Feature Engineering for Knowledge Base Construction

Christopher Ré[†] Amir Abbas Sadeghian[†] Zifei Shan[†]
Jaeho Shin[†] Feiran Wang[†] Sen Wu[†] Ce Zhang[†‡]
[†]Stanford University
[‡]University of Wisconsin-Madison
{chrismre, amirabs, zifei, jaeho.shin, feiran, senwu, czhang}@cs.stanford.edu

Abstract

Knowledge base construction (KBC) is the process of populating a knowledge base, i.e., a relational database together with inference rules, with information extracted from documents and structured sources. KBC blurs the distinction between two traditional database problems, information extraction and information integration. For the last several years, our group has been building knowledge bases with scientific collaborators. Using our approach, we have built knowledge bases that have comparable and sometimes better quality than those constructed by human volunteers. In contrast to these knowledge bases, which took experts a decade or more human years to construct, many of our projects are constructed by a single graduate student.

Our approach to KBC is based on joint probabilistic inference and learning, but we do not see inference as either a panacea or a magic bullet: inference is a tool that allows us to be systematic in how we construct, debug, and improve the quality of such systems. In addition, inference allows us to construct these systems in a more loosely coupled way than traditional approaches. To support this idea, we have built the DeepDive system, which has the design goal of letting the user "think about features—not algorithms." We think of DeepDive as declarative in that one specifies what they want but not how to get it. We describe our approach with a focus on feature engineering, which we argue is an understudied problem relative to its importance to end-to-end quality.

1 Introduction

This document highlights what we believe is a critical and underexplored aspect of building high-quality knowledgebase construction (KBC) systems: ease of feature engineering. The hypothesis of our work is that the easier the feature engineering process is to debug and diagonse, the easier it is to improve the quality of the system. The single most important design goal of DeepDive is to make KBC systems easier to debug and improve. Although such techniques are important for end-to-end quality, techniques to debug KBC systems are understudied.

To describe our thoughts more precisely, we describe our motivation for building KBC systems, our choice of language that defines the syntax of our feature engineering problem, and a set of debugging techniques that we have found useful:[1]

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1]This document is superficial, and we refer the interested reader to more detailed material including example code and data that are available online. `http://deepdive.stanford.edu` contains the latest material and is under active construction.

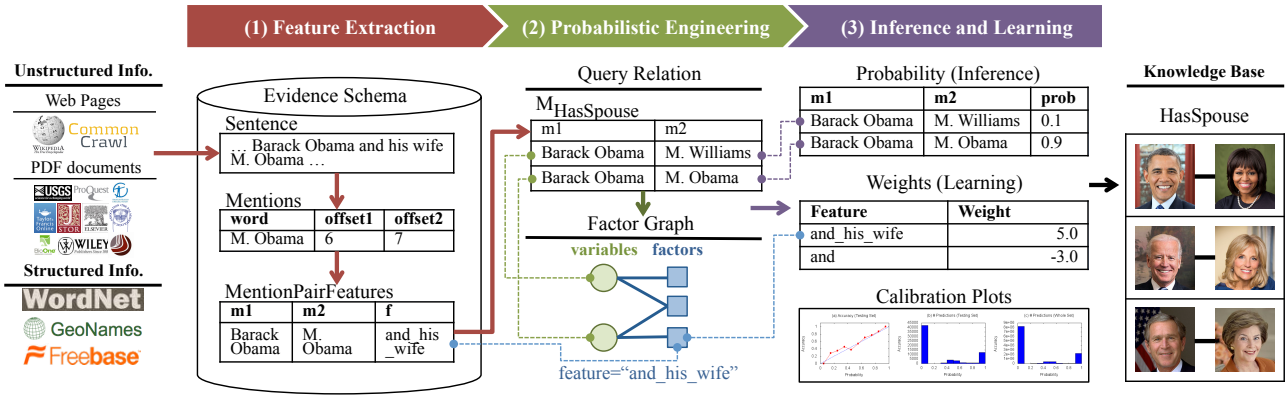


Figure 1: An overview of a KBC system built with DeepDive that takes as input both structured and unstructured information and creates a knowledge base as the output. There are three phases of DeepDive's execution model: (1) Feature Extraction; (2) Probabilistic Engineering; and (3) Inference and Learning. Section 2 contains a more detailed walkthrough of these phases, and Figure 4 shows more details of how to conduct tasks in these three phases using SQL and script languages, e.g., Python.

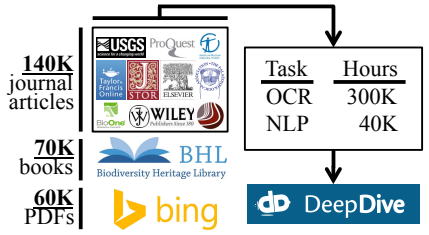
Motivation. In Section 2, we discuss a key motivating application for DeepDive, KBC. We take a holisitic approach that performs integration and extraction as a single task (rather than as two separate tasks). This holistic approach allows is to acquire data at all levels, including from larger and more diverse corpora, more human annotations, and/or larger (but still incomplete) knowledge bases. In turn, we can choose the source of data that has the best available signal for the least cost, a concept that we call *opportunistic acquisition*.

Language. In Section 2, we describe our choice of language model with the goal of abstracting the details of statistical learning and inference from the user to the extent possible. To integrate domain knowledge and support standard tools on top of DeepDive, we built our system on the relational model. Originally, we had our own custom-built language for inference based on Markov Logic [30], but we found that this language was unfamiliar to scientists. Instead, we decided to be as language agnostic as possible. A scientist can now write in almost any language to engineer features in DeepDive. However, for scalability reasons, bulk operations are written in SQL.

Debugging. In Section 4, we describe our approach to debugging. A key challenge is how to identify the relevant domain knowledge to integrate and what types of input sources have valuable signal. In our experience, without a systematic error analysis, it is not uncommon for developers to add rules that do not significantly improve the quality, which we call *prematurely optimizing* the KBC system. This is analogous to the popular engineering mistake of optimizing code without first profiling its end-to-end performance, akin to a failure to appreciate Amdahl's law. We view the key contribution of this work as highlighting the importance of error analysis in KBC applications.

Performance is also a major challenge. In our KBC systems using DeepDive, we may need to perform inference and learning on billions of highly correlated random variables. Therefore, one of our technical focus areas has been to speed up probabilistic inference [41, 40, 24, 23, 27]. Although this challenge is important, we do not focus on it in this paper, but there is much interesting work in this direction [5, 14, 4, 34, 31, 7] including work on how to automatically select algorithms from a declarative specification [17, 25, 25, 27]

(a) PaleoDeepDive Dataflow



(b) Quality Assessment

Relation	PDD # Ext.	PDD Acc.	Human Acc.
Taxon-Taxon	27 M	97%	92%
Taxon-Fm.	4 M	96%	84%
Fm.-Time	3 M	92%	89%
Fm.-Location	5 M	94%	90%

(c) Scientific Application: Biodiversity

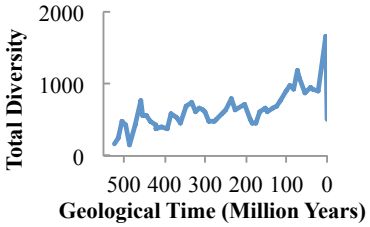


Figure 2: PaleoDeepDive, a KBC system for paleontology built with DeepDive. (a) The dataflow of PaleoDeepDive; (b) The assessment of quality for extractions in PaleoDeepDive (PDD); (c) One scientific application of PaleoDeepDive—biodiversity during the Phanerozoic period. More detailed results can be found in Peters et al. [28].

2 Knowledge Base Construction: The Case for Opportunistic Acquisition

Knowledge base construction is the process of populating a knowledge base with facts extracted from text, tabular data expressed in text and in structured forms, and even maps and figures. In *sample-based science* [28], one typically assembles a large number of facts (typically from the literature) to understand macroscopic questions, e.g., about the amount of carbon in the Earth's atmosphere throughout time, the rate of extinction of species, or all the drugs that interact with a particular gene. To answer such questions, a key step is to construct a high-quality knowledge base, and some forward-looking sciences have undertaken decade-long sample collection efforts, e.g., PaleoDB.org and PharmaGKB.org.

In parallel, KBC has attracted interest from industry [10, 42] and academia [18, 32, 33, 22, 9, 37, 2, 15, 3, 6, 29, 26]. To understand the common patters in KBC systems, we are actively collaborating with scientists from a diverse set of domains, including geology [38], paleontology [28], pharmacology for drug repurposing, and others. We briefly describe an application that we have constructed.

Example 1 (Paleontology and KBC [28]): Paleontology is based on the description and biological classification of fossils, an enterprise that has played out in countless collecting expeditions, museum visits, and an untold number of scientific publications over the past four centuries. One central task for paleontology is to construct a knowledge base about fossils from scientific publications, and an existing knowledge base compiled by human volunteers has greatly expanded the intellectual reach of paleontology and led to many fundamental new insights into macroevolutionary processes and the nature of biotic responses to global environmental change. However, the current process of using human volunteers is usually expensive and time-consuming. For example, PaleoDB, one of the largest such knowledge bases, took more than 300 professional paleontologists and 11 human years to build over the last two decades, resulting in `PaleoDB.org`. To get a sense of the impact of this database on this field, at the time of writing, this dataset has contributed to 205 publications, of which 17 have appeared in *Nature* or *Science*.

This provided an ideal test bed for our KBC research. In particular, we constructed a prototype called PaleoDeepDive [28] that takes in PDF documents. As a result, this prototype attacks challenges in optical character recognition, natural language processing, and information extraction and integration. Some statistics about the process are shown in Figure 2(a). As part of the validation of this system, we performed a double-blind experiment to assess the quality of the system versus the PaleoDB. We found that the KBC system built on DeepDive has achieved comparable—and sometimes better—quality than a knowledge base built by human volunteers over the last decade [28]. Figure 2(b) illustrates the accuracy of the results in PaleoDeepDive.

We have found that text is often not enough: often, the data that are interesting to scientists are located in

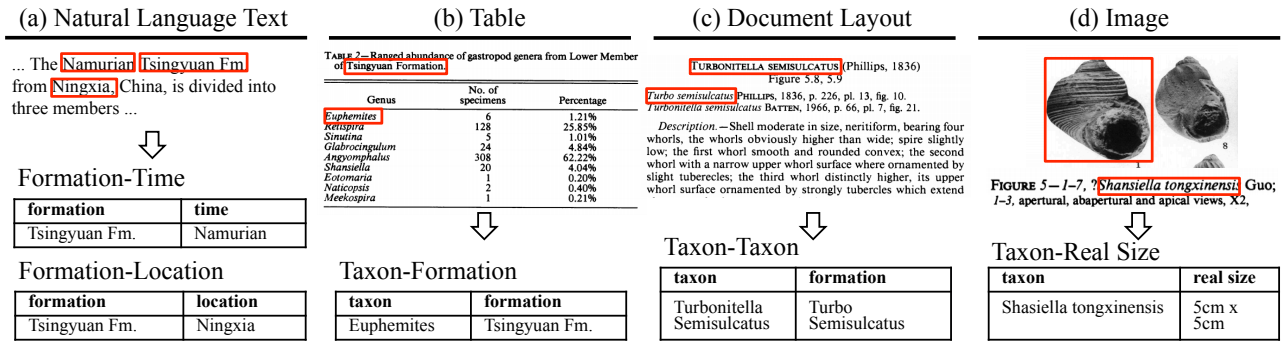


Figure 3: An illustration of data sources that PaleoDeepDive supports for KBC.

the tables and figures of articles. For example, in geology, more than 50% of the facts that we are interested in are buried in tables [11]. For paleontology, the relationship between taxa, as known as taxonomy, is almost exclusively expressed in section headers [28]. For pharmacology, it is not uncommon for a simple diagram to contain a large number of metabolic pathways. To build a KBC system with the quality that scientists will be satisfied with, we need to deal with these diverse sources of input. Additionally, external sources of information (other knowledge bases) typically contain high-quality signals (e.g., Freebase[2] and Macrostrat[3]). Leveraging these sources in information extraction is typically not studied in the classical information extraction context. To perform high-quality and high-coverage knowledge extraction, one needs a model that is able to ingest whatever presents itself *opportunistically*—that is, it is not tied solely to text but can handle more general extraction and integration.

Opportunistic Acquisition. We outline why we believe that the system must be able to *opportunistically acquire* data, by which we mean *the ability to acquire data from a wide variety of sources only to the degree to which they improve the end-to-end result quality*.

We describe the mechanisms by which we have improved the quality in KBC applications.[4] Consider one pattern that we observed in each of these KBC systems: we start from a state-of-the-art relation extraction model, e.g., a logistic regression model with rich linguistic features [21, 13]. Out of the box, such models do not have the quality that a domain expert requires. As a result, the expert must improve the model. This process of improving the model accounts for the dominant amount of time in which they interact with the system. To improve the extraction quality in terms of both precision and recall, the system needs some additional domain knowledge. For example, paleontologists might want to add a rule to improve the recall of a KBC system by informing the system about geological time; e.g., if a fossil appears in Late Cretaceous, then it also appears in the Cretaceous period. Biologists might want to add a rule to improve precision stating that one mention should only be recognized as a certain class of drug if there is clear linguistic clue that the drug is actually administered to a patient. These rules, however, must be expressed by domain scientists explicitly for the KBC system. Allowing users to express their knowledge is a challenge that has been studied in the extraction literature for some time, notably by SystemT [18, 19], which allows users to express their knowledge using declarative queries. In the next section, we describe our joint or collective approach, which is combined with a powerful technique called *distant supervision* that has allowed us to build knowledge bases with low cost. This technique allows DeepDive to take in this knowledge in many ways: rules, features, example datasets, and labeled data.

Choosing how to improve the system in the most effective way is arguably the key pain point in KBC. Nevertheless, we have noticed that there is a tendency to *prematurely optimize the quality* of intermediate results

[2] `http://www.freebase.com/`

[3] `http://macrostrat.org/`

[4] A complete list of the features we used for PaleoDeepDive can be found in our paper [28].

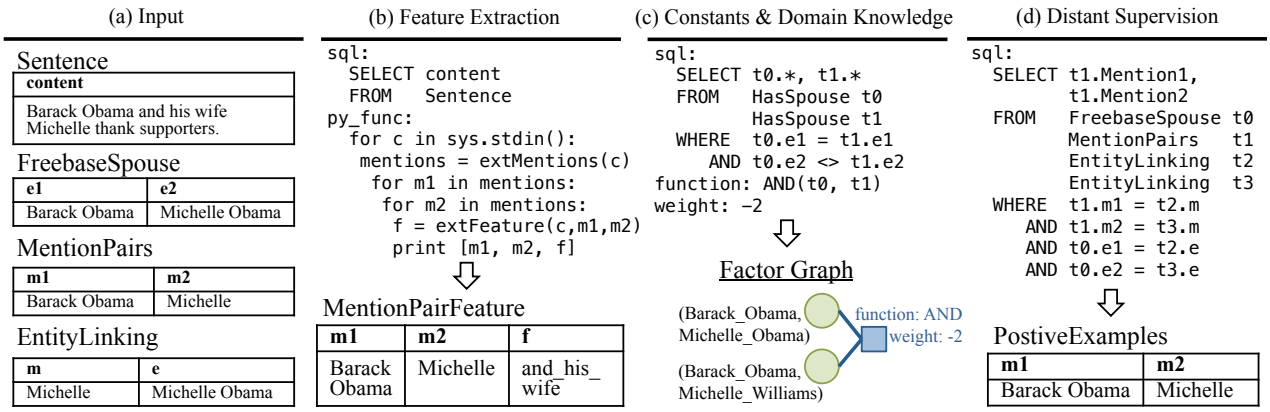


Figure 4: Illustration of popular operations in DeepDive. (a) Prepare data sets in relational form that can be used by DeepDive. (b) Generate labels using distant supervision with SQL; (c) Integrate constraints with SQL and logic functions; (d) Extract features with SQL and script languages (e.g., Python).

if the system is constructed in the absence of end-to-end goals. In our experience, we found that many of these premature optimizations seem reasonable in isolation and do locally improve the quality of the system. However, such optimizations have only marginal improvement on the end-to-end quality. As there is a never ending see of intermediate fixes prioritizing these fixes seems to be the critical issue. As a result, we advocate that one should prioritze how to improve the quality of the system based on how it affects the end-to-end quality of the system. We describe our first cut of how to make these choices in Section 4, with an emphasis on avoiding premature optimization.

3 A Brief Overview of DeepDive

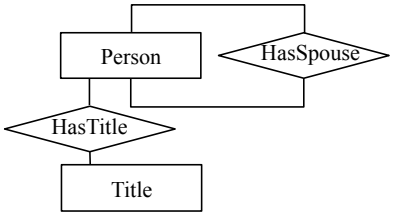
We briefly introduce the programming and execution model of DeepDive. Figure 4 shows the code that a user writes to interact with DeepDive. The reader can find a more detailed description of the language model of DeepDive in our previous work [26, 40] and on DeepDive's Web site. There are three phases in DeepDive's execution model, as shown in Figure 1:

(1) Feature Extraction. The input to this stage often contains both structured and unstructured information, and the goal is to produce a relational database that describes the features or signals of the data, which we call the *evidence schema*. We use the phrase evidence to emphasize that, in this stage, data is not required to be precisely correct as in traditional ETL; as a result, this is a much lighter ETL process.

- One responsibility of this phase is to run various OCR and NLP tools to acquire information from text, HTML, or images. The output database is often unnormalized: it may contain JSON objects to represent parse trees or DOM structures. This phase is essentially a high-throughput workflow system and may involve MapReduce jobs, Condor jobs, and SQL queries.
- The user also performs feature extraction in that they write user-defined functions (UDF) over existing query and evidence relations. DeepDive supports both ways, and the user can use SQL queries, and script languages, e.g., Python or Perl, to specify UDFs. Figure 4(b) and (d) show two examples.

(2) Probabilistic Engineering. The goal of this phase is to transform the evidence schema into a probabilistic model, specifically a factor graph that specifies:

(a) ER Diagram



(b) KBC Model

M_{Person}

id	Mention	Entity	offset
1	Michelle	Michelle_Obama	doc1_char1-8

M_{Title}

id	Mention	Entity	offset
4	Sen.	Senator	doc2_char5-8

$M_{HasSpouse}$

id	Mention1	Mention2
2	Barack Obama	Michelle

$M_{HasTitle}$

id	Mention1	Mention2
5	Obama	Sen.

HasSpouse

id	Entity1	Entity2
3	Barack_Obama	Michelle_Obama

HasTitle

id	Entity1	Entity2
6	Barack_Obama	Senator

Figure 5: Illustration of the KBC model in DeepDive. (a) An example ER diagram for the TAC-KBP application. (b) The corresponding KBC model, where each tuple in each relation corresponds to one random variable that DeepDive will predict. The column "offset" in mention tables M_{Person} and M_{Title} is the pointer to the position in the text, e.g., character offset in a document, and might be more complicated in real applications, e.g., a bounding box in a PDF.

- The set of random variables that the user wants to model. To define the set of random variables in DeepDive is to create new relations called query relations, in which each tuple corresponds to one random variable. This operation can be done by using SQL queries on existing relations.
- How those random variables are correlated; e.g., "The mention 'Obama' refers to the president is correlated with the random variable that indicates whether 'Obama' is a person." To specify *how*, the user specifies a factor function. Figure 4(c) shows a rough example that describes the intuition that *"people tend to be married to only a single person."* One (of many ways) to say this is to say that there is some correlation between pairs of married tuples; i.e., using the logical function AND(t0, t1) returns 1 in possible worlds in which both married tuples are true and 0 in others. DeepDive then learns the "strength" of this correlation from the data, which is encoded as weight.[5] Here, -2 indicates that it is less likely that both married tuples are correct. This phase is also where the user can write logical constraints, e.g., hard functional constraints.

Our previous work has shown that this grounding phase [23] can be a serious bottleneck if one does not use scalable relational technology. We have learned this lesson several times.

(3) Inference and Learning. This phase is largely opaque to the user: it takes the factor graph as input, estimates the weights, performs inference, and produces the output database along with a host of diagnostic information, notably calibration plots (see Fig. 6). More precisely, the output of DeepDive is a database that contains each random variable declared by the user with its marginal probability. For example, one tuple in the relation HasSpouse might be (Barack Obama, Michelle Obama), and ideally, we hope that DeepDive outputs a larger probability for this tuple as output.

3.1 Operations to Improve Quality in DeepDive

We illustrate the KBC model using one application called TAC-KBP,[6] in which the target knowledge base contains relations between persons, locations, and organizations. As is standard in the information extraction literature, a mention is a sequence of tokens in text, while an entity refers to the real-world object in the database. Figure 5(a) shows an excerpt from the ER diagram of the database. In this diagram, we have two types of entities, Person and Title, and two relations, HasSpouse and HasTitle. Figure 5(b) shows the KBC model induced

[5] A weight is roughly the log odds, i.e., the $\log \frac{p}{1-p}$ where p is the marginal probability of this random variable. This is standard in Markov Logic Networks [30], on which much of DeepDive's semantics are based.

[6] http://www.nist.gov/tac/2013/KBP/

from this ER diagram. For example, consider the Person entity and the HasSpouse relation. For Person, the KBC model contains a relation M_{Person}, which contains the candidate linking between a person mention (e.g., Michelle) to the person entity (e.g., Michelle_Obama or Michelle_Williams). The relation $M_{HasSpouse}$ contains mention pairs, which are candidates that participate in the HasSpouse relation, and the relation $HasSpouse$ contains the entity pairs. Each tuple of these relations corresponds to a Boolean random variable.

Routine Operations in DeepDive. We describe three routine tasks that a user performs to improve a KBC system.

An Example of Feature Extraction. The concept of a feature is one of the most important concepts for machine learning systems, and DeepDive's data model allows the user to use any scripting language for feature extraction. Figure 4(b) shows one such example using Python. One baseline feature that is often used in relation extraction systems is the word sequence between mention pairs in a sentence [21, 13], and Figure 4(b) shows an example of extracting this feature. The user first defines the input to the feature extractor using an SQL query, which selects all available sentences. Then the user defines a UDF that will be executed for each tuple returned by the SQL query. In this example, the UDF is a Python function that first reads a sentence from STDIN, extracts mentions from the sentence, extracts features for each mention pair, and outputs the result to STDOUT. DeepDive will then load the output of this UDF to the MentionPairFeature relation.

Constraints and Domain Knowledge. One way to improve a KBC system is to integrate domain knowledge, as we mentioned in Section 2. DeepDive supports this operation by allowing the user to integrate constraints and domain knowledge as correlations among random variables, as shown in Figure 4(c).

Imagine that the user wants to integrate a simple rule that says "one person is likely to be the spouse of only one person." For example, given a single entity "Barack_Obama," this rule gives positive preference to the case where only one of (Barack_Obama, Michelle_Obama) and (Barack_Obama, Michelle_Williams) is true. Figure 4(c) shows one example of implementing this rule. The SQL query in Figure 4(c) defines a view in which each tuple corresponds to two relation candidates with the same first entity but different second entities. The function AND(t0, t1) defines the "type of correlation" among variables, and the weight "-2" defines the strength of the correlation. This rule indicates that the it is less likely that both (Barack_Obama, Michelle_Obama) and (Barack_Obama, Michelle_Williams) are true (i.e., when AND(t0, t1) returns 1). Typically, DeepDive is used to learn the weights from data.

Distant Supervision. One challenge with building a machine learning system for KBC is generating training examples. As the number of predicates in the system grows, specifying training examples for each relation is tedious and expensive. One common technique to cope with this is distant supervision. Distant supervision starts with an (incomplete) entity-level knowledge base and a corpus of text. The user then defines a (heuristic) mapping between entities in the database and text. This map is used to generate (noisy) training data for mention-level relations [21, 13]. We illustrate this procedure by example.

Example 2 (Distant Supervision): Consider the mention-level relation $M_{HasSpouse}$. To find training data, we find sentences that contain mentions of pairs of entities that are married, and we consider the resulting sentences positive examples. Negative examples could be generated from pairs of persons who are, say, parent-child pairs. Ideally, the patterns that occur between pairs of mentions corresponding to mentions will contain indicators of marriage more often than those that are parent-child pairs (or other pairs). Selecting those indicative phrases or features allows us to find features for these relations and generate training data. Of course, engineering this mapping is a difficult task in practice and requires many iterations.

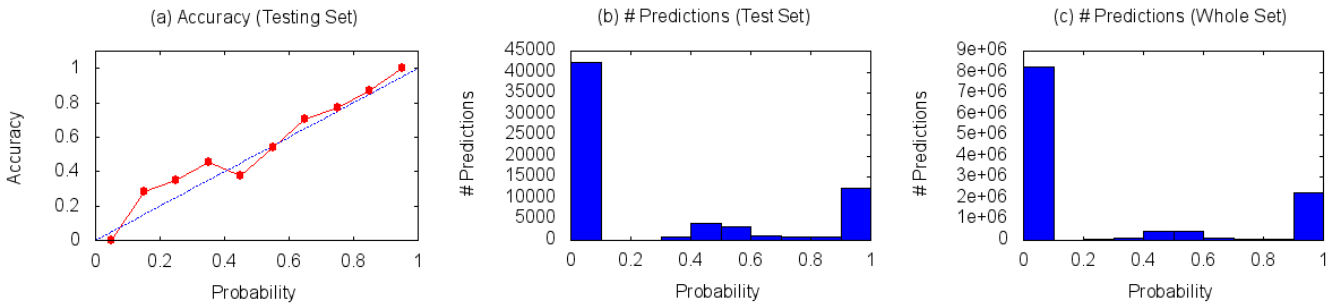


Figure 6: Illustration of calibration plots in DeepDive.

Concretely, Figure 4(d) shows an example of distant supervision in DeepDive. The FreebaseSpouse relation is an entity-level relation in Freebase (it contains pairs of married people). The EntityLinking relation specifies a mapping from entities to mentions in the text. The user provides an SQL query like the one shown in Figure 4(d) to produce another relation, PositiveExamples, that contains mention-level positive training examples. In our applications, users may spend time improving this mapping, which can lead to higher quality (but imperfect) training data much more cheaply than having a human label this data.

As we have described, the user has at least the above three ways to improve the system and is free to use one or a combination of them to improve the system's quality. The question we address next is, *"What should the user do next to get the largest quality improvement in the KBC system?"*

4 Debugging and Improving a KBC System

A DeepDive system is only as good as its features and rules. In the last two years, we have found that understanding which features to add is the most critical—but often the most overlooked—step in the process. Without a systematic analysis of the errors of the system, developers often add rules that do not significantly improve their KBC system, and they settle for suboptimal quality. In this section, we describe our process of error analysis, which we decompose into two stages: a macro-error analysis that is used to guard against statistical errors and gives an at-a-glance description of the system and a fine-grained error analysis that actually results in new features and code being added to the system.

4.1 Macro Error Analysis: Calibration Plots

In DeepDive, *calibration plots* are used to summarize the overall quality of the KBC results. Because DeepDive uses a joint probability model, each random variable is assigned a marginal probability. Ideally, if one takes all the facts to which DeepDive assigns a probability score of 0.95, then 95% of these facts are correct. We believe that probabilities remove a key element: the developer reasons about features, not the algorithms underneath. This is a type of *algorithm independence* that we believe is critical.

DeepDive programs define one or more test sets for each relation, which are essentially a set of labeled data for that particular relation. This set is used to produce a calibration plot. Figure 6 shows an example calibration plot for the Formation-Time relation in PaleoDeepDive, which provides an aggregated view of how the KBC system behaves. By reading each of the subplots, we can get a rough assessment of the next step to improve our KBC system. We explain each component below.

As shown in Figure 6, a calibration plot contains three components: (a) accuracy, (b) # predictions (test set), which measures the number of extractions in the test set with a certain probability; and (c) # predictions

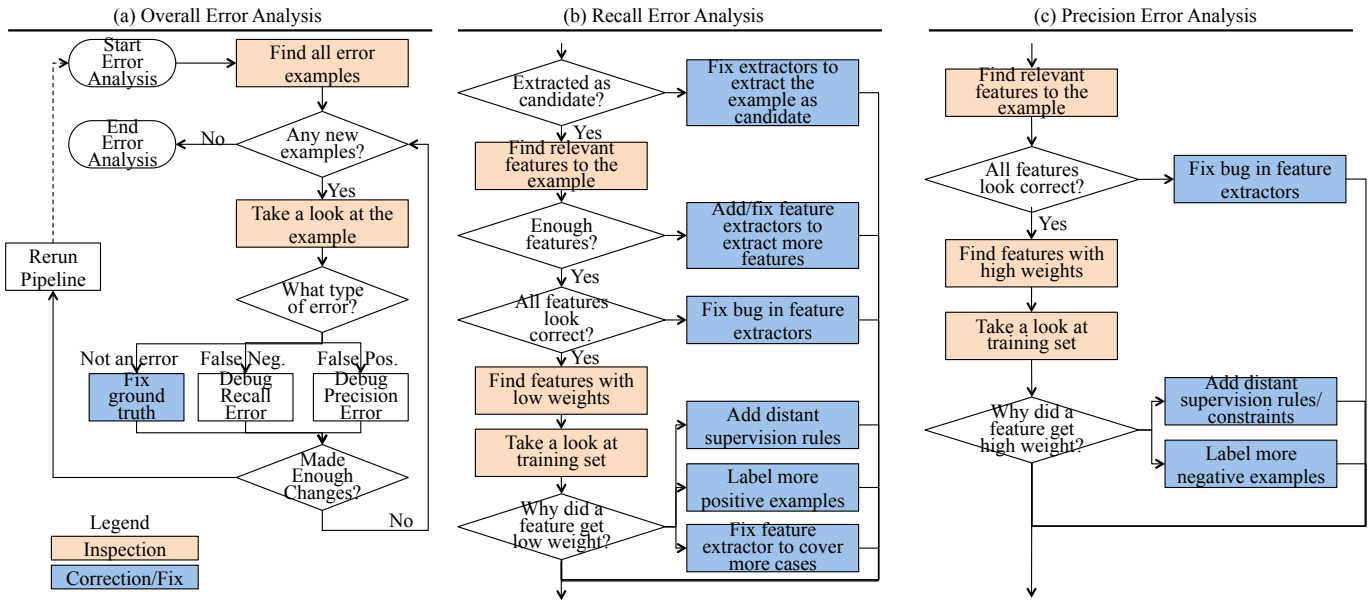


Figure 7: Error Analysis Workflow of DeepDive.

(whole set), which measures the number of extractions in the whole set with a certain probability. The test set is assumed to have labels so that we can measure accuracy, while the whole set does not.

(a) Accuracy. To create the accuracy histogram, we bin each fact extracted by DeepDive on the test set by the probability score assigned to each fact; e.g., we round to the nearest value in the set $k/10$ for $k = 1, \ldots, 10$. For each bin, we compute the fraction of those predictions that is correct. Ideally, this line would be on the (0,0)-(1,1) line, which means that the DeepDive-produced probability value is calibrated, i.e., it matches the *test-set accuracy*. For example, Figure 6(a) shows a curve for calibration. Differences in these two lines can be caused by noise in the training data, quantization error due to binning, or sparsity in the training data.

(b) # Predictions (Testing Set). We also create a histogram of the number of predictions in each bin. In a well-tuned system, the # Predictions histogram should have a "U" shape. That is, most of the extractions are concentrated at high probability and low probability. We do want a number of low-probability events, as this indicates DeepDive is considering plausible but ultimately incorrect alternatives. Figure 6(b) shows a U-shaped curve with some masses around 0.5-0.6. Intuitively, this suggests that there is some hidden type of example for which the system has insufficient features. More generally, facts that fall into bins that are not in (0,0.1) or (0.9,1.0) are candidates for improvements, and one goal of improving a KBC system is to "push" these probabilities into either (0,0.1) or (0.9,1.0). To do this, we may want to sample from these examples and add more features to resolve this uncertainty.

(c) # Predictions (Whole Set). The final histogram is similar to Figure 6(b), but illustrates the behavior of the system, for which we do not have any training examples. We can visually inspect that Figure 6(c) has a similar shape to (b); If not, this would suggest possible overfitting or some bias in the selection of the hold-out set.

4.2 Micro Error Analysis: Per-Example Analysis

Calibration plots provide a high-level summary of the quality of a KBC system, from which developers can get an abstract sense of how to improve it. For developers to produce rules that can improve the system, they need to actually look at data. We describe this fine-grained error analysis, illustrated in Figure 7.

Figure 7(a) illustrates the overall workflow of error analysis. The process usually starts with the developer finding a set of errors, which is a set of random variables having predictions that are inconsistent with the training data. Then, for each random variable, the developer looks at the corresponding tuples in the user relation (recall that each tuple in the user relation corresponds to a random variable) and tries to identify the types of errors. We describe two broad classes of errors that we think of as improving the recall and the precision, which we describe below.

Recall Error Analysis. A recall error, i.e., a false negative error, occurs when DeepDive is expected to extract a fact but fails to do so. In DeepDive's data model, this usually means that a random variable has a low probability or that the corresponding fact is not extracted as a candidate. Figure 7(b) illustrates the process of debugging a recall error.

First, it is possible that the corresponding fact that is expected to be extracted does not appear as a candidate. In this case, there is no random variable associated with this fact, so it is impossible for DeepDive to extract it. For example, this case might happen when the corresponding fact appears in tables, but we only have a text-based extractor. In this case, the developer needs to write extractors to extract these facts as candidates.

Another common scenario is when the corresponding fact appears as a candidate but is not assigned a high enough probability by DeepDive. In this case, the developer needs to take a series of steps, as shown in Figure 7(b), to correct the error. First, it is possible that this candidate is not associated with any features or that the corresponding features have an obvious bug. In this case, one needs to fix the extractors of the features. Second, it is possible that the corresponding features have a relatively low weight learned by DeepDive. In that case, one needs to look at the training data to understand the reason. For example, one might want to add more distant supervision rules to increase the number of positive examples that share the same feature or even manually label more positive examples. One might also want to change the feature extractors to collapse the current features with other features to reduce the sparsity.

Precision Error Analysis. A precision error, i.e., a false positive error, occurs when DeepDive outputs a high probability for a fact, but it is an incorrect one or not supported by the corresponding document. Figure 7(c) illustrates the process of debugging a precision error. Similar to debugging recall errors, for precision errors, the developer needs to look at the features associated with the random variable. The central question to investigate is why some of these features happen to have high weights. In our experience, this usually means that we do not have enough negative examples for training. If this is the case, the developer adds more distant supervision rules or (less commonly) manually labels more negative examples.

Avoiding Overfitting. One possible pitfall for the error analysis at the per-example level is overfitting, in which the rules written by the developer overfit to the corpus or the errors on which the error analysis is conducted. To avoid this problem, we have insisted in all our applications on conducting careful held-out sets to produce at least (1) a training set; (2) a testing set; and (3) an error analysis set. In the error analysis process, the developer is only allowed to look at examples in the error analysis set, and validate the score on the testing set.

5 Related Work

Knowledge Base Construction (KBC) has been an area of intense study over the last decade [18, 32, 33, 22, 9, 37, 2, 15, 3, 6, 42, 29]. Within this space, there are a number of approaches.

Rule-Based Systems. The earliest KBC systems used pattern matching to extract relationships from text. The most well-known example is the "Hearst Pattern" proposed by Hearst [12] in 1992. In her seminal work, Hearst observed that a large number of hyponyms can be discovered by simple patterns, e.g., "X such as Y." Hearst's technique has formed the basis of many further techniques that attempt to extract high-quality patterns from text. Rule-based (pattern matching-based) KBC systems, such as IBM's SystemT [18, 19], have been built to aid developers in constructing high-quality patterns. These systems provide the user with a (declarative) interface to specify a set of rules and patterns to derive relationships. These systems have achieved state-of-the-art quality on tasks such as parsing [19].

Statistical Approaches. One limitation of rule-based systems is that the developer needs to ensure that all rules provided to the system are high-precision rules. For the last decade, probabilistic (or machine learning) approaches have been proposed to allow the system to select from a range of a priori features automatically. In these approaches, the extracted tuple is associated with a marginal probability that it is true. DeepDive, Google's knowledge graph, and IBM's Watson are built on this approach. Within this space, there are three styles of systems:

- **Classification-Based Frameworks.** Here, traditional classifiers assign each tuple a probability score, e.g., a naïve Bayes classifier or a logistic regression classifier. For example, KnowItAll [9] and TextRunner [37, 2] use a naïve Bayes classifier, and CMUs NELL [3, 6] uses logistic regression. Large-scale systems typically use these types of approaches in sophisticated combinations, e.g., NELL or Watson.
- **Maximum a Posteriori (MAP).** Here, a probabilistic approach is used, but the MAP or most likely world (which do differ slightly) is selected. Notable examples include the YAGO system [15], which uses a PageRank-based approach to assign a confidence score. Other examples include SOFIE [33] and Prospera [22], which use an approach based on constraint satisfaction.
- **Graphical Model Approaches.** The classification-based methods ignore the interaction among predictions, and there is a hypothesis that modeling these correlations yields higher quality systems more quickly. A generic graphical model has been used to model the probabilistic distribution among all possible extractions. For example, Poon et al. [29] used Markov logic networks (MLN) [8] for information extraction. Microsoft's StatisticalSnowBall/EntityCube [42] also uses an MLN-based approach. A key challenge in these systems is scalability. For example, Poon et al. was limited to 1.5K citations. Our relational database-driven algorithms for MLN-based systems are dramatically more scalable [23].

6 Future Work

DeepDive is our first step toward facilitating the building of knowledge base construction systems of sufficient quality to support applications, including rigorous scientific discoveries. Given our experience in interacting with scientists and understanding their needs, we found the following directions interesting candidates for future exploration, most of which seem to be challenging open research problems.

- **Dealing with Time and Temporal Information.** The KBC model we describe in this work treats time as a distinguished predicate such that each fact is extracted as long as there exists a time point such that it is true. However, prior work has shown that each fact could be associated with a more expressive temporal

tag [20, 36, 35], and our previous participation in the DARPA machine reading challenge also attempted to produce a KBC system with such tags. From our conversations with scientists, we have found that these temporal tags are sometimes important. For example, in paleontology, where the corpus spans more than 400 years, the ability to find the most recent version of the fact is important to produce the up-to-date view of the tree of life. It is interesting to study how to integrate temporal information into DeepDive's KBC model and conduct reasoning over it.

- **Visual Information Extraction.** As we mentioned in Section 2, we observed that there is a vast amount of information buried in data sources such as images and diagrams. Although the current DeepDive system can perform some simple image-based extraction, e.g., to extract body size from paleontology publications [28], it is an interesting direction to study how to extract information from tables, charts, and diagrams.
- **Incremental Processing.** One observation we have about debugging a KBC system is that, to conduct error-analysis efficiently, one usually needs to rerun the system with slightly different DeepDive programs (e.g., more rules) and data (e.g., more structural resources). In our application, it is not uncommon for the factor graph to contain billions of random variables. One obvious solution to this is to study how to incrementally conduct inference and learning to support more efficient error analysis. This problem is distinct from incremental or online learning, as the focus is on maintaining the downstream data products. We have done some work in this direction in a simplified classifier-based setting [16].
- **"Active Debugging" and Rule Learning.** The current debugging infrastructure in DeepDive is "passive" in that DeepDive waits for instructions from the user for error analysis or adding more features. One future direction is to study whether DeepDive can play a more "active" role. One example is whether DeepDive could automatically suggest errors for the users to investigate or report relations where more training data are required. It is also interesting to study whether it is possible for DeepDive to automatically learn inference rules or feature extractors and recommend them to the user. We envisioned a similar system in our previous work [1] and have worked on a domain-specific language for a related problem [39].
- **Even-More-joint Information Extraction.** As we show in this work, one of DeepDive's advantage is the ability to jointly take advantage of different sources of signals. One interesting question is whether we can extend the current KBC model to be even more joint. First, it is interesting to study how to use DeepDive for low-level NLP tasks, e.g., linguistic parsing and optical character recognition, and how these low-level tasks interact with high-level tasks, e.g., relation extraction. Second, it is interesting to study how application-level knowledge, e.g., the R-script used by paleontologists over DeepDive's knowledge base, can be used to constrain and improve the extraction task.
- **Visualization of Uncertainty.** The current DeepDive system expresses the uncertainty of inference results to the user using marginal probabilities. Even simple visualizations have proved to be far more effective at conveying debugging information about statistical or aggregate properties.

7 Conclusion

We have described what we believe is one of the key features of DeepDive: the ability to rapidly debug and engineer a better system. We have argued that probabilistic and machine learning techniques are critical, but only in that they enable developers to think in terms of features—not algorithms.

8 Acknowledgments

We would like to thank the users of DeepDive, especially Shanan Peters and Emily Doughty, who have given a great deal of helpful (and patient) feedback. We would also like to thank Michael J. Cafarella and Dan Suicu

who gave comments on an earlier draft of this document. 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, DARPA's MEMEX program, the National Science Foundation (NSF) CAREER Award under No. IIS-1353606 and EarthCube Award under No. ACI-1343760, the Office of Naval Research (ONR) under awards No. N000141210041 and No. N000141310129, the Sloan Research Fellowship, 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, or the U.S. government.

References

[1] M. Anderson, D. Antenucci, V. Bittorf, M. Burgess, M. J. Cafarella, A. Kumar, F. Niu, Y. Park, C. Ré, and C. Zhang. Brainwash: A data system for feature engineering. In *CIDR*, 2013.

[2] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the Web. In *IJCAI*, 2007.

[3] J. Betteridge, A. Carlson, S. A. Hong, E. R. Hruschka, E. L. M. Law, T. M. Mitchell, and S. H. Wang. Toward never ending language learning. In *AAAI Spring Symposium*, 2009.

[4] Z. Cai, Z. Vagena, L. L. Perez, S. Arumugam, P. J. Haas, and C. M. Jermaine. Simulation of database-valued Markov chains using SimSQL. In *SIGMOD*, 2013.

[5] J. Canny and H. Zhao. Big Data analytics with small footprint: Squaring the cloud. In *KDD*, 2013.

[6] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

[7] Y. Chen and D. Z. Wang. Knowledge expansion over probabilistic knowledge bases. In *SIGMOD*, 2014.

[8] P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2009.

[9] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in KnowItAll: (preliminary results). In *WWW*, 2004.

[10] D. A. Ferrucci, E. W. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. M. Prager, N. Schlaefer, and C. A. Welty. Building Watson: An overview of the DeepQA project. *AI Magazine*, 2010.

[11] V. Govindaraju, C. Zhang, and C. Ré. Understanding tables in context using standard NLP toolkits. In *ACL*, 2013.

[12] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING*, 1992.

[13] R. Hoffmann, C. Zhang, and D. S. Weld. Learning 5000 relational extractors. In *ACL*, 2010.

[14] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. MCDB: A Monte Carlo approach to managing uncertain data. In *SIGMOD*, 2008.

[15] G. Kasneci, M. Ramanath, F. Suchanek, and G. Weikum. The YAGO-NAGA approach to knowledge discovery. *SIGMOD Rec.*, 2009.

[16] M. L. Koc and C. Ré. Incrementally maintaining classification using an RDBMS. *PVLDB*, 2011.

[17] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. Mlbase: A distributed machine-learning system. In *CIDR*, 2013.

[18] R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu. SystemT: A system for declarative information extraction. *SIGMOD Rec.*, 2009.

[19] Y. Li, F. R. Reiss, and L. Chiticariu. SystemT: A declarative information extraction system. In *HLT*, 2011.

[20] X. Ling and D. S. Weld. Temporal information extraction. In *AAAI*, 2010.

[21] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL*, 2009.

[22] N. Nakashole, M. Theobald, and G. Weikum. Scalable knowledge harvesting with high precision and high recall. In *WSDM*, 2011.

[23] F. Niu, C. Ré, A. Doan, and J. Shavlik. Tuffy: Scaling up statistical inference in Markov logic networks using an RDBMS. *PVLDB*, 2011.

[24] F. Niu, B. Recht, C. Ré, and S. J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

[25] F. Niu, C. Zhang, C. Ré, and J. W. Shavlik. Felix: Scaling inference for Markov Logic with an operator-based approach. *ArXiv e-print.*, 2011.

[26] F. Niu, C. Zhang, C. Ré, and J. W. Shavlik. Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *Int. J. Semantic Web Inf. Syst.*, 2012.

[27] F. Niu, C. Zhang, C. Ré, and J. W. Shavlik. Scaling inference for Markov logic via dual decomposition. In *ICDM*, 2012.

[28] S. E. Peters, C. Zhang, M. Livny, and C. Ré. A machine-compiled macroevolutionary history of Phanerozoic life. *ArXiv e-prints*, 2014.

[29] H. Poon and P. Domingos. Joint inference in information extraction. In *AAAI*, 2007.

[30] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 2006.

[31] P. Sen, A. Deshpande, and L. Getoor. PrDB: Managing and exploiting rich correlations in probabilistic databases. *The VLDB Journal*, 2009.

[32] W. Shen, A. Doan, J. F. Naughton, and R. Ramakrishnan. Declarative information extraction using datalog with embedded extraction predicates. In *VLDB*, 2007.

[33] F. M. Suchanek, M. Sozio, and G. Weikum. SOFIE: A self-organizing framework for information extraction. In *WWW*, 2009.

[34] D. Z. Wang, M. J. Franklin, M. Garofalakis, J. M. Hellerstein, and M. L. Wick. Hybrid in-database inference for declarative information extraction. In *SIGMOD*, 2011.

[35] Y. Wang, B. Yang, S. Zoupanos, M. Spaniol, and G. Weikum. Scalable spatio-temporal knowledge harvesting. In *WWW*, 2011.

[36] Y. Wang, M. Zhu, L. Qu, M. Spaniol, and G. Weikum. Timely YAGO: harvesting, querying, and visualizing temporal knowledge from Wikipedia. In *EDBT*, 2010.

[37] A. Yates, M. Cafarella, M. Banko, O. Etzioni, M. Broadhead, and S. Soderland. TextRunner: Open information extraction on the Web. In *NAACL*, 2007.

[38] C. Zhang, V. Govindaraju, J. Borchardt, T. Foltz, C. Ré, and S. Peters. GeoDeepDive: statistical inference using familiar data-processing languages. In *SIGMOD*, 2013.

[39] C. Zhang, A. Kumar, and C. Ré. Materialization optimizations for feature selection workloads. In *SIGMOD*, 2014.

[40] C. Zhang and C. Ré. Towards high-throughput Gibbs sampling at scale: a study across storage managers. In *SIGMOD*, 2013.

[41] C. Zhang and C. Ré. DimmWitted: A study of main-memory statistical analytics. *PVLDB*, 2014.

[42] J. Zhu, Z. Nie, X. Liu, B. Zhang, and J.-R. Wen. StatSnowball: A statistical approach to extracting entity relationships. In *WWW*, 2009.