Web-Scale Knowledge Inference Using Markov Logic Networks

Yang Chen
Daisy Zhe Wang
University of Florida, Dept of Computer Science, Gainesville, FL 32611 USA

Abstract

In this paper, we present our on-going work on ProbKB, a PROBabilistic Knowledge Base constructed from web-scale extracted entities, facts, and rules represented as a Markov logic network (MLN). We aim at web-scale MLN inference by designing a novel relational model to represent MLNs and algorithms that apply rules in batches. Errors are handled in a principled and elegant manner to avoid error propagation and unnecessary resource consumption. MLNs infer from the input a factor graph that encodes a probability distribution over extracted and inferred facts. We run parallel Gibbs sampling algorithms on Graphlab to query this distribution. Initial experiment results show promising scalability of our approach.

1. Introduction

With the exponential growth in machine learning, statistical inference techniques, and big-data analytics frameworks, recent years have seen tremendous research interest in automatic information extraction and knowledge base construction. A knowledge base stores entities, facts, and their relationships in a machine-readable form so as to help machines understand information from human.

Currently, most popular techniques used to acquire knowledge include automatic information extraction (IE) from text corpus (Carlson et al., 2010; Poon et al., 2010; Schoenmackers et al., 2010) and massive human collaboration (Wikipedia, Freebase, etc). Though they proved their success in a broad range of applications, there is still much improvement we can make by making inference. Existing IE works extract entities, facts, and rules automatically from the web (Fader et al., 2011; Schoenmackers et al., 2010), but due to corpus noise and the inherent probabilistic nature of the learning algorithms, most of these extractions are uncertain. Our goal is to facilitate inference over such noisy, uncertain extractions in a principled, probabilistic, and scalable manner. To achieve this, we use Markov logic networks (MLNs) (Richardson & Domingos, 2006), an extension of first-order logic that augments each clause with a weight. Clauses with finite weight are allowed to be violated, but with a penalty determined by the weight. Together with all extracted entities and facts, the MLN defines a Markov network (factor graph) that encodes a probability distribution over all inferred facts. Probabilistic inference is thus supported by querying this distribution.

The main challenge related to MLNs is scalability. The state-of-the-art implementation, TUFFY (Niu et al., 2011) and FELIX (Niu et al., 2012), partially solved this problem by using relational databases and task specialization. However, their systems work well only on a small set of relations and hand-crafted MLNs, and were not able to scale up to the web-scale datasets like ReVerb and Sherlock (See Section 3). To gain this scalability, we design a relational model that pushes all facts and rules into the database. In this way, the grounding is reduced to a few joins among these tables and the rules are thus applied in batches. This helps achieve much greater scalability than TUFFY and FELIX.

The second challenge stems from extraction errors. Most IE systems assign a score to each of their extractions to indicate their fidelity so they typically don’t do any further cleaning. This poses a significant challenge to inference engines: errors propagate rapidly and violently without constraints. Figure 1 shows an example of such propagation: starting from a single er-
ror\textsuperscript{1} stating that Paris is the capital of Japan, a whole bunch of incorrect results are produced. Errors come from diverse sources: incorrectly extracted entities and facts, wrong rules, ambiguity, inference, etc. In each rule application, with a single erroneous facts participating, the result is likely to be erroneous. In a worse scenario, even if the extractions are correct, errors may arise due to word ambiguity. Removing errors early is a crucial task in knowledge base construction; it improves knowledge quality and saves computation resources for high-quality inferences.

This paper introduces the ProbKB system that aims at tackling these challenges and presents our initial results over a web-scale extraction dataset.

2. The ProbKB System

This section presents our on-going work on ProbKB. We designed a relational model to represent the extractions and the MLN and designed grounding algorithms in SQL that applies MLN clauses in batches. We implemented it on Greenplum, a massive parallel processing (MPP) database system. Inference is done by a parallel Gibbs sampler (Gonzalez et al., 2011) on GraphLab (Low et al., 2010). The overview of the system architecture is shown in Figure 2.

The remaining of this section is structured as follows: Section 2.1 justifies an important assumption regarding Horn clauses. Section 2.2 introduces our relational model for MLNs. Section 2.3 presents the grounding algorithms in terms of our relational model. Section 2.4 describes our initial work on maintaining knowledge integrity. Section 2.5 describes our inference engine built on GraphLab. Experiment results show ProbKB scales much better than the state-of-the-art.

2.1. First-Order Horn Clauses

A first-order Horn clause is a clause with at most one positive literal\textsuperscript{2}. In this paper, we focus on this class of clauses only, though in general Markov logic supports arbitrary forms. This restriction is reasonable in our context since our goal to discover implicit knowledge from explicit statements in the text corpus; this type of inference is mostly expressed as “if... then...” statements in human language, corresponding to the set of Horn clauses. Moreover, due to their simple structure, Horn clauses are more easily to learn and be represented in a structured form than general ones. We used a set of extracted Horn clauses from SHERLOCK (Schoenmackers et al., 2010).

2.2. The Relational MLN Model

We represent the knowledge base as a relational model. Though previous approaches already used relational databases to perform grounding (Niu et al., 2011; 2012), the MLN model and associating weights are stored in an external file. During runtime, a SQL query is constructed for each individual rule using a host language. This approach is inefficient when there are a large number of rules. Our approach, on the contrary, stores the MLN in the database so that the rules are applied in batches using joins. The only component that the database does not efficiently support is a probabilistic inference engine that needs many random accesses to the input data. This component is discussed in Section 2.5.

Based on the assumptions made in Section 2.1, we considered Horn clauses only. We classified SHERLOCK rules according to their sizes and argument orders. We identified six rule patterns in the dataset:

\textsuperscript{1}See http://en.wikipedia.org/w/index.php?title=Statement_(logic)&diff=545338546&oldid=269607611

\textsuperscript{2}http://en.wikipedia.org/wiki/Horn_clause
We use type 3 rules to illustrate the grounding algorithm in SQL.

\[ p(x, y) \leftarrow q(x, z), r(y, z) \]  \quad (3)

Assume these rules are stored in table \( M_3(p, q, r) \), and relationships \( p(x, y) \) are stored in \( R(p, x, y) \), then the following SQL query infers new facts given a set of extracted and already inferred facts:

```sql
SELECT DISTINCT M3.p AS p,
            R1.x AS x, R2.y AS y
FROM M3 JOIN R R1 ON M3.q = R1.p
    JOIN R R2 ON M3.r = R2.p
WHERE R1.y = R2.y;
```

This process is repeated until convergence (i.e. no more facts can be inferred). Then the following SQL query then generates the factors:

```sql
SELECT DISTINCT
            R1.id AS id1, R2.id AS id2, R3.id AS id3
FROM M3 JOIN R R ON M3.p = R.p
    JOIN R R1 ON M3.q = R1.p
    JOIN R R2 ON M3.r = R2.p
WHERE R.x = R1.x AND R.y = R2.x AND R1.y = R2.y;
```

To see why it is more efficient than TUFFY and ALCHEMY, consider the first query. Suppose we first compute \( RM_3 := M_3 \bowtie_{M_1.q=R.p} R \). Since the \( M_i \) tables are often small, we use a one-pass join algorithm and hash \( M_3 \) using \( q \). Then when each tuple \( (p, x, y) \) in \( R \) is read, it is matched against all rules in \( M_3 \) with the first argument being \( p \). In the second join \( RM_3 \bowtie_{RM_3.r=R.p \land RM_3.y=R.y} R \), since \( RM_3 \) is typically much larger than \( M_3 \), we assume using a two-pass hash-join algorithm, which starts by hasing \( RM_3 \) and \( R \) into buckets using keys \( (r, y) \) and \( (p, y) \), respectively. Then, for any tuple \( (p, x, y) \) in \( R \), all possible results can be formed in one pass by considering tuples from \( RM_3 \) in the corresponding bucket. As a result, each tuple is read into memory for at most 3 times and rules are simultaneously applied. This is in sharp contrast with TUFFY, where tuples have to be read into main memory as many times as the relation appears in the rules.

Our performance gain over TUFFY owes also to the simple syntax of Horn clauses. In order to support general first-order clauses, TUFFY materializes each possible grounding of all clauses to identify a set of
active clauses (Singla & Domingos, 2006)\(^3\). This process quickly uses up disk space when trying to ground a dataset with large numbers of entities, relations and clauses like SHERLOCK. Our algorithms avoid this time- and space-consuming process by taking advantage of simplicity of Horn clauses.

The result of grounding is a factor graph (Markov network). This graph encodes a probability distribution over its variable nodes, which can be used to answer user queries. However, marginal inference in Markov networks is \#P-complete (Roth, 1996), so we turn to approximate inference methods. The state-of-the-art marginal inference algorithm is MC-SAT (Poon & Domingos, 2006), but due to absence of deterministic rules and access to efficient parallel implementations of the widely-adopted Gibbs sampler, we stucked to it and present an initial evaluation in Section 3.2.

2.4. Knowledge Integrity

(Schoenmackers et al., 2010) observed a few common error patterns (ambiguity, meaningless relations, etc) that might affect learning quality and tried to remove these errors to obtain a cleaner corpus to work on. This manual pruning strategy is useful, but not enough, for an inference engine since errors often arise and propagate in unexpected manners that are hard to describe systematically. For example, the fact “Paris is the capital of Japan” shown in Figure 1 is accidentally extracted from a Wikipedia page that describes logical statement, and no heuristic in (Schoenmackers et al., 2010) is able to filter it out. Unfortunately, even a single and unfrequent error like this propagates rapidly in the inference chain. These errors hamper knowledge quality, waste computation resources, and are hard to catch.

Instead of trying to enumerate common error patterns, we feel that it is much easier to identify correct inferences: facts that are extracted from multiple sources, or inferred by facts from multiple sources are more likely to be correct than others. New errors are less likely to arise when we applied rules to this subset of facts. Thus, we split the database for the facts, moving the qualified facts to another table called beliefs. The state-of-the-art marginal inference algorithm is MC-SAT (Poon & Domingos, 2006), but due to absence of deterministic rules and access to efficient parallel implementations of the widely-adopted Gibbs sampler, we stucked to it and present an initial evaluation in Section 3.2.

In Algorithm 1, infer is almost the same as discussed in Section 2.3, except that the operands are replaced by beliefs and delta, potentially much smaller than the original \(R\). The promote algorithm is what we are still working on. It takes a new set of inference results and uses them to promote candidates to beliefs. Our intuition was to exploit lineage and promote facts implied by most external sources or to learn a set of constraints to detect erroneous rules, facts, and join keys.

2.5. The GraphLab Inference Engine

The state-of-the-art MLN inference algorithm is MC-SAT (Singla & Domingos, 2006). For an initial evaluation and access to existing libraries, though, our prototype used a parallel Gibbs sampler (Gonzalez et al., 2011) implemented on the GraphLab (Low et al., 2010; 2012) framework. GraphLab is a distributed framework that improves upon abstractions like MapReduce for asynchronous iterative algorithms with sparse computational dependencies while ensuring data consistency and achieving a high degree of parallel performance. We ran the parallel Gibbs algorithm on our grounded factor graph and got better results than TUFFY.

Though the performance looks good, one concern we have is coordinating the two systems. Our initial goal is to build a coherent system that works on MLN inference problem. Getting two independent systems to

\[\forall x \forall y (p(x, y) \lor q(x, y)) \lor \exists x q(x)\]. These clauses can be made unsatisfied only by considering all possible assignments for \(x\) and \(y\).
work together is hard and error-prone. Synchronization becomes especially troublesome: to query even a single atom, we need to get output from GraphLab, write the results to the database, and perform a database query to get the result. Motivated by this, we are trying to work out a shared memory model that pushes external operators directly into the database. In Section 3.2, we only present results from GraphLab.

3. Experiments

We used extracted entities and facts from ReVerb (Fader et al., 2011) and applied Sherlock rules (Schoenmackers et al., 2010) to discover implicit knowledge. Sherlock uses TextRunner extractions, an older version of ReVerb, to learn the rules, so there is a schema mismatch between the two. We are working on resolving this issue by either mapping the ReVerb schema to Sherlock’s or implementing our own rule learner for ReVerb, but for now we use 50K out of 400K extracted facts to present the initial results. The statistics of our dataset is shown in Table 1.

<table>
<thead>
<tr>
<th>#entities</th>
<th>480K</th>
</tr>
</thead>
<tbody>
<tr>
<td>#relations</td>
<td>10K</td>
</tr>
<tr>
<td>#facts</td>
<td>50K</td>
</tr>
<tr>
<td>#rules</td>
<td>31K</td>
</tr>
</tbody>
</table>

Table 1. ReVerb-Sherlock data statistics.

Experiment Setup We ran all the experiments on a 32-core machine at 1400MHz with 64GB of RAM running Red Hat Linux 4. ProbKB, Tuffy, and GraphLab are implemented in SQL, SQL (using Java as a host language), and C++, respectively. The database system we use is Greenplum 4.2.

3.1. Grounding

We used Tuffy as a baseline comparison. Before grounding, we remove some ambiguous mentions, including common first or last names and general class references (Schoenmackers et al., 2010). The resulting dataset has 7K relationships. We run ProbKB and Tuffy using this cleaner dataset and learn 100K new facts. Performance result is shown in Table 2.

<table>
<thead>
<tr>
<th>System</th>
<th>Time/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProbKB</td>
<td>85</td>
</tr>
<tr>
<td>Tuffy</td>
<td>Crash</td>
</tr>
</tbody>
</table>

Table 2. Grounding time for ProbKB and Tuffy.

As discussed in Section 2.3, the reason for our performance improvement is the Horn clauses assumption and batch rule application. In order to support general MLN inference, Tuffy materializes all ground atoms for all predicates. This makes the active closure algorithm efficient but consumes too much disk space. For a dataset with large numbers of entities and relations like ReVerb-Sherlock, the disk space is used up even before the grounding starts.

3.2. Inference

This section reports ProbKB’s performance for inference using GraphLab. Again we used Tuffy as comparison. Since Tuffy cannot ground the whole ReVerb-Sherlock dataset, we sampled a subset with 700 facts and compared the time spent on generating 200 joint samples for ProbKB and Tuffy. The result is shown in Table 3.

<table>
<thead>
<tr>
<th>System</th>
<th>Time/min</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProbKB</td>
<td>0.47</td>
</tr>
<tr>
<td>Tuffy</td>
<td>55</td>
</tr>
</tbody>
</table>

Table 3. Time to generate 200 joint samples.

The performance boost owes mostly to the GraphLab parallel engine.

The experimental results presented in this section are preliminary, but it shows the need for better grounding and inference system to achieve web scale and the promise of our proposed techniques to solve the scalability and integrity challenges.

4. Conclusion and Future Work

This paper presents our on-going work on ProbKB. We built an initial prototype that stores MLNs in a relational form and designed an efficient grounding algorithm that applies rules in batches. We maintained a computational-safe set of “beliefs” to which rules are applied with minimal errors occurring. We connect to GraphLab for a parallel Gibbs sampler inference engine. Future works are discussed throughout the paper and are summarized below:

- Develop techniques for the robust semi-naive algorithm to maintain knowledge integrity.
- Tightly integrate grounding and inference over MLN using shared-memory model.
- Port SQL implementation of grounding to other frameworks, such as Hive and Shark.
References


