and work labor ads on the internet. The system extracts information such as phone numbers, wages, and identifiers of trafficked victims from ads crawled from the dark web, which is not readily accessible from ordinary search engines. A wide range of data processing components are put together in the system: crawlers pull in large amounts of unstructured text and image data from various sources; ETL (extract, transform, and load) components clean the raw data and add natural language processing markups to the text; an array of extractors extract candidate mentions of entities and relationships along with their features; an inference engine constructs and trains a probabilistic model from the data to predict probabilities of the extracted information; finally, a search engine surfaces such information to law enforcement personnel. Such an end-to-end system achieves higher quality as its interdependent components are not developed in isolation but are improved as a whole. Knowing where to look to improve the quality is difficult, and this is where the tool we demonstrate, Mindtagger helps.

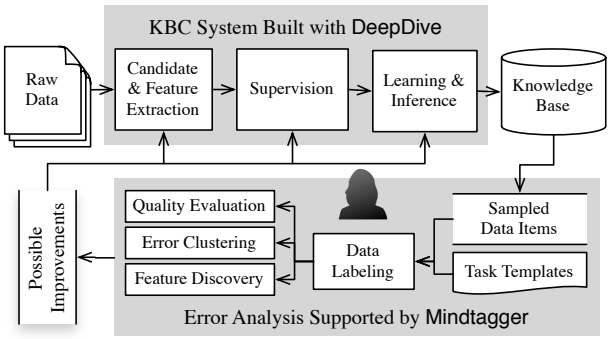


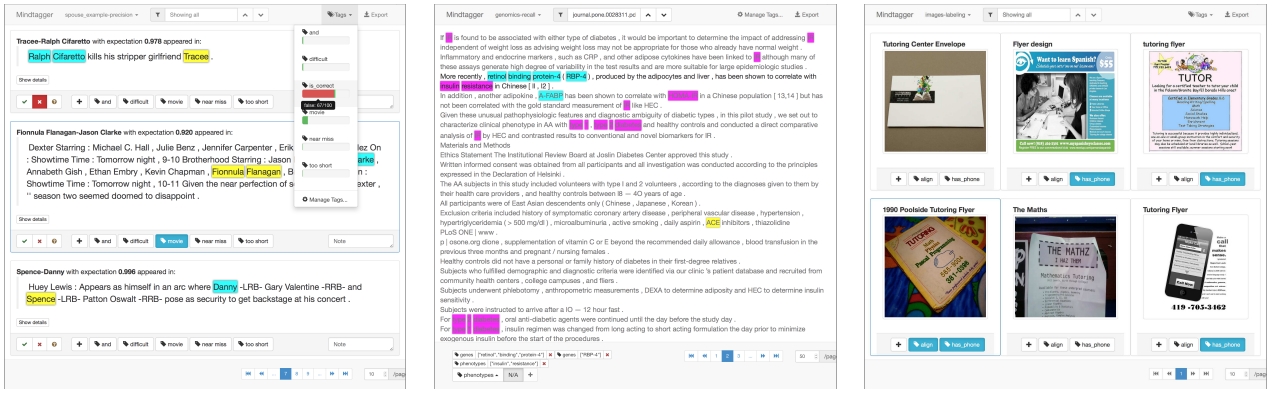
Figure 1: Iterative development cycle of a knowledge base construction system built with DeepDive and error analysis steps supported by Mindtagger.

Mindtagger is a tool built on top of DeepDive [4],[1] a framework we created that allows users to process unstructured data to extract structured knowledge using statistical inference. Several groups in different domains are using DeepDive and Mindtagger to construct high-quality knowledge bases: paleo-biology, genomics, pharma domains, intelligence, law-enforcement, and material science. We identified the following challenges in developing KBC systems:

- **Principled Error Analysis.** DeepDive users make incremental improvements over many short iterations to the basic versions of its user-defined components. Users evaluate the end result after each iteration because every component can have an impact on the overall quality of the knowledge base being constructed. However, in many cases, our colleagues were tempted to examine only a small sample of errors and to use their intuition and luck to fix whichever attractive ones they encountered. Modeling after the best practices from our successful collaboration, we created guidelines for *error analysis* [3] that help users

[1]Both are open source at `http://git.io/mindtagger` and `http://deepdive.stanford.edu`.

Proceedings of the VLDB Endowment, Vol. 8, No. 12



(a) Precision mode (b) Full markup recall mode (c) Inspection mode for labeling images

Figure 2: Screenshots of Mindtagger operating in three different modes.

identify possible improvements and assess their potential impact in a principled way.

- **Productive Data Labeling.** Lack of tooling for inspecting and labeling data products of the system under development was slowing down every iteration of the development cycle. In every step of our error analysis guidelines, *data labeling* plays a key role. For example, identifying errors from sampled data products and inspecting the errors in more depth to collect ideas for improvements both require labeling the data. We noticed that even when our colleagues followed good principle, precious development time was being wasted in mundane data transformation and cleaning tasks, such as ad-hoc reformatting of data products to more human-friendly representations and manual collection of the labels. To enable more productive data labeling and to study unanticipated types of labeling tasks involved in the actual error analysis, we created Mindtagger, an interactive graphical tool for labeling data. Innovative systems such as Data Tamer [5] and Trifacta [2] are making human involvement more productive in data integration, cleaning, and transformation, but our setup is richer in the sense that end-to-end KBC systems typically handle those problems as part of the statistical inference.
- **Variable Data Products and Labeling Tasks.** Although conceptually similar error analysis steps are performed every time, the data to be inspected as well as the detail of necessary labeling tasks constantly vary as development progresses. For example, deciding the correctness of certain relationship extraction may require presentation of extra information very specific to it, and newly extracted features may need special visualizations for proper examination. We designed Mindtagger to support a wide range of tasks and data types by mixing and matching predefined configuration fragments so that the task configurations could be easily reused and extended.

In this demonstration, we present in detail the data labeling tasks modeled in our error analysis guidelines, and how our versatile labeling tool can be used in a principled way to perform each of them, using a simplified part of a hypothetical knowledge base construction system as an example.

2. SYSTEM OVERVIEW

Mindtagger is an interactive data labeling tool we created to help DeepDive users perform error analysis in more systematic and productive ways. Mindtagger takes as input a list of data items and a template that defines the presentation of, and possible interactions with the items and provides a graphical user interface for annotating labels while browsing them. It displays simple statistics of the entered annotations and also outputs them in various formats (e.g., CSV, SQL, and JSON). Users produce more sophisticated statistics from the annotations or take them back into the DeepDive data flow as training sets. Figure 1 shows where Mindtagger and the error analysis steps it supports fit into the overall iterative development cycle of an end-to-end knowledge base construction system. For error analysis, DeepDive users run a set of SQL queries against their underlying database to obtain samples of data products resulting from the last development iteration. Each of those queries have associated Mindtagger templates set up for a particular data labeling task, and some have additional scripts for post-processing the collected annotations. Common configurations of the tool for error analysis are described in more detail in Section 3.

Mindtagger can be easily configured to support a wide range of data labeling tasks by customizing one of its predefined modes: precision, full markup recall, quick check recall, and inspection mode, some of which are shown in Figure 2. Let's take an example of estimating the precision of an extraction, which is about labeling false positive errors in the sample. Such a task can be set up by simply declaring how each item is presented in a user-defined template for precision mode. The tool then provides a GUI optimized for the task with buttons for labeling each item as correct or incorrect along with keyboard shortcuts for quicker navigation and annotation. In addition to the correct/incorrect labels, it provides a way to annotate each item with ad-hoc, user-defined tags. We found this functionality to be particularly powerful for error analysis because it gives a means to clustering the items as the annotator goes through them while entering the main labels. Various types of data items are supported by the tool's minimal HTML-based template syntax. For example, an array of words decorated with natural language processing markups, which is a data representation widely used across text-based DeepDive applications, can be easily presented as a human-readable sentence, and image data can be presented as nicely rendered graphics with minimal effort. Although the tool was primarily created for DeepDive, it can be used independently as a general data labeling tool for any data items materialized in either CSV/TSV or JSON format.

3. DEMONSTRATION DETAILS

We use a small part of a hypothetical knowledge base construction system to demonstrate how Mindtagger supports the data labeling tasks necessary for a full error analysis cycle. Our guidelines for error analysis are modeled after the best practices used by successful DeepDive users, and it boils down to following these three steps after every iteration:

1. Evaluate the precision and/or recall of the extractions by identifying errors from a sample of the data product.
2. Inspect and cluster the errors with their immediate details, such as extracted features.
3. Explore extra data products relevant to the most common errors to develop concrete ideas for improvement.

For each step, we also modeled the necessary data labeling tasks and designed Mindtagger to support them with varying configurations of data items, their presentation, the labels to collect, and allowed interactions during the tasks.

Phone Number Extraction Example. As a classical example, a demo attendee will suppose she is extracting phone numbers mentioned in a collection of ads on the web as part of a greater knowledge base constructed, such as the anti-human-trafficking one built for MEMEX. In real KBC systems, highly domain-specific concepts such as gene, protein, drug, disease, phenotype, and material mentions are extracted, but we use this example as it is familiar to everyone. For simplicity, we assume the information extraction is done using a simple binary classifier, and the following four components are in her system:

- **Candidate Extraction.** A basic candidate extractor scans the full text of the web pages with a set of simple regular expressions and emits matching text-spans that are likely to be phone numbers.
- **Feature Extraction.** Then a feature extractor collects words and phrases appearing near each candidate as its features for classification.
- **Supervision.** Instead of manually curating a training set, a program with a set of rules collects candidates as positive training examples that appear in a sentence that contains words highly likely to appear next to phone numbers, such as "call," "phone," or "contact."
- **Inference.** Finally, the binary classifier uses logistic regression to learn and predict whether the text-span is a phone number or not.

3.1a. Evaluating Precision. To get a sense of how well each component is performing at the moment, the attendee first evaluates precision by interacting with Mindtagger. Precision of the candidate extractor as well as the classifier is evaluated by first taking a sample of the output, then labeling the false positives. The estimated precision is simply the ratio of the number of true positives to the total number of samples examined. We present Mindtagger's *precision mode* with the following configuration so the attendee can try performing the labeling task as shown in Figure 2 (a).

- **Data items.** Random samples of text-spans extracted as candidates for phone numbers (or classified as phone numbers) are prepared.
- **Presentation.** Each sentence containing a candidate text-span is shown with the candidate highlighted.
- **Labels/Interaction.** Whether a candidate is a valid phone number can be annotated by the attendee. Buttons and keyboard shortcuts are used to quickly mark and move to the next one. Counts of each label are displayed, and items can be quickly filtered by their labels.

3.1b. Evaluating Recall. Mindtagger provides two modes for recall estimation: full markup and quick check modes. Estimating recall is slightly more challenging than precision, as it's about the unseen part of the data. The foremost objective of evaluating recall is to discover false negatives from the samples drawn from the entire corpus, not from the output of the extractor or classifier. The estimated recall is then the reciprocal of 1 + the relative size of false negatives to true positives in the sample.

1. *Full markup mode* is designed for carefully identifying all mentions appearing in sampled documents, as shown in Figure 2 (b). The exact number of false negatives can be counted by comparing how many were missed by the extractor or classifier among all mentions highlighted through this mode. The attendee can interact with the Mindtagger task set up as below to understand how sampled text could be fully marked up.
   - **Data items.** All sentences of a few randomly sampled documents drawn from the entire corpus are prepared.
   - **Presentation.** Sentences are displayed in a readable format similar to their original appearance on the web.
   - **Labels/Interaction.** Multiple text-spans in each sentence can be highlighted as valid phone numbers by the attendee. The highlighted positions are collected, and we show an example script for comparing them against the candidate extractor and classifier's output.
2. *Quick check mode* is for tagging whether a sentence or paragraph simply contains a mention or not. The false negatives can be counted approximately from the number of sentences/paragraphs that were marked by this mode yet didn't contain any output of the candidate extractor or classifier. This mode is less burdensome than the full markup mode, so it can be performed much more quickly for a greater number of samples. Although the coarser granularity may contribute to inaccuracy, we found it's a cost-effective way for obtaining a reasonable estimate in many cases. The attendee can also try this mode in Mindtagger to compare with the full markup mode.
   - **Data items.** Random samples of sentences or paragraphs drawn from the entire corpus are prepared.
   - **Presentation.** Each sentence/paragraph is displayed separately in a readable format.
   - **Labels/Interaction.** Whether each item contains a valid phone number can be annotated by the attendee. Buttons and keyboard shortcuts similar to precision mode are used to quickly mark and move to the next one. We show an example script for using the entered labels to compute the approximate recall and compare with the estimate computed from full markup mode.

3.2. Inspecting Errors. We assume the attendee wants to improve the extraction quality after evaluating precision and recall, although error analysis in general can stop here when the measures are satisfactory. She now has to understand the cause for each error found in the previous steps in order to cluster them and focus on the most common type. Improvement of the candidate extractor can be guided by concrete

error examples of false negatives that were neglected and false positives that are easy to suppress from becoming a candidate. Having the extracted features visible along with their learned weights while inspecting each error helps analyze how certain false negatives got such low probability or how some false positives weren't suppressed by the classifier. Because it is impossible to know in advance which types of errors are the most widespread ones, and hence having the greatest impact when fixed, clustering them during the inspection is very important. The attendee browses the errors partially labeled with their causes in Mindtagger's *inspection mode* using the configuration below. She can add her own labels to the text mistaken as phone numbers, such as "part of URL" or "part of product code," which hints at how the regular expressions should be tuned or what new features should be extracted.

- **Data items.** False positive errors from the precision evaluation task are prepared. In general, the set of errors to inspect is chosen from corresponding labeling tasks depending on the target measures to improve (precision, recall, or both). Each error is joined with the list of extracted features and weights if available.
- **Presentation.** Each error is displayed as a highlighted text-span in its sentence along with the probability predicted by the classifier and extracted features with their learned weights as a table if available.
- **Labels/Interaction.** Labels denoting the analyzed cause of the error can be added by the attendee. Text input box, buttons, and keyboard shortcuts are used to quickly enter a new label or to select already entered ones. The size of each class of errors is displayed by cause, and the examples can be quickly accessed through filtering by label.

3.3. Calibrating Supervision. By now the attendee is supposed to observe that the most common errors are due to confusing part of a URL that contains the word "phone" as phone numbers. To correct the inspected errors, it is often necessary to explore more data that share features with the errors to calibrate the supervision rules. False positives can be explained by features having spuriously high weights learned from a biased training set with little negative examples but too many positive ones. Taking a broader look at more candidates sharing such features helps in understanding which rules were overly supervising the candidates as positive training examples, or more importantly, what new rules could be created to turn some of the unsupervised candidates into negative training examples. Similarly, false negatives due to low-weighted features can be handled by calibrating the supervision rules to introduce more positive examples after examining other candidates sharing the features. From previous steps, the attendee already senses that the rule treating all 10-digit numbers near the word "phone" as positive training examples is problematic. Using Mindtagger's inspection mode, the attendee can reliably come up with new rules for negative training examples based on concrete observations.

- **Data items.** Other candidates that also have the "phone" word feature are sampled. They are all positive training examples in this case, but in general, they are joined with training labels and provenance information for the responsible supervision rules. Other features extracted for each candidate along with their learned weights are also joined.
- **Presentation.** Each candidate is displayed as highlights within its sentence along with its supervised label and the relevant rules, extracted features, and learned weights.
- **Labels/Interaction.** Labels denoting ideas for fixing existing rules or creating new rules can be added to each candidate by the attendee. Text input box, buttons, and keyboard shortcuts are used to quickly annotate them.

3.4. Labeling Images. To demonstrate Mindtagger's capability of handling data types other than text, we show a task for labeling images. Suppose a large fraction of phone numbers actually appears in images embedded in web ads. Another flow of extraction with an image classifier must detect and classify parts of images as containing phone numbers. The attendee sees how data labeling tasks for text presented so far can be set up similarly for images with little modifications and tries interacting with Mindtagger's inspection mode set up for images as shown in Figure 2 (c).

- **Data items.** Candidates are now parts of images that appear in web ads instead of text-spans of sentences.
- **Presentation.** Each data item is displayed as a highlighted part in the rendered image.
- **Labels/Interaction.** Labels to collect and supported interactions are mostly invariant to the data type difference. In the case of full markup mode for images, the tool allows the attendee to select a region in the displayed image to collect the boundary coordinates.

4. CONCLUSION

We introduced the challenges of debugging and improving end-to-end knowledge base construction systems, learned from our collaborative experience using DeepDive. We created guidelines for error analysis of such system's data products, where data labeling must be performed at every step. Mindtagger is a tool we created for users to perform a wide range of labeling tasks more productively and systematically. Our demonstration shows how the tool supports the data labeling tasks modeled in our guidelines.

Acknowledgments We gratefully acknowledge the support of the Defense Advanced Research Projects Agency (DARPA) XDATA program under No. FA8750-12-2-0335 and DEFT program under No. FA8750-13-2-0039, MEMEX and SIMPLEX program, the National Science Foundation (NSF) CAREER Award under No. IIS-1353606, the Office of Naval Research (ONR) under awards No. N000141210041 and No. N000141310129, the National Institutes of Health Grant U54EB020405 awarded by the National Institute of Biomedical Imaging and Bioengineering (NIBIB) through funds provided by the trans-NIH Big Data to Knowledge (BD2K, http://www.bd2k.nih.gov) initiative, the Sloan Research Fellowship, the Moore Foundation, American Family Insurance, Google, and Toshiba. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, AFRL, NSF, ONR, NIH, or the U.S. government.

5. REFERENCES

[1] http://www.cbsnews.com/news/new-search-engine-exposes-the-dark-web/.
[2] http://www.trifacta.com/.
[3] C. Ré, A. A. Sadeghian, Z. Shan, J. Shin, F. Wang, S. Wu, and C. Zhang. Feature engineering for knowledge base construction. *IEEE Data Eng. Bull.*, 37(3):26–40, 2014.
[4] J. Shin, S. Wu, F. Wang, C. D. Sa, C. Zhang, and C. Ré. Incremental knowledge base construction using deepdive. *PVLDB*, 8(11), 2015.
[5] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The data tamer system. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. www.cidrdb.org, 2013.