Extracting Databases from Dark Data with DeepDive

Ce Zhang, Jaeho Shin, Christopher Ré
Stanford University
Palo Alto, CA
{czhang,jaeho,chrismre}
@cs.stanford.edu

Michael Cafarella
University of Michigan
Ann Arbor, MI
michjc@umich.edu

Feng Niu
Lattice Data, Inc.
Palo Alto, CA
feng.niu@lattice.io

ABSTRACT

DeepDive is a system for extracting relational databases from *dark data*: the mass of text, tables, and images that are widely collected and stored but which cannot be exploited by standard relational tools. If the information in dark data — scientific papers, Web classified ads, customer service notes, and so on — were instead in a relational database, it would give analysts a massive and valuable new set of "big data."

DeepDive is distinctive when compared to previous information extraction systems in its ability to obtain very high precision and recall at reasonable engineering cost; in a number of applications, we have used DeepDive to create databases with accuracy that meets that of human annotators. To date we have successfully deployed DeepDive to create data-centric applications for insurance, materials science, genomics, paleontologists, law enforcement, and others. The data unlocked by DeepDive represents a massive opportunity for industry, government, and scientific researchers.

DeepDive is enabled by an unusual design that combines large-scale probabilistic inference with a novel developer interaction cycle. This design is enabled by several core innovations around probabilistic training and inference.

1. INTRODUCTION

DeepDive is a system for extracting structured data from unstructured *dark data*. Dark data is the mass of text, tables, and images that is collected and stored but which cannot be queried using traditional relational database tools. Consider the set of text notes about insurance claims, or scientific papers, or internet classified ads; these different sources contain valuable information that is not in a relational format, but in principle should be amenable to analysis by relational tools. Currently, organizations pay the cost of collecting and maintaining dark data, but cannot exploit it. DeepDive consumes dark data and populates a relational database that can be used with standard data management tools, such as OLAP query processors, visualization software like Tableau, and analytical tools such as R or Excel.

SIGMOD'16, June 26-July 01, 2016, San Francisco, CA, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-3531-7/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2882903.2904442

Dark data often holds information that is not available in any other format. For example, an insurance company's claim notes resemble a small blog dedicated to a single claim, with entries authored by many different individuals. They will contain service representatives' notes from customer interactions, bills from doctors who treat claimants, reports from car repair shops, and so on. Simple claims might have a single text note, but more complicated claims might have hundreds. If this information were in a relational database instead of text, it would be possible to use simple SQL queries to answer a range of useful questions, including:

- Which doctors were responsible for the most claims?
- Is the distribution of injury types changing over time?
- Do certain inspectors yield larger claims than others?

Because success in analytical tasks is often limited by the data available [18], the information embedded in dark data can be massively valuable. In some cases, the data is so valuable that institutions have hired human beings to manually read documents and populate a relational database by hand [37]. However, this practice is expensive, incredibly slow, and surprisingly error-prone.

DeepDive can extract structured databases from dark data accurately and vastly more quickly than human beings. Of course, information extraction has been an area of active academic investigation since at least the early 1990s [1]. DeepDive is distinctive because of its ability to produce databases of extremely high accuracy with reasonable amounts of human engineering effort. In a remarkable range of applications, DeepDive has been able to obtain data with precision that meets or beats that of human annotators. DeepDive's high quality is possible because of a unique design based around probabilistic inference and a specialized engineering development cycle; they are underpinned by several technical innovations for efficient statistical training and sampling.

DeepDive is an open source project that has been under development since 2011. We have used it to extract high-quality information in a number of disparate areas, including genomics, insurance, Web classified ads, materials science, paleontology, and others. We believe that creating and exploiting extracted information from dark data (nowadays often sitting around in rapidly-growing data lakes) represents a massive opportunity for academia, industry, and government. We have deployed many of these extracted datasets to domain-specific users and have been gratified to see the information enable novel and remarkable domain-specific results.

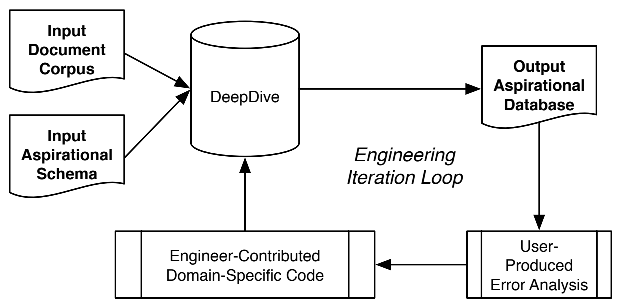

Input Document Corpus
Input Aspirational Schema
DeepDive
Output Aspirational Database
Engineering Iteration Loop
Engineer-Contributed Domain-Specific Code
User-Produced Error Analysis

Figure 1: The typical DEEPDIVE user interaction cycle. A knowledge engineer runs DEEPDIVE to obtain an output, evaluates its quality, then adjusts the input rules and runs DEEPDIVE again.

In this paper we will discuss the core elements of the DEEPDIVE software architecture, as well as a number of applications driven by DEEPDIVE, drawn from both academic and industrial settings. Many of the algorithms and technical components behind DEEPDIVE have been documented previously; this paper is the first to offer a comprehensive look at the software that ties the technical elements into a coherent whole. Although DEEPDIVE is designed to handle multiple media types, we will primarily focus on text.

In this paper we discuss:

- Design goals for DEEPDIVE (Section 2)
- DEEPDIVE's software architecture (Section 3)
- A brief overview of the technical components needed to implement the above architecture (Section 4)
- The DEEPDIVE development cycle (Section 5)
- An overview of DEEPDIVE-enabled applications, including insurance, materials science, and human trafficking (Section 6)

Finally, we conclude with a discussion of related and future work.

2. USER INTERACTION AND DESIGN CRITERIA

DEEPDIVE's overriding goal is to extract data with very high quality. Figure 1 shows DEEPDIVE's core inputs and outputs. The user presents DEEPDIVE with an *input document corpus* and an *input aspirational schema* to populate with information from that corpus. For example, if the *document corpus* were a collection of genomics research papers, the *aspirational schema* might be (gene, phenotype). DEEPDIVE then emits an *output aspirational table* that has the given schema and uses information found in the input corpus.

The success of a single DEEPDIVE run is determined by the quality — the precision and recall — of the output aspirational table. DEEPDIVE obtains high quality by asking an engineer to repeatedly examine the output table, diagnose any problems, and repair problems by contributing domain-specific code. By repeatedly going through this iterative improvement cycle, a DEEPDIVE engineer should be able to obtain extremely high accuracy data.

DEEPDIVE's success is entirely dependent on effective and rapid execution of this iterative development cycle. In this section we discuss the motivation behind this developer model, as well as the system design criteria driven by the model.

2.1 Iterative Development Model

We have observed that when working on an information extraction task, it is very easy to spend one's time poorly. Even well-trained computer scientists will work in an *ad hoc* fashion and contribute useless features, laboriously perfect features that have little impact, solve error cases that are obvious rather than common, fiddle with the statistical infrastructure, rarely know when to add more training data, and fall victim to common (though sometimes subtle) statistical errors. In short, even the best of us flail aimlessly in the face of a low precision or recall number.

This process superficially resembles that of *performance debugging*, an area where it is famously easy for a software engineer to spend huge amounts of time flailing, making changes that do not yield concrete results. But in contrast to extractor development, the software engineer who wants to make a slow piece of software run fast can pursue several methods that are very effective, if not always practiced.

An effective engineer knows the wisdom in Amdahl's Law, and will focus only on the slowest component. Identifying the slowest component is often counterintuitive, so she heavily instruments the code before making any changes. Finally, she knows that she should make performance improvements piecemeal, because solving one can change the overall picture of which components are now the bottleneck. By repeatedly applying this deeply-ingrained, reliable, and predictable discipline, the engineer can obtain a fast piece of software.

DEEPDIVE aims to use a similar development cycle to obtain high accuracy. The human engineer identifies the most common error in the output table. She then examines the relevant region of the raw input corpus, plus DEEPDIVE's instrumented extraction decision. This information enables the engineer to solve the error class, rerun the system, and move onto the next error. We describe the developer experience in more detail in Section 5.

2.2 System Design Criteria

In order to support the above developer workflow, DEEPDIVE has three core design criteria:

- **Mechanism independence** — Human developers improve accuracy by contributing domain information, *not* by changing the statistical procedure.
- **Integrated processing** — The interrelated steps of extraction, cleaning, and integration should all be handled simultaneously.
- **Debuggable decisions** — Any extraction error made by the system should be analyzable and tied to a root cause that can be understood and fixed by a human engineer.

2.3 Mechanism Independence

In an effort to obtain either high accuracy or faster runtimes, it is commonplace for machine learning engineers to modify the core statistical procedure for a given task. Machine learning conferences focus heavily on novel methods, often justified with experiments showing improved accuracy numbers for a workload of tasks. This practice is disastrous from an engineering point of view, encouraging the developer to fiddle with the most error-prone and difficult-to-debug portion of the system. It also distracts the developer from contributing what a human is uniquely able to contribute, and what usually solves real extraction problems: additional problem-specific domain knowledge.

Unlike machine learning toolkits such as MLib, MADLib, or SAS, DeepDive does not offer the user an elaborate suite of statistical algorithms. The training and inference procedure is not controllable by the developer. Rather, the developer uses a high-level datalog-like language called **DDlog** to describe the structured extraction problem. DeepDive then figures out how to most efficiently perform the extraction task described by the developer.

This approach should sound very familiar to readers in the database community, which has long favored high-level languages over direct user control of the query processing mechanism. However, this design choice posed a serious technical challenge well into the 1980s, until it eventually became known how to build a query processing system that could obtain acceptable performance. DeepDive faced a similar technical challenge: a system for probabilistic inference that is efficient enough that users can generally rely on it. Addressing this challenge was a key hurdle in building DeepDive (though DeepDive is not the only possible way to use these efficient training and inference techniques). We describe it in more detail in Section 4.

2.4 Integrated Processing

DeepDive handles in a single platform several processing steps that in the past have been treated as distinct: extraction, integration, and cleaning. This approach may seem unwisely ambitious, since each individual step has been the focus of dedicated software products. But in contrast to the past *siloed processing* model, we believe that *integrated processing* is crucial for supporting the developer workflow above. The key is that integrated processing allows the developer to solve extraction problems where they are easiest to solve.

Consider when the DeepDive developer wants to create a book catalog out of book review web pages, using a siloed system of two steps: extraction, followed by integration with a partial catalog. Imagine the target schema is **(bookTitle, author, uniqueId, price)** and the extractor has a very high precision of 98%. The remaining 2% of emitted tuples are not books, but are movies that were incorrectly extracted, due to a mistake made by the extractor's NLP parser. The downstream integration module fails to integrate some of the correct extractions (because they are novel) and all of the incorrect ones (because the existing database does not contain movies).

In a siloed system, the extractor will appear to be providing incorrect data to the integrator, which of course will fail. The team responsible for the integration system will naturally ask the extractor team to fix the problem, with the reasoning that no system can integrate incorrect data[1]. Unfortunately, NLP parsing problems are likely extremely difficult for the extraction team to repair, forcing them to either devote huge technical resources to solving the problem, or devote human resources to manually reviewing the extractor's output.

[1]Could the extractor team respond that 98% is "good enough"? There is no way to answer this question without an integrated end-to-end evaluation. Consider the similar problem of evaluating whether a given function in a piece of software is "fast enough." For a function considered in isolation, the question is impossible to answer. One needs to know the function's role in an overall program — the performance requirements, how often the function is called, etc.

It would be vastly simpler for the integration team to simply filter out extracted tuples that contain movie titles (for which there are free and high-quality downloadable databases). However, this easy fix will not be apparent in the siloed system; it is only apparent to an engineer responsible for integrated end-to-end data quality.

It may seem artificial to say that the siloed "extractor" cannot use dictionaries; we entirely agree. Using separate software components for each stage — written and maintained by separate teams — will inevitably lead to these artificial distinctions, which will lead to poor engineering decisions. In an integrated system, the developer simply asks whether the final data product is good enough; if the answer is no, the developer picks the most convenient place to fix the most important problem.

There may also be a further statistical advantage in incorporating multiple sources of information during the joint inference task.

2.5 Debuggable Decisions

Our final design goal is that DeepDive should enable *debuggable decisions.* When the developer has identified an extraction error, she must be able to examine what incorrect decision DeepDive made, and diagnose why it happened. Satisfying this goal has three practical impacts on the system:

- DeepDive uses human-understandable features that are implemented in traditional Python code, not the opaque features yielded by deep learning systems. In Section 5 we describe how to use this approach and still obtain some of the feature induction advantages associated with deep learning.

- All weighted decisions are described using calibrated probabilities, not scores. This allows humans to examine individual and intermediate decisions, then decide whether the intermediate decision was correct or not. Doing so is impossible unless the numerical value is human-comprehensible. This places additional requirements on our probabilistic inference mechanism.

- It retains a statistical "execution history" and can present it to the user in an easy-to-consume form. For example, our debugging tool always presents, for each feature, the number of times the feature was observed in the training data. This allows engineers to detect whether the feature has an incorrect weight due to insufficient training data.

In the following section we describe the overall DeepDive architecture, and discuss how it meets these three design criteria.

3. SYSTEM OVERVIEW

Figure 2 shows DeepDive's three execution phases:

1. In *candidate generation*, DeepDive applies a user-defined function (UDF) to each document in the input corpus to yield candidate extractions. For example, the book price extractor might emit every numerical value from each input webpage. The goal of the candidate generator is to eliminate "obviously" wrong outputs (such as non-numeric prices). The candidate generation step is thus intended to be *high-recall, low-precision.*

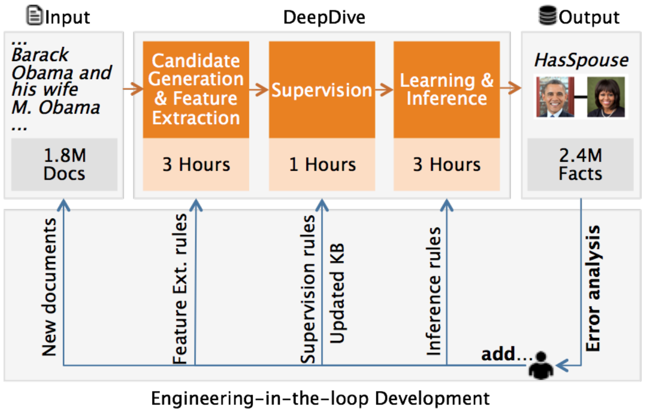


Figure 2: DeepDive takes as input dark data — such as unstructured documents — and outputs a structured database. The runtimes here are for the TAC-KBP competition system. To improve quality, the developer adds new rules and new data with error analysis conducted on the result of the current snapshot of the system. DeepDive provides a declarative language to specify each type of different rules and data, and techniques to incrementally execute this iterative process.

The system also applies user-defined feature functions to each candidate. For example, one feature might indicate whether the candidate has a **$** symbol to its left. We believe that improving feature quality is one of the core mechanisms by which a statistical system can improve its quality [2]. Unlike some statistical frameworks in which features are largely or entirely synthetic — say, deep learning — we strongly believe that all features should be comprehensible by humans, so as to permit downstream debugging.

2. In *supervision*, DeepDive applies user-defined *distant supervision rules* to provide labels for some of the candidates. For example, we might know the true price for a subset of downloaded Web pages because of a previous hand-annotated database. Supervision rules do not have to label each candidate: they are intended to be *low-recall, high-precision*. Distant supervision rules in some cases may be more error-prone than direct human label generation, but the former allows us to fix and reexecute rules, and also to generate massively more labeled examples.

3. In *learning and inference*, we construct a graphical model that represents all of the labeled candidate extractions, train weights, then infer a correct probability for each candidate. At the end of this stage, we apply a threshold to each inferred probability and thereby obtain the extractions that will be placed in the output database.

Every time the developer makes a pass through the Figure 1 iteration loop — that is, each time she changes a candidate generation UDF, a feature UDF, or the distant supervision rules — we rerun DeepDive to obtain a new output table. The user can also specify many smaller-bore task-specific details: where to find input data, the schema of the output, how to incorporate information about multiple possible extractions, and so on. The user provides this information using **DDlog**.

3.1 Candidate Generation and Feature Extraction

All data in DeepDive is stored in a relational database. The first phase populates the database using a set of SQL queries and user-defined functions (UDFs). By default, DeepDive stores all documents in the database in one sentence per row with markup produced by standard NLP pre-processing tools, including HTML stripping, part-of-speech tagging, and linguistic parsing. After this loading step, DeepDive executes two types of queries: (1) *candidate mappings*, which are SQL queries that produce possible mentions, entities, and relations, and (2) *feature extractors* that associate features to candidates.

Example 3.1. *Candidate mappings are usually simple. Here, we create a relation mention for every pair of candidate persons in the same sentence (s):*

$$\text{(R1)}\ \texttt{MarriedCandidate}(m1, m2) :\text{-}\ \texttt{PersonCandidate}(s, m1), \texttt{PersonCandidate}(s, m2).$$

Candidate mappings are simply SQL queries with UDFs that look like low-precision but high-recall ETL scripts. Such rules must be high recall: if the union of candidate mappings misses a fact, DeepDive will never extract it.

We also need to extract features. Here we extend classical Markov Logic in two ways: (1) user-defined functions and (2) weight tying, which we illustrate by example.

Example 3.2. *Suppose that UDF* $\mathsf{phrase}(m1, m2, sent)$ *returns the phrase between two mentions in the sentence, e.g., "and his wife" in the above example. The phrase between two mentions may indicate whether two people are married. We would write this as:*

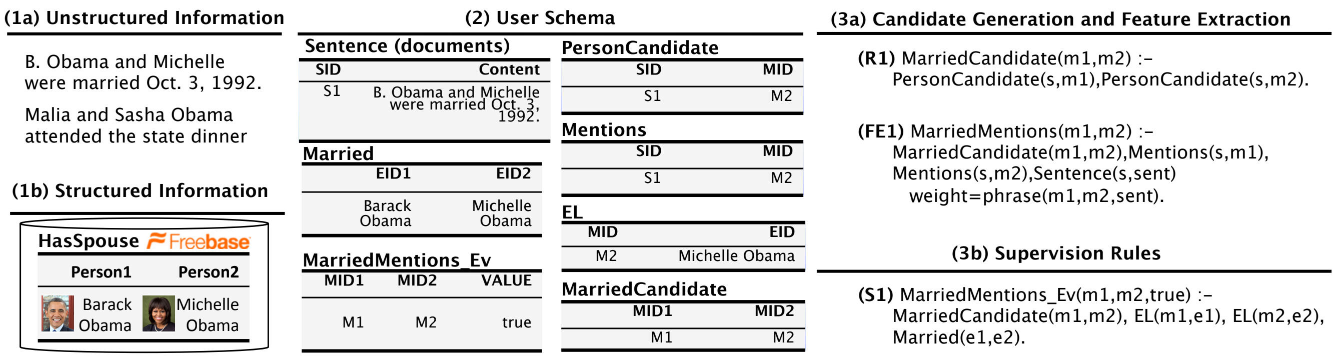
$$\begin{aligned}\text{(FE1)}\ &\texttt{MarriedMentions}(m1, m2) :\text{-}\\ &\texttt{MarriedCandidate}(m1, m2), \texttt{Mention}(s, m1),\\ &\texttt{Mention}(s, m2), \texttt{Sentence}(s, sent)\\ &weight = \mathsf{phrase}(m1, m2, sent).\end{aligned}$$

One can think about this like a classifier: This rule says that whether the text indicates that the mentions m1 and m2 are married is influenced *by the phrase between those mention pairs. The system will infer based on training data its confidence (by estimating the weight) that two mentions are indeed indicated to be married.*

Technically, phrase returns an identifier that determines which weights should be used for a given relation mention in a sentence. If phrase returns the same result for two relation mentions, they receive the *same* weight. In general, phrase could be an arbitrary UDF that operates in a per-tuple fashion. This allows DeepDive to support common examples of features such as "bag-of-words" to context-aware NLP features to highly domain-specific dictionaries and ontologies. In addition to specifying sets of classifiers, DeepDive inherits Markov Logic's ability to specify rich correlations between entities via weighted rules. Such rules are particularly helpful for data cleaning and data integration.

3.2 Supervision

Just as in Markov Logic, DeepDive can use training data or evidence about any relation; in particular, each user relation is associated with an evidence relation with the same



(1a) Unstructured Information

B. Obama and Michelle were married Oct. 3, 1992.

Malia and Sasha Obama attended the state dinner

(1b) Structured Information

HasSpouse Freebase

Person1	Person2
Barack Obama	Michelle Obama

(2) User Schema

Sentence (documents)

SID	Content
S1	B. Obama and Michelle were married Oct. 3, 1992.

Married

EID1	EID2
Barack Obama	Michelle Obama

MarriedMentions_Ev

MID1	MID2	VALUE
M1	M2	true

PersonCandidate

SID	MID
S1	M2

Mentions

SID	MID
S1	M2

EL

MID	EID
M2	Michelle Obama

MarriedCandidate

MID1	MID2
M1	M2

(3a) Candidate Generation and Feature Extraction

(R1) MarriedCandidate(m1,m2) :-
PersonCandidate(s,m1),PersonCandidate(s,m2).

(FE1) MarriedMentions(m1,m2) :-
MarriedCandidate(m1,m2),Mentions(s,m1),
Mentions(s,m2),Sentence(s,sent)
weight=phrase(m1,m2,sent).

(3b) Supervision Rules

(S1) MarriedMentions_Ev(m1,m2,true) :-
MarriedCandidate(m1,m2), EL(m1,e1), EL(m2,e2),
Married(e1,e2).

Figure 3: An example DeepDive deployment.

schema and an additional field that indicates whether the entry is true or false. Continuing our example, the evidence relation $MarriedMentions\_Ev$ could contain mention pairs with positive and negative labels. As a rule, we use distant supervision to obtain labels rather than manual efforts.

Example 3.3. *Distant supervision [20, 33] is a popular technique to create evidence in information extraction systems. The idea is to use an incomplete database of married entity pairs to* heuristically *label (as* `True` *evidence) all relation mentions that link to a pair of married entities:*

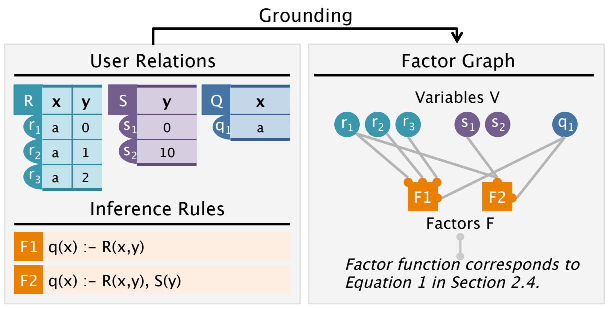
(S1) $\texttt{MarriedMentions\_Ev}(m1, m2, true) \text{ :-}$
$\texttt{MarriedCandidates}(m1, m2), \texttt{EL}(m1, e1),$
$\texttt{EL}(m2, e2), \texttt{Married}(e1, e2).$

Here, `Married` is an (incomplete) list of married real-world persons that we wish to extend. The relation EL is for "entity linking" that maps mentions to their candidate entities. At first blush, this rule seems incorrect. However, it generates noisy, imperfect examples of sentences that indicate two people are married. Machine learning techniques are able to exploit redundancy to cope with the noise and learn the relevant phrases (e.g., "and his wife"). Negative examples are generated by relations that are largely disjoint (e.g., siblings). Similar to DIPRE [6] and Hearst patterns [19], distant supervision exploits the "duality" [6] between patterns and relation instances; furthermore, it allows us to integrate this idea into DeepDive's unified probabilistic framework.

3.3 Learning and Inference

In the learning and inference phase, DeepDive generates a factor graph, similar to Markov Logic, and uses techniques from Tuffy [36]. The inference and learning are done using standard techniques (Gibbs Sampling) that we describe below after introducing the formal semantics.

As in Figure 4, DeepDive explicitly constructs a factor graph for inference and learning using a set of SQL queries. Recall that a factor graph is a triple $(V, F, \hat{w})$ in which V is a set of nodes that correspond to Boolean random variables, F is a set of hyperedges (for $f \in F$, $f \subseteq V$), and $\hat{w} : F \times \{0,1\}^V \to \mathbb{R}$ is a weight function. In DeepDive, each variable corresponds to one tuple in the database, and each hyperedge f corresponds to the set of groundings for a rule γ. In DeepDive, V and F are explicitly created using a set of SQL queries. These data structures are then passed to the sampler, which runs outside the database, to estimate the marginal probability of each node or tuple in the database. (In principle there could be some data movement advantages to running the sampler inside the core database system, but probably not enough to warrant the additional code complexity.) Each tuple is then reloaded into the database with its marginal probability.



R	x	y
r1	a	0
r2	a	1
r3	a	2

S	y
s1	0
s2	10

Q	x
q1	a

F1 q(x) :- R(x,y)

F2 q(x) :- R(x,y), S(y)

Figure 4: Schematic illustration of grounding. Each tuple corresponds to a Boolean random variable and node in the factor graph. We create one factor for every set of groundings.

Example 3.4. *Take the database instances and rules in Figure 4 as an example, each tuple in relation R, S, and Q is a random variable, and V contains all random variables. The inference rules $F1$ and $F2$ ground factors with the same name in the factor graph as illustrated in Figure 4. Both $F1$ and $F2$ are implemented as SQL in* DeepDive.

To define the semantics, we first define the concept of *possible world*: a possible world is a function that assigns each variable a truth value. Let $\mathcal{I}$ be the set of all possible worlds, for each factor f and a possible world I, we use $\hat{w}(f, I)$ as the value returned by factor f for possible world I. We then define $\hat{W}(F, I) = \sum_{f \in F} \hat{w}(f, I)$, and then the probability of a possible world is the following function:

$$\Pr[I] = Z^{-1} \exp\left\{\hat{W}(F, I)\right\} \text{ where } Z = \sum_{I \in J} \exp\left\{\hat{W}(F, I)\right\}$$

The main task that DeepDive conducts on factor graphs is statistical inference, i.e., for a given node, what is the marginal probability that this node takes the value 1? This marginal probability is defined as follows. Given a variable v, let $\mathcal{I}_+$ be the set of all possible worlds in which v is assigned to true, the marginal probability of v is $\sum_{I \in \mathcal{I}_+} \Pr[I]$.

3.4 Execution

DeepDive runs the above three phases in sequence, and at the end of the learning and inference, it obtains a marginal probability p for each candidate fact. To produce the output database, DeepDive applies a user-chosen threshold, e.g., $p > 0.95$. For some applications that favor extremely high recall at the expense of precision, it may be appropriate to lower this threshold.

Typically, the developer will need to perform an *error analysis* and repeat the development cycle. Error analysis is the process of understanding the most common mistakes (incorrect extractions, too-specific features, candidate mistakes, etc.) and deciding how to correct them [40]. To facilitate error analysis, users write standard SQL queries or use the Mindtagger tool [45]. We will describe these methods in more detail in Section 5.

4. SYSTEM INFRASTRUCTURE

DeepDive relies on two novel technological components for *incremental grounding* as well as *statistical inference and learning.*

4.1 Incremental Grounding

Grounding takes place when DeepDive translates the set of relations and rules into a concrete factor graph upon which probabilistic inference is possible. Because DeepDive is based on SQL, we are able to take advantage of decades of work on incremental view maintenance. The input to this phase is the same as the input to the grounding phase, a set of SQL queries and the user schema. The output of this phase is how the output of grounding changes, i.e., a set of modified variables ΔV and their factors ΔF. Since V and F are simply views over the database, any view maintenance techniques can be applied to incremental grounding. DeepDive uses the DRed algorithm [17] that handles both additions and deletions. Recall that in DRed, for each relation R_i in the user's schema, we create a *delta relation*, R_i^δ, with the same schema as R_i and an additional column *count*. For each tuple t, *t.count* represents the number of derivations of t in R_i. On an update, DeepDive updates delta relations in two steps. First, for tuples in R_i^δ, DeepDive directly updates the corresponding counts. Second, a SQL query called a *"delta rule"*[2] is executed which processes these counts to generate modified variables ΔV and factors ΔF. We found that the overhead of DRed is modest and the gains may be substantial, so DeepDive always runs DRed–except on initial load.

4.2 Statistical Inference and Learning

The main task that DeepDive conducts on factor graphs is statistical inference, i.e., for a given node, what is the marginal probability that this node takes the value 1? Since a node takes value 1 when a tuple is in the output, this process computes the marginal probability values returned to users. In general, computing these marginal probabilities is ♯P-hard [50]. Like many other systems, DeepDive uses Gibbs sampling [43] to estimate the marginal probability of every tuple in the database.

[2]For example, for the grounding procedure illustrated in Figure 4, the delta rule for $F1$ is $q^\delta(x) : -R^\delta(x,y)$.

Efficiency and Scalability. There are two components to scaling statistical algorithms: *statistical efficiency*, roughly how many steps an algorithm takes to converge, and *hardware efficiency*, how efficient each of those step is. We introduced this terminology and studied this extensively in a recent paper [55].

DimmWitted [55], the statistical inference and learning engine in DeepDive, is built upon our research of how to design a high-performance statistical inference and learning engine on a single machine [29, 41, 54, 55]. DimmWitted models Gibbs sampling as a "column-to-row access" operation: each row corresponds to one factor, each column to one variable, and the non-zero elements in the matrix correspond to edges in the factor graph. To process one variable, DimmWitted fetches one column of the matrix to get the set of factors, and other columns to get the set of variables that connect to the same factor. In standard benchmarks, DimmWitted was 3.7× faster than GraphLab's implementation without any application-specific optimization. Compared with traditional work, the main novelty of DimmWitted is that it considers *both* hardware efficiency and statistical efficiency for executing an inference and learning task.

- **Hardware Efficiency** DeepDive takes into consideration the architecture of modern Non-uniform memory access (NUMA) machines. A NUMA machine usually contains multiple nodes (sockets); each socket contains multiple CPU cores. To obtain high hardware efficiency, one wants to decrease the communication across different NUMA nodes.
- **Statistical Efficiency** Pushing hardware efficiency to the extreme might suffer on statistical efficiency because the lack of communication between nodes might decrease the rate of convergence of a statistical inference and learning algorithm. DeepDive takes advantage of the theoretical results of model averaging [57] and our own results about lock-free execution [29, 41].

When performing extraction from a corpus of 0.3 million papers from the paleobiology literature, the factor graph contains more than 0.2 billion random variables and 0.3 billion factors. On this factor graph, DeepDive is able to run Gibbs sampling on a machine with 4 sockets (10 core per sockets), and we find that we can generate 1,000 samples for all 0.2 billion random variables in 28 minutes. This is more than 4× faster than a non-NUMA-aware implementation.

Incremental Inference. Due to our choice of incremental grounding, the input to DeepDive's inference phase is a factor graph along with a set of changed variables and factors. The goal is to compute the output probabilities computed by the system. Our approach is to frame the incremental maintenance problem as approximate inference. Previous work in the database community has looked at how machine learning data products change in response to both to new labels [25] and to new data [9, 10]. When extracting databases from dark data, both the program and data change on each iteration. Our system can cope with both types of change simultaneously.

There are two popular classes of approximate inference techniques: *sampling-based materialization* (inspired by sampling-based probabilistic databases such as MCDB [22]) and *variational-based materialization* (inspired by techniques for approximating graphical models [49]). We conducted an experimental evaluation of these two approaches on a diverse set of DeepDive programs. We found these two approaches are sensitive to changes in the size of the factor graph, the sparsity of correlations, and the anticipated number of future changes. The performance varies by up to two orders of magnitude in different points of the space. To automatically choose the materialization strategy, we use a simple rule-based optimizer.

5. DEVELOPING AND DEBUGGING

The main claim for DeepDive is that it enables data quality that is much higher than previous or competing systems. But so far we have covered mainly how to match our three design criteria of mechanism independence, integrated processing, and debuggable decisions. DeepDive meets these criteria by framing extraction as a single integrated probabilistic inference task, along with infrastructure that can perform fast and accurate probabilistic inference without any one-off hacking of the inference procedure, plus some code that generates log-like data about DeepDive's decisions.

In this section we describe how these advantages translate into DeepDive's reliably high-quality output.

5.1 Improvement Iteration Loop

DeepDive reaches high accuracy by enabling an efficient *improvement iteration loop*. The basic steps in this loop are described in Figure 1. As mentioned in Section 2, this developer loop is inspired by engineering behavior in traditional performance debugging settings: the engineer repeatedly runs the system, carefully diagnoses problems, and makes minimal modifications to address only the diagnosed problems. Performing this cycle is not easy — new students require lots of training in order to do it — but can be done reliably and repeatedly. Put another way, a good software engineering team can reliably obtain better performance by applying systematic effort to the problem.

In contrast, most machine learning-driven efforts in our experience do not exhibit either reliable accuracy improvement over time or systematic practice by engineers. Instead, the accuracy improvement over time is often fitful and unreliable. The engineering practice is often characterized by tinkering with the core statistical machinery and opaque "rules of thumb" that often lead to wasted or low-productivity effort.

We have devoted substantial time to evolving a systematic engineering practice that yields reliable improvements to extraction quality over time. Below we describe how the engineer identifies and remedies accuracy errors, plus some dead ends and common failure modes we have observed while building DeepDive.

5.2 Fixing Failure

Rapidly identifying and repairing accuracy errors is the core mechanism by which DeepDive obtains high accuracy. The first step in this process is when an engineer produces an *error analysis*. This is a strongly stylized document that helps the engineer determine:

- The true precision and recall of the extractor

- An enumeration of observed extractor failure modes, along with error counts for each failure mode. For example, a flawed extractor for medical doctors might incorrectly yield city names because they appear after the term *Dr.*

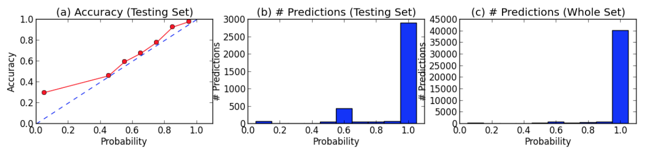


Figure 5: Example plots emitted by DeepDive after training. From left-to-right: a probability calibration plot, a histogram of the different emitted probabilities for the test set, and a histogram of different emitted probabilities for the training set.

- For the top-ranked failure modes, the underlying reason DeepDive made a mistake.

The error analysis document plays a role similar to the software engineer's performance instrumentation tool. Unfortunately, unlike most performance tools, producing the error analysis cannot be wholly automated. It requires a human being to:

1. Manually mark a sample of emitted extractions — usually roughly 100 — as correct or incorrect. This sample will be used to compute DeepDive's *precision.*

2. Manually mark a sample of correct answers from the documents — usually roughly 100 — as correctly or incorrectly-extracted. This sample will be used to compute DeepDive's *recall.*

3. Place each extraction failure into one of several "failure mode buckets." These buckets are not hard-coded in advance, and are not strictly defined; they are more akin to semantic tags applied by the engineer to errors that seem similar. Examples include *bad doctor name from addresses*, or *bad gene name from capitalized title*, or *missed price extraction when preceded by Euro symbol.*

Producing the error analysis requires human annotation and can be time-consuming. (Though note that the person producing the above document can be someone other than the core DeepDive engineer, and in the case of highly-technical input corpora, the primary DeepDive engineer might not be qualified to produce it.)

The error analysis document also includes information the user does not have to generate manually: commodity statistics that describe the inputs and outputs, checksums of all data products and code, and summaries of features, including their learned weights and observed counts. It also has a version checksum and references to GitHub versions that contain the relevant feature functions and distant supervision code.

After producing the error analysis, the engineer sorts the failure mode buckets in descending order of frequency. She always tries to address the largest bucket first. Bugs usually fall into one of three categories:

1. The *candidate generator* failed to identify the extraction, or through a preprocessing error emitted a nonsense candidate (perhaps due to a bad character in the input, or an OCR failure in the case of hand-written forms) . This is easily checked by testing whether the correct answer was contained in the set of candidates evaluated probabilistically.

2. There is not a *feature function* that can successfully distinguish the right answer. The engineer checks for this

failure by examining each incorrect example in the current bucket, and asking whether the observed features would be sufficient for a human being to distinguish a correct from incorrect answer.

For example, if the doctor extractor has just a single feature — *is 'Dr' to the left of the candidate?* — then clearly there is not sufficient feature information for DeepDive to emit the correct result, and the engineer should add a new feature function.

3. If there is sufficient feature information, but DeepDive still yielded an incorrect result, then the learned feature weights are likely incorrect, most often due to insufficient *distant supervision rules.* This can easily happen when the distant supervision rules provide few or no labels for certain feature values.

   For example, if the extractor has two features — *is 'Dr' to the left of the candidate?* and *is the candidate inside an address?* — then the distant supervision rules must provide labels for candidates that exhibit both features. If the distant supervision rules instead just happen to label non-address examples, then DeepDive will not have enough evidence to correctly compute the relative weights of these features. The engineer should add a new distant supervision rule to cover this case.

That is all. The DeepDive engineer repeatedly creates the error analysis, then diagnoses the reason behind the most important errors. Do this enough times, and DeepDive can rapidly obtain extremely high data quality.

In addition, after each training run, DeepDive emits the diagrams shown in Figure 5. This data helps the engineer understand the system's probabilistic output without having to understand the statistical algorithm. The leftmost diagram is a *calibration plot* that shows whether DeepDive's emitted probabilities are accurate; *e.g.*, for all of the items assessed a 20% probability, are 20% of them actually correct extractions? In the case shown here, the actual red line does not reflect the ideal dotted blue line, suggesting DeepDive lacks sufficient feature evidence to compute entirely correct probabilities. The center and right diagrams show a histogram of predictions in various probability buckets for the test and training sets, respectively. Ideal prediction histograms are U-shaped, with the vast majority of items receiving a probability of either 0% or close to 100%. The center (test set) histogram here is worrisome, suggesting that for many test cases, DeepDive does not have enough evidence to push its belief either to zero or 1.

5.3 Design Choices

Making the engineer as productive as possible has been a driving force in the last few years of DeepDive development. We have made a number of nonobvious design choices and pulled back from a few interesting dead ends.

Human-understandable features only — All of the features that DeepDive uses are easily human-understandable. This decision stands in sharp contrast to recent deep learning trends that deemphasize manual feature engineering in favor of manually-designed networks and training procedures that can yield effective but opaque features.

Note that human comprehensibility is core to the debugging loop described above. In our view, the human engineer must be able to determine whether an observed mistake was due to a lack of effective features or due to some other source of failure; that determination is impossible if the feature set is not comprehensible.

In the past year we have introduced a *feature library* system that automatically proposes a massive number of features that plausibly work across many domains, and then uses statistical regularization to throw away all but the most effective features. This method gives a bit of the feel of deep learning, in that some features come "for free" with no explicit engineer involvement. However, the hypothesized features are designed to always be human-understandable; we describe the space of all possible features using code-like "feature templates" instead of the opaque network designs common in deep learning.

Distant supervision — Our standard development loop never calls upon anyone to manually label training data examples. The only explicit human labels come during the error analysis phase, after the system has already been trained. This decision stands in contrast to most previous supervised trained systems, which rely on huge numbers of manually-labeled examples.

We use distant supervision for three reasons. First, it is far less burdensome than manual labeling and thus enables rapid execution of the improvement iteration cycle. Second, we have found in other work that the massive number of labels enabled by distant supervision rules may simply be more effective than the smaller number of labels that come from manual processes, even in the face of possibly-higher error rates [53]. Third, distant supervision rules can be revised, debugged, and cheaply reexecuted; in contrast, a flaw in the human labeling process can only be fixed by expensively redoing all of the work.

Few deterministic rules — This point is best described as an engineering failure mode we now actively strive to avoid. When faced with an extraction task, it is often possible to rapidly obtain middling data quality by writing a simple regular expression. For example, simply extracting capitalized words after *Dr.* would indeed give a healthy number of correct doctor extractions. This approach can be extremely appealing from an engineering perspective, as it is quick to write and relies entirely on well-understood software libraries instead of probabilistic machinery that may seem novel or surprising.

However, this approach is also a dead end for all but the most trivial extraction targets. A conscientious engineer, having met with success after writing the first regular expression, will quickly address the remaining quality bugs by writing a second. In our experience, this second deterministic rule will indeed address some bugs, but will be vastly less productive than the first one. The third regular expression will be even less productive. Very quickly, the engineer will have written a depressing page full of increasingly-obscure regular expressions that are hard to debug, hard to maintain, and still do not obtain human-level quality.

We have seen this pattern in new DeepDive engineers many, many times. It has led to failure every single time but two: when extracting phone numbers and email addresses. There is a real cognitive resistance to adopting the development cycle described above; it appears to run strongly against common engineering habit, perhaps similar to the way habits for effective performance debugging can be very counterintuitive. Training new engineers in the DeepDive debugging methodology can be initially difficult, but once these engineers are trained, they are enormously effective. Indeed, over the last several years, improvements in tools, practices, and training have enabled experienced DeepDive engineers to produce many novel and high-quality databases in 1-2 days, down from several months originally.

6. APPLICATIONS

We now explore several applications in detail, to give a flavor for the kinds of applications DeepDive can enable.

6.1 Medical Genetics

The body of literature in life sciences has been growing rapidly for years, to the extent that it has become unrealistic for scientists to perform research solely based on reading and memorization (even with the help of keyword search). As a result, there have been numerous initiatives to build structured databases from the research literature. For example, OMIM is an authoritative database of human genes and genetic disorders. It dates back to the 1960s, and so far contains about 6,000 hereditary diseases or phenotypes. Because OMIM is curated by humans, it has been growing at a rate of roughly 50 records / month for many years. In collaboration with Prof. Gill Bejerano at Stanford, we are developing DeepDive applications in the field of medical genetics. Specifically, we use DeepDive to extract mentions of genes, diseases, and phenotypes from the literature, and statistically infer their relationships.

Consider a very simple schema of *regulate* relationships that consists of (**gene**, **phenotype**, **research-paper**). This database can be crucial for treating genetic illnesses: consider a patient who arrives at the doctor with a symptom that the doctor believes is likely to be genetically-linked. The doctor takes two steps to find a possible clinical intervention: (1) she runs a test on the patient to determine which genes are atypical, (2) she checks the scientific literature to see if there is any known linkage between the patient's symptom and the atypical genes. This second step — sometimes known informally as "asking Doctor Google" — can be hugely time-consuming: the scientific literature is vast, genes and phenotypes are often known by different names, and identifying a relevant relationship between a gene and phenotype often requires reading and understanding the paper. Doctors can spend large amounts of time researching rare genetic illnesses, and if Googling does not yield immediate results, it can be difficult for the doctor to distinguish between humanity's lack of clinical knowledge on a topic vs. an especially challenging Internet browsing effort. As a result, doctors spend a large amount of time performing research instead of treating patients, and in some cases may fail to identify literature that is crucial for formulating treatment.

If the doctor had access to a high-quality version of the (**gene**, **phenotype**, **research-paper**) database, it would be simple and quick to look up patient information. DeepDive enables the construction of this database.

6.2 Pharmacogenomics

Understanding the interactions of chemicals in the body is key for drug discovery. However, the majority of this data resides in the biomedical literature and cannot be easily accessed. The Pharmacogenomics Knowledgebase is a high quality database that aims to annotate the relationships between drugs, genes, diseases, genetic variation, and pathways in the literature. With the exponential growth of the literature, manual curation requires prioritization of specific drugs or genes in order to stay up to date with current research. In collaboration with Emily Mallory and Prof. Russ Altman [30] at Stanford, we are developing DeepDive applications in the field of pharmacogenomics. Specifically, we use DeepDive to extract relations between genes, diseases, and drugs in order to predict novel pharmacological relationships.

6.3 Materials Science

The world of semiconductors offers a scenario similar to that of medical genetics. There is a large universe of chemical formulas from which it is possible to create semiconductors with different technical physical properties, such as *electron field mobility*. It would be useful for a semiconductor engineer to have a handbook of semiconductor materials and their properties, using which she could simply look up a desirable set of numbers for any specific application. Unfortunately, there is no such handbook — the evolving research literature itself is the best source of information on semiconductors data. As a result, engineers can spend large amounts of time rifling through the scientific literature in the search for just the right material.

In conjunction with researchers at Toshiba Corporation, we have worked since the fall of 2014 to create this database of semiconductors and their properties.

6.4 Fighting Human Trafficking

Human trafficking is an odious crime that uses physical, economic, or other means of coercion to obtain labor from human beings. Trafficked individuals are often used in sex work or factory work. Unfortunately, identifying human trafficking victims in the United States can often be difficult for authorities — the individuals may be kept from regular contact with friends or family, and may be unable, afraid, or too ashamed to contact law enforcement by themselves. (Trafficking patterns differ from country to country.) It would be far better if law enforcement could proactively identify and investigate possible trafficking victims.

Somewhat surprisingly, there is data online that may contain evidence of trafficking activity. Like many other forms of commerce, sex work advertising is now online; there are many websites where providers of sex services post ads containing price, location, contact information, physical characteristics, and other data. Details differ across sites, but in most cases the posts resemble Craigslist-style textual classified ads: they have very little structure, lots of extremely nonstandard English, and tend to contain the same data values. It is easy to imagine a relational schema that would capture most of the relevant data in these ads.

There is additional information available on discussion forum websites where users of sex services post information about their experiences. Again, these posts are primarily textual, but tend to contain similar information: when the encounter took place, what happened, the sex worker's details, etc. For a surprisingly large percentage of forum posts, there is enough contact information for us to identify the relevant sex worker ad. The posts are also remarkably candid, describing both the illegal activity itself as well as details about the sex worker (*e.g.*, evidence of drug abuse or physical abuse) that would never have been found in the original advertisement.

It was initially surprising to us that forum writers are willing to be so forthcoming. There appears to be a strong social element to the forum posts, roughly similar to what can be observed in Amazon and Yelp discussion fora; domain experts in sex commerce have reported to us that this finding is not surprising to them.

We ran DeepDive on approximately 45M advertisements and 0.5M forum posts, creating two distinct structured tables. We can use this data in many different ways:

- Statistics from advertisements can be used by themselves to identify trafficking warning signs. For example, a sex worker who posts from multiple cities in relatively rapid succession may be moved from place to place by traffickers in order to be kept disoriented and easy to manage. A worker who posts an unusually low price or is willing to engage in extremely risky practices may also be doing so as a trafficking victim. We have shipped extracted data for other companies to use in law enforcement products; this data has been deployed to several law enforcement agencies and has been used in at least one arrest.
- Using price data from the advertisements alone, we can compute aggregate statistics and analyses about sex commerce in the United States. We are writing a paper with economists and political scientists about the relationship between pricing in the sex economy in individual cities and the licit economies in those places.
- Using the joined advertisement and forum tables, we can use forum posters' comments about experiences to identify sex workers who may be in danger. For example, signs of drug or physical abuse on their own are worrisome on their own, and may also be evidence of possible trafficking. Engineers at Lattice Data (`http://lattice.io`), a startup company designed to commercialize DeepDive, have used this data to create a user-facing faceted search tool for law enforcement officers.

This work was done as part of the DARPA MEMEX project and has been covered on 60 Minutes and other news sources.

7. RELATED WORK

Knowledge Base Construction (KBC). KBC has been an area of intense study over the last decade, moving from pattern matching [19] and rule-based systems [28] to systems that use machine learning for KBC [5, 8, 13, 14, 34]. Many groups have studied how to improve the quality of specific components of KBC systems [33, 51]. DeepDive's effort to extract structured databases from dark data can be seen as an effort to ease and accelerate the KBC process. DeepDive has many common features to Chen and Wang [11], Google's Knowledge Vault [13], and a forerunner of DeepDive, Tuffy [36]. We focus on the incremental evaluation from feature extraction to inference.

DeepDive's model of KBC is motivated by the recent attempts of using machine learning-based technique for KBC [3, 4, 24, 38, 46, 52, 56] and the line of research that aims to improve the quality of a specific component of KBC system [7, 12, 15, 21, 26, 27, 31–33, 35, 39, 42, 47, 48, 51, 53, 54]. When designing DeepDive, we used these systems as test cases to justify the generality of our framework. In fact, we find that DeepDive is able to model thirteen of these popular KBC systems [15,23,26,27,31–33,35,39,42,51,53,54].

Declarative Information Extraction. The database community has proposed declarative languages for information extraction, a task with similar goals to knowledge base construction, by extending relational operations [16, 28, 44], or rule-based approaches [34]. These approaches can take advantage of classic view maintenance techniques to make the execution incremental, but they do not study how to incrementally maintain the result of statistical inference and learning, which is the focus of our work.

Data Cleaning and Integration. Data cleaning and integration systems have been active research topics in academia, with moderate deployment in industrial settings. Many deployed systems to date have focused on generating precise and correct integrations for traditional transaction-focused systems, such as merged human resources databases.

A newer set of software systems, such as Wrangler and Tamr, have focused on data ingest, cleaning, and integration for analytical purposes. To the best of our knowledge, these systems are focused more on ingesting diverse sources of structured data rather than free-form dark data, but the line between the two can be hazy (*e.g.*, template-driven electronic medical records with a mixture of structured and unstructured components). More importantly, these systems are rule-driven rather than probabilistic, which we believe will lead to the higher maintenance costs and lower data quality we described in Section 5.3. We have found that probabilistic inference and an effective accompanying engineering cycle are core to DeepDive's success in obtaining reliably high quality.

8. CONCLUSIONS AND FUTURE WORK

We have demonstrated DeepDive's ability to obtain high-quality databases in many different domains. We believe there is a vast amount of future work in applying DeepDive to novel problems in the sciences, medicine, finance, and other real-world domains with huge amounts of dark data. We have been repeatedly surprised by the sheer number and variety of problems that can be addressed using DeepDive's results.

Much of the future technical work on DeepDive entails reducing the amount of developer effort needed to obtain high quality. This work can be reduced using novel developer tools, novel data-centric methods, or some combination.

For example, over time we have modified DeepDive to require *less* information from the user, not more. Users have fairly limited direct control over the machine learning elements of our knowledge base construction procedure, especially compared to what is possible given most standard machine learning toolkits. However, it is still possible for users to make inadvertent and hard-to-debug errors. For example, consider a user who has provided the above-listed feature, which we might call $isGene_{f1}$. This is an entirely reasonable piece of feature code, and is likely just one of several features the system uses when classifying an in-document string as **is-gene** or not.

However, now consider what happens when the user has also provided a distant supervision rule. Distant supervision rules are algorithmic methods for generating labeled data; they can be less accurate than traditional hand-labeled data examples, but when applied to a large corpus they can generate far more labeled data than is possible with hand-marking.

Unfortunately, if the distant supervision rule is identical to or extremely similar to a feature function, standard statistical training procedures will fail badly. Even if the user has provided a large number of helpful features, there is one that *perfectly* predicts the labeled answer, and so the training procedure will build a model that places all weight on the single feature that overlaps with the supervision rule. The trained statistical model will — reasonably enough — have little effectiveness in the real world, in which the feature and the truth do not perfectly overlap. This failure mode is extremely hard to detect: to the user, it simply appears that the training procedure has failed. Worse, it has failed after the user has been conscientious enough to provide what should have been a high-quality feature function. Continuing to address such DeepDive engineering failure modes is an ongoing project.

9. 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, DARPA's MEMEX program and SIMPLEX program, the National Science Foundation (NSF) CAREER Awards under No. IIS-1353606 and No. IIS-1054913, 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, Dow, Google, Yahoo!, 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.

10. REFERENCES

[1] *MUC6 '95: Proceedings of the 6th Conference on Message Understanding*, Stroudsburg, PA, USA, 1995. Association for Computational Linguistics.

[2] M. R. 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, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*, 2013.

[3] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2670–2676, 2007.

[4] D. Barbosa, H. Wang, and C. Yu. Shallow information extraction for the knowledge web. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 1264–1267, 2013.

[5] J. Betteridge, A. Carlson, S. A. Hong, E. R. H. Jr., E. L. M. Law, T. M. Mitchell, and S. H. Wang. Toward never ending language learning. In *Learning by Reading and Learning to Read, Papers from the 2009 AAAI Spring Symposium, Technical Report SS-09-07, Stanford, California, USA, March 23-25, 2009*, pages 1–2, 2009.

[6] S. Brin. Extracting patterns and relations from the world wide web. In *The World Wide Web and Databases, International Workshop WebDB'98, Valencia, Spain, March 27-28, 1998, Selected Papers*, pages 172–183, 1998.

[7] R. C. Bunescu and R. J. Mooney. Learning to extract relations from the web using minimal supervision. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*, 2007.

[8] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, 2010.

[9] F. Chen, A. Doan, J. Yang, and R. Ramakrishnan. Efficient information extraction over evolving text data. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 943–952, 2008.

[10] F. Chen, X. Feng, C. Re, and M. Wang. Optimizing statistical information extraction programs over evolving text. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, pages 870–881, 2012.

[11] Y. Chen and D. Z. Wang. Knowledge expansion over probabilistic knowledge bases. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 649–660, 2014.

[12] M. Craven and J. Kumlien. Constructing biological knowledge bases by extracting information from text sources. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology, August 6-10, 1999, Heidelberg, Germany*, pages 77–86, 1999.

[13] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, K. Murphy, S. Sun, and W. Zhang. From data fusion to knowledge fusion. *PVLDB*, 7(10):881–892, 2014.

[14] O. Etzioni, M. J. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in knowitall: (preliminary results). In *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*, pages 100–110, 2004.

[15] M. Fan, D. Zhao, Q. Zhou, Z. Liu, T. F. Zheng, and E. Y. Chang. Distant supervision for relation extraction with matrix completion. In *EMNLP*, 2014.

[16] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The lixto data extraction project - back and forth between theory and practice. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 14-16, 2004, Paris, France*, pages 1–12, 2004.

[17] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993.*, pages 157–166, 1993.

[18] A. Y. Halevy, P. Norvig, and F. Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

[19] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *14th International Conference on Computational Linguistics, COLING 1992, Nantes, France, August 23-28, 1992*, pages 539–545, 1992.

[20] R. Hoffmann, C. Zhang, X. Ling, L. S. Zettlemoyer, and D. S. Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 541–550, 2011.

[21] R. Hoffmann, C. Zhang, and D. S. Weld. Learning 5000 relational extractors. In *ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, July 11-16, 2010, Uppsala, Sweden*, pages 286–295, 2010.

[22] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. MCDB: a monte carlo approach to managing uncertain data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 687–700, 2008.

[23] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA*, pages 655–665, 2014.

[24] G. Kasneci, M. Ramanath, F. M. Suchanek, and G. Weikum. The YAGO-NAGA approach to knowledge discovery. *SIGMOD Record*, 37(4):41–47, 2008.

[25] M. L. Koc and C. Ré. Incrementally maintaining classification using an RDBMS. *PVLDB*, 4(5):302–313, 2011.

[26] S. Krause, H. Li, H. Uszkoreit, and F. Xu. Large-scale learning of relation-extraction rules with distant supervision from the web. In *ISWC*, 2012.

[27] J. Li, A. Ritter, and E. H. Hovy. Weakly supervised user profile extraction from twitter. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 165–174, 2014.

[28] Y. Li, F. R. Reiss, and L. Chiticariu. SystemT: A declarative information extraction system. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 109–114, 2011.

[29] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 469–477, 2014.

[30] E. K. Mallory et al. Large-scale extraction of gene interactions from full text literature using deepdive. *Bioinformatics*, 2015.

[31] M. Marchetti-Bowick and N. Chambers. Learning for microblogs with distant supervision: Political forecasting with twitter. In *EACL*, 2012.

[32] B. Min, R. Grishman, L. Wan, C. Wang, and D. Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *NAACL-HLT*, 2013.

[33] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL 2009, Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore*, pages 1003–1011, 2009.

[34] N. Nakashole, M. Theobald, and G. Weikum. Scalable knowledge harvesting with high precision and high recall. In *Proceedings of the Forth International Conference on Web Search and Web Data Mining, WSDM 2011, Hong Kong, China, February 9-12, 2011*, pages 227–236, 2011.

[35] T.-V. T. Nguyen and A. Moschitti. End-to-end relation extraction using distant supervision from external semantic repositories. In *ACL*, 2011.

[36] F. Niu, C. Ré, A. Doan, and J. W. Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an RDBMS. *PVLDB*, 4(6):373–384, 2011.

[37] S. E. Peters et al. A machine reading system for assembling synthetic Paleontological databases. *PloS ONE*, 2014.

[38] H. Poon and P. M. Domingos. Joint inference in information extraction. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 913–918, 2007.

[39] M. Purver and S. Battersby. Experimenting with distant supervision for emotion classification. In *EACL 2012, 13th Conference of the European Chapter of the Association for Computational Linguistics, Avignon, France, April 23-27, 2012*, pages 482–491, 2012.

[40] 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.

[41] B. Recht, C. Re, S. J. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 693–701, 2011.

[42] S. Riedel, L. Yao, and A. McCallum. Modeling relations and their mentions without labeled text. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part III*, pages 148–163, 2010.

[43] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[44] W. Shen, A. Doan, J. F. Naughton, and R. Ramakrishnan. Declarative information extraction using datalog with embedded extraction predicates. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 1033–1044, 2007.

[45] J. Shin, C. Ré, and M. J. Cafarella. Mindtagger: A demonstration of data labeling in knowledge base construction. *PVLDB*, 8(12):1920–1931, 2015.

[46] F. M. Suchanek, M. Sozio, and G. Weikum. SOFIE: a self-organizing framework for information extraction. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 631–640, 2009.

[47] M. Surdeanu, S. Gupta, J. Bauer, D. McClosky, A. X. Chang, V. I. Spitkovsky, and C. D. Manning. Stanford's distantly-supervised slot-filling system. In *Proceedings of the Fourth Text Analysis Conference, TAC 2011, Gaithersburg, Maryland, USA, November 14-15, 2011*, 2011.

[48] M. Surdeanu, D. McClosky, J. Tibshirani, J. Bauer, A. X. Chang, V. I. Spitkovsky, and C. D. Manning. A simple distant supervision approach for the TAC-KBP slot filling task. In *Proceedings of the Third Text Analysis Conference, TAC 2010, Gaithersburg, Maryland, USA, November 15-16, 2010*, 2010.

[49] M. J. Wainwright and M. I. Jordan. Log-determinant relaxation for approximate inference in discrete markov random fields. *IEEE Transactions on Signal Processing*, 54(6-1):2099–2109, 2006.

[50] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[51] L. Yao, S. Riedel, and A. McCallum. Collective cross-document relation extraction without labelled data. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, EMNLP 2010, 9-11 October 2010, MIT Stata Center, Massachusetts, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1013–1023, 2010.

[52] A. Yates and O. Etzioni. Unsupervised resolution of objects and relations on the web. In *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, April 22-27, 2007, Rochester, New York, USA*, pages 121–130, 2007.

[53] C. Zhang, F. Niu, C. Ré, and J. W. Shavlik. Big data versus the crowd: Looking for relationships in all the right places. In *The 50th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, July 8-14, 2012, Jeju Island, Korea - Volume 1: Long Papers*, pages 825–834, 2012.

[54] C. Zhang and C. Ré. Towards high-throughput gibbs sampling at scale: a study across storage managers. In *Proceedings of the ACM SIGMOD International Conference*

on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013, pages 397–408, 2013.

[55] C. Zhang and C. Re. Dimmwitted: A study of main-memory statistical analytics. *PVLDB*, 7(12):1283–1294, 2014.

[56] J. Zhu, Z. Nie, X. Liu, B. Zhang, and J. Wen. StatSnowball: a statistical approach to extracting entity relationships. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 101–110, 2009.

[57] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems. Vancouver, Canada, December 6-9, 2010.*, pages 2595–2603, 2010.