Incremental Knowledge Base Construction Using DeepDive

Ofir Yair
Introduction

- **What is DeepDive?**
  System that combines database and machine learning ideas to help develop Knowledge Base Constructions (KBC) systems.

- **What is KBC system?**
  The process of populating a knowledge base (KB) with facts (or assertions) extracted from data (e.g., text, audio, video, tables, diagrams, ...).
The Goal

Unstructured information

High-quality structured Knowledge Base
How do we assess quality?

- **Precision** – how often a claimed tuple is correct
- **Recall** – how many of the possible tuples to extract were actually extracted
One may use DeepDive to build an application to extract spouse relations from sentences in the Web.

U.S President Barack Obama's wife Michelle Obama honored all mothers on Mother's Day and offered her thoughts...

Relation: has_spouse

<table>
<thead>
<tr>
<th>Subject</th>
<th>Object</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barack Obama</td>
<td>Michelle Obama</td>
<td>0.97</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Tuple in the has_spouse table representing the fact (for example), “Barak Obama” is married to “Michelle Obama”.
KBC Terminology

Mention Level Data (sentence)– The textual data with mention of entities
Mention – Reference in the sentence to an Entity (the word “Barack”)
Mention Level Relation – Relation (e.g. has_spouse) among mentions
Entity – Object in the real world (the person Barack)
Entity Linking – The process of referring mentions to entities (the words “Barack” refers to the entity Barack)
Entity Level Relation – Relation (e.g. has_spouse) among entities
Entity Level Data – Relational data in the KB
KBC Terminology

Entity-level data (knowledge base)

Entities

Barack Obama

has_spouse (entity-level relation)

Michelle Obama

Entity Linking

refers to

Mentions

"Barack"

has_spouse (mention-level relation)

"Michelle"

Mention-level data (sentences)

Barack and Michelle are married ...

KB
DeepDive – End to End framework

**Building KBC system:**

*Input* – Collection of unstructured data ranging from text document to existing but incomplete KB’s.

*Output* – Relational database containing facts extracted from the input.
DeepDive – End to End framework

The developer (user) develop the orange parts
DeepDive – 2 main phases

- **Grounding**
  SQL queries to produce a data structure called *factor graph* that describes a set of nodes and how they are correlated.

- **Inference**
  Statistical action using standard techniques on the *factor graph*. The output of the inference phase is marginal probability of every tuple in the database.
Data flow

Phase 1
- Data Preprocessing
- Extraction
- Factor Graph Generation

Phase 2
- Statistical Inference And Learning
- Error Analysis
DeepDive takes input data (articles in text format), loads them into a relational database:

• By default, DeepDive stores all documents in the database in one sentence per row with markup produced using standard *Natural Languages Processing (NLP)* pre-processing tools.

• The output is sentence-level information including words in each sentence, POS tags, named entity tags, etc.
DeepDive processes the data to create entities. It performs entity linking, feature extraction, and distant supervision, to create the variables (nodes) on which it will then perform inference.

• The results of extraction will be then used to build the factor graph according to rules specified by the user.
DeepDive executes 2 types of queries:

- **Candidate mapping** – SQL queries that produce possible mentions, entities and relations.

- **Feature extractors** – Associate features to candidates e.g., “… and his wife…” in the input file.
Candidate mapping:

Simple SQL queries with User Defined Functions (UDF) with low precision but high-recall. If candidate mapping misses a fact then DeepDive has no chance to extract it.

\[(R1) \text{MarriedCandidate}(m1, m2):- \]
\[
\text{PersonCandidate}(s, m1), \text{PersonCandidate}(s, m2).
\]
Feature extraction (2 ways):

• **User Defined functions** – Specified by the user.

• **Weights** – Which weight should be used for a given mentioning in a sentence.
Weight (example):

\[
\text{phrase}(m1, m2, \text{sent}) \Rightarrow \text{which weight should be used for a given phrase in a sentence e.g. “and his wife”}
\]
Distant Supervision:

Popular technique to create evidence in KBC systems.

Collecting examples from existing database for the relation we want to extract. We then use these examples to automatically generate our training data.

For example, database contains the fact: “Barack Obama and Michelle Obama are married”.

We take this fact, and then label each pair of "Barack Obama" and "Michelle Obama" that appear in the same sentence as a positive example for our marriage relation.
A **factor graph** is a type of probabilistic graphical model. It has two types of nodes:

- **Variables** - Either *evidence variables* when their value is known, or *query variables* when their value should be predicted.

- **Factors** - Define the relationships between variables in the graph. Each factor can be connected to many variables and comes with a *factor function* to define the relationship between these variables.
For example, if a factor node is connected to two variables nodes \( A \) and \( B \) then a possible factor function could be \( \text{imply}(A,B) \Rightarrow \text{if } (A = 1) \text{ then } (B = 1) \).

Each factor function has a weight associated. The weight is the confidence we have in the relationship expressed by the factor function.
The developer (user) writes SQL queries:

1. Instruct the system about which variables to create
2. How to connect them using *factor functions*.

These queries usually involve tables (evidence) from the extraction step.
Factor Graph Generation

**User Relations**

<table>
<thead>
<tr>
<th>R</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>r2</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>r3</td>
<td>a</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>y</th>
<th>Q</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>0</td>
<td>Q1</td>
<td>a</td>
</tr>
<tr>
<td>S2</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Inference Rules**

- **F1**: $q(x) \models R(x,y)$
- **F2**: $q(x) \models R(x,y), S(y)$

**Factor Graph**

- Variables: $V$
- Factors: $F$
- $F_1$, $F_2$, $q_1$
Grounding is the process of writing the graph to disk so that it can be used to perform inference.

DeepDive writes the graph to a set of five files: one for variables, one for factors, one for edges, one for weights, and one for metadata useful to the system. The format of these files is special so that they can be accepted as input by the sampler.
I. Sentence is processed into words, POS tags and named entity tags

II. Extracting (1) mentions of person and location, (2) candidate relations of has_spouse, and (3) features of candidates relations (e.g. words between mentions)

III. DeepDive uses rules written by developers (like inference_rule_1) to build a factor graph

```plaintext
inference_rule_1: {
  input_query: ""
  SELECT CandidateRelations.has_spouse, Features.feature
  FROM CandidateRelations, Features
  WHERE CandidateRelations.Subject=Features.Subject
  AND CandidateRelations.Object=Features.Object"
  function: "IsTrue(CandidateRelations.has_spouse)"
  weight: "?(Features.feature)"
}
```
The final step performs marginal inference on the factor graph variables to learn the probabilities of different values.

The inference step we take the grounded graph (i.e., the five files written during the grounding step) as input, and a number of arguments to specify the parameters for the learning procedure.

The values of factor weights specified in inference rules are calculated. These weights represent, intuitively, the confidence in the rule.
During the inference step, the marginal probabilities of the variables are computed, which in some cases, can represent the probability that a specific fact is true.

Like many other systems, DeepDive uses *Gibbs Sampling* to estimate the marginal probability of every tuple in the database.

*Gibbs Sampling* - is a Markov chain Monte Carlo (MCMC) algorithm for obtaining a sequence of observations which are approximated from a specified multivariate probability distribution, when direct sampling is difficult.
After inference, the results are written into a set of database tables.

The developer (user) can get results via a SQL query and perform error analysis to improve results.
At the end of the learning and inference, we have the marginal probability for each candidate fact.

Error analysis is the process of understanding the most common mistakes (incorrect extractions, too specific feature, candidate mistake, etc.) and deciding how to correct them.

The error analysis is written by the developer (user) using standard SQL queries.
DeepDive - Advantages

- No reference to the underlying machine learning algorithms. Enable debugging the system independently from the algorithms (inference phase).

- Developers write code, e.g. *feature extraction*, in familiar languages (such as Python, SQL, Scala).

- Familiar languages allows standard tools to inspect and visualize the data.

- The developer construct End-To-End system and then refines the quality of the system.
Appendices
Incremental KBC

To help the KBC system developer be more efficient, an incremental technique performed on the grounding and inference steps of KBC execution.

The approach to incrementally maintaining a KBC runs in 2 phases:

• **Incremental grounding** – The goal is to evaluate an update to for the DeepDive program to produce the “delta” of the modified factor graph, i.e. the modified variables $\Delta V$ and factors $\Delta F$.

• **Incremental inference** – The goal is by given $(\Delta V, \Delta F)$ to run statistical inference on the changed factor graph.
Standard technique for delta rules

DeepDive is based on SQL, therefore we are able to take advantage of decades of work on incremental view maintenance.

The input to this phase is the same as in the grounding phase, a set of SQL queries and the user schema.

The output of this phase is how the grounding changes, i.e., a set of modified variables $\Delta V$ and their factors $\Delta F$.

Since $V$ and $F$ are simply views over the database, any common view maintenance technique can be applied to the incremental grounding.
DeepDive uses algorithm named DRed which includes both additional and deletion.

In DRed, for each relation $R_i$ (table) in the user’s schema, we create a \textit{delta relation} $R_i^\delta$, with the same schema as $R_i$. For each tuple $t$ (row), $t\.count$ represent the number of derivations of $t$ in $R_i$. 
Standard technique for delta rules

On update, DeepDive update delta relations in 2 steps:

1. For tuples in $R^δ_i$, DeepDive directly updates the corresponding counts.

2. A SQL query called a “delta rule” written by the developer is executed which process this counts to generate modified variables $ΔV$ and factors $ΔF$.

*The overhead of DRed is modest and the gain is substantial.*
Novel technique for incremental maintenance inference

Given a set of \( (\Delta V, \Delta F) \), the goal is to compute the new distribution. We split the problem into 2 phases:

1. In the **Materialization phase**, we are given access to entire DeepDive program and we attempt to store information about the original distribution, denoted \( \Pr^{(0)} \).

2. In the **Inference phase**, we get the input from the **Materialization phase** and the \( (\Delta V, \Delta F) \). Our goal is to perform inference with respect to the changed distribution, denoted \( \Pr^{(\Delta)} \).
We present 3 techniques for the incremental inference phase on a *factor graph*:

**Strawman: Complete Materialization**

*Materialization phase* – We explicitly store the values of the probability \( \text{Pr}[I] \) for every possible world \( I \). This approach has perfect accuracy, but storing all possible worlds takes exponential amount of space and time in the numbers of \( V \) in the original *factor graph*.

*Inference phase* – We use *Gibbs Sampling*: Even if distribution has changed to \( \text{Pr}^{(\Delta)} \), we only need access to \( \text{Pr}[I] \) and to the new factors in \( \Delta F \) to perform *Gibbs* update. We get speed improvement because we don’t need to access all factors in the original and to perform a computation with them since we can look at them up in \( \text{Pr}[I] \).
**Sampling Approach**

At this approach we sample a set of possible worlds from the original distribution instead of all possible worlds. However, because we take samples from distribution which is different form the updated distribution $\Pr^{(\Delta)}$ we cannot use them directly, so we use standard scheme to ensure convergence to $\Pr^{(\Delta)}$.

**Variation Approach**

Rather storing the exact original distribution, we store a factor graph with fewer factors that will approximates the original distribution. On a smaller graph, running inference and learning is faster.
References

• Incremental Knowledge Base Construction Using DeepDive

Jaeho Shin†  Sen Wu†  Feiran Wang†  Christopher De Sa†  Ce Zhang††  Christopher Ré†
†Stanford University
††University of Wisconsin-Madison
{jaeho, senwu, feiran, cdesa, czhang, chrismre}@cs.stanford.edu

• http://deepdive.stanford.edu/
Questions

thanks for listening!