WHIRL in ProbLog

Wannes Meert, Anton Dries, Angelika Kimmig
KU Leuven, Belgium
firstname.lastname@cs.kuleuven.be

Abstract

We present how WHIRL can be modelled as a ProbLog program using ProbLog's Python interface to execute information retrieval algorithms using standard toolkits such as scikit-learn and the natural language toolkit.

Introduction

The "information representation language" WHIRL synergistically combines properties of logic-based and text-based representation systems (Cohen 2000b). It extends a subset of Datalog with an atomic type for textual entities, an atomic operation for computing textual similarity, and a "soft" semantics. The answer to a WHIRL query is a list of answer substitutions, each associated with a numerical score or confidence, and presented in decreasing order. A key application of WHIRL is data integration between distributed, heterogeneous databases, where similarity joins are used to overcome the lack of common object identifiers (Cohen 2000a).

In this paper, we present an implementation of WHIRL in the probabilistic programming language ProbLog. We use ProbLog's Python interface to execute information retrieval algorithms using well-known toolkits such as scikit-learn (Pedregosa et al. 2011) and the natural language toolkit (NLTK) (Bird, Loper, and Klein 2009).

ProbLog

The probabilistic programming language ProbLog extends Prolog based on Sato's distribution semantics (De Raedt, Kimmig, and Toivonen 2007; Fierens et al. 2015). A ProbLog program has two parts: (1) a probabilistic part that defines a probability distribution over truth values of a subset of the program's atoms, and (2) a logical part that derives truth values of remaining atoms using a reasoning mechanism similar to Prolog. While the latter part simply contains Prolog clauses, the former is specified by *probabilistic facts* $\mathtt{p} :: \mathtt{fact}$, meaning that `fact` is true with probability `p`. All these are probabilistically independent; in case they contain variables, all ground instances are independent as well.

For ease of modeling, ProbLog also allows the use of *annotated disjunctions* $\mathtt{p_1} :: \mathtt{h_1}; \ldots; \mathtt{p_n} :: \mathtt{h_n} :- \mathtt{body}$ with $\sum_{i=1}^{n} \mathtt{p_i} \leq 1$, meaning that if `body` is true, one of the $\mathtt{h_i}$ will be true according to the specified probabilities $\mathtt{p_i}$, or none of them is true (with probability $1 - \sum_{i=1}^{n} \mathtt{p_i}$).

The ProbLog inference task most relevant to this paper consists of finding all answer substitutions of a query together with their probabilities of being true.

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

WHIRL

WHIRL uses the concept of a *score* to introduce soft semantics for logic programs. Specifically, WHIRL allows to (a) associate a score or confidence with extensionally defined predicates (or facts) and to (b) use *similarity literals* of the form $X \sim Y$, where X and Y are logical variables. The score of a similarity literal is typically given by cosine similarity under the TF-IDF weighing scheme. Scores of conjunctive or disjunctive queries are then computed as respectively the multiplication or addition of the individual scores.

WHIRL's score directly maps to ProbLog probabilities, and its Datalog queries to ProbLog clauses, where we write $X \sim Y$ as `similar(X,Y)`. We use NLTK to create a bag-of-words feature vector, the scikit-learn toolboxes to compute the TF-IDF similarity and connect to ProbLog using its Python external interface. The similarity computation could be optimized by using advanced indexing as in WHIRL.

```
@problog_external('+string','+string','-float')
def similarity(doc1, doc2):
  b1,b2 = bag_words(doc1),bag_words(doc2)
  v1 = tfidf.transform(b1)
  v2 = tfidf.transform(b2)
  return = cosine_similarity(v1,v2)
```

The TF-IDF weights are computed when loading the whirl.py module. In ProbLog the Python definition can be called to associate its output with a probabilistic fact:

```
:- use_module('whirl.py').
P::similar(X,Y):- similarity(X,Y,P),P>0.3.
```

This labels each ground `similar/2` fact with the probability that the two strings are identical, excluding cases with low similarity (≤ 0.3) to reduce complexity. Given a set of reviews the probability of `similar('comedy Smith','space Smith')` is for example 0.77 because Smith only appears in a small fraction of all movies, which makes it a good indicator of document similarity.

Example 1. We illustrate WHIRL in ProbLog using the movie listings examples presented in (Co-

hen 2000b). The database contains items such as:

```
listing('Sony Mountainside Theater',
        'Men in Black', '9:30-10:10').
review('Men in Black, 1997',
 '... comedy about space aliens with Will Smith ...').
academy_award('Best makeup').
winner('Men in Black','Best makeup').
```

The movie title is not a unique identifier, as movie titles may vary due to extra words such as the release date or the subtitle, spelling mistakes, or incorrect translations. In contrast to WHIRL, ProbLog does not restrict given scores to extensional facts, but can evaluate scores defined via ProbLog code during inference.

Conjunctive Queries

Conjunctive WHIRL queries are Datalog queries combining multiple conditions, whose scores are multiplied to obtain the score of an answer. They directly translate to ProbLog queries as illustrated in the following examples.

Example 2. Reviews about comedies with space aliens:

```
q2(Movie,Rev):- review(Movie,Rev),
  similar(Rev,'comedy with space aliens').
```

The outcome of this query is a list of movie-review pairs and their scores, which may contain, for example, the movie Men in Black with a probability of 0.54. It is not 1.0 because the review is longer and thus not identical.

Example 3. Queries can also contain multiple similarity atoms as well as similarity atoms over two logical variables, for instance to ask where to view a science fiction comedy:

```
q3(Mov,Theater):- review(M2,R),
 listing(Theater,Mov,T),similar(Mov,M2),
 similar(R,'comedy with space aliens').
```

Disjunctive Queries

A disjunctive WHIRL query, also called *disjunctive view*, expresses a set of alternative, sufficient conditions. It is written as a set of clauses with identical heads. If multiple conditions are satisfied simultaneously, WHIRL combines scores as a noisy-or, just as ProbLog does.

Example 4. Find cinemas that are playing either a science fiction comedy or an animated film produced by Disney

```
view(Theater):- listing(Theater,M1,T),
  review(M2,R), similar(M1,M2),
  similar(R,'comedy with space aliens').
view(Theater):- listing(Theater,M1,T),
  review(M2,R), similar(M1,M2),
  similar(R,'animated Walt Disney film').
q4(Theater):- view(Theater).
```

Soft Universal Quantification

WHIRL queries can use a soft version of universal quantification, expressed by the `many(Template,Test)` operator where `Test` is a conjunction of ordinary literals and `Template` is a single literal `p(Y1,...,Yn)`. The score of `many/2` is the average score of the `Test` conjunction on items that match the `Template`.

Example 5. A query to find movies that are currently playing and have won many academy awards:

```
q5(M):- listing(_,M,_),
  many(academy_award(Y),winner(M,Y)).
```

Part of the outcome is that Men in Black is ranked higher than Hercules because it has won more academy awards.

The `many/2` predicate is easily implemented in ProbLog using its probabilistic `findall/3` meta-call. The advantage of ProbLog, however, is that it relaxes the constraints on using the `many/2` predicate present in the WHIRL system as it is not required that `Template` is an extensional predicate, the `Yi` do not have to be distinct and it is allowed to have multiple, possibly nested, `many/2` calls.

```
many2(L):- many2(L,0,0,L).
many2([],P,N,L):- T is P+N, T > 0, S is P/T,w(S,L).
S::w(S,_).
many2([H|T],PA,NA,S) :-
  (  call(H),PAN is PA+1,NAN is NA;
   \+call(H),PAN is PA,  NAN is NA+1),
  many2(T,PAN,NAN,S).
many(Template,Test) :-
  findall(Test, Template, L),many2(L).
```

The code constructs a binomial tree of all possible combinations of which calls succeed. Additionally, the `many2/1` predicate can be optimzed to achieve inference with complexity linear in the number of the literals returned by `findall/3`.

Discussion

We have shown how to express WHIRL queries in ProbLog. This allows us to directly use ProbLog inference to obtain all query answers and their probabilities. An interesting direction for future work is to implement a top answers inference algorithm for ProbLog along the lines of WHIRL's ranking algorithm but for general programs, which focuses on quickly finding the highest scoring answers without looking at all possible answers.

References

Bird, S.; Loper, E.; and Klein, E. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

Cohen, W. 2000a. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems* 18(3):288–321.

Cohen, W. 2000b. Whirl: A word-based information representation language. *Artificial Intelligence* 118(1):163–196.

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. ProbLog: A probabilistic Prolog and its application in link discovery. In *20th International Joint Conference on Artificial Intelligence (IJCAI)*, 2468–2473.

Fierens, D.; Van den Broeck, G.; Renkens, J.; Shterionov, D.; Gutmann, B.; Thon, I.; Janssens, G.; and De Raedt, L. 2015. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming* 15(03):358–401.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.