Expert Systems With Applications 87 (2017) 291–303

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Improving scalability of inductive logic programming via pruning and best-effort optimisation

Mishal Kazmi[a], Peter Schüller[b,*], Yücel Saygın[a]

[a] Faculty of Engineering and Natural Science, Sabanci University, Istanbul, Turkey
[b] Faculty of Engineering, Marmara University, Istanbul, Turkey

ARTICLE INFO

Article history:
Received 17 March 2017
Revised 19 May 2017
Accepted 9 June 2017
Available online 16 June 2017

Keywords:
Answer Set Programming
Inductive logic programming
Natural Language Processing
Chunking

ABSTRACT

Inductive Logic Programming (ILP) combines rule-based and statistical artificial intelligence methods, by learning a hypothesis comprising a set of rules given background knowledge and constraints for the search space. We focus on extending the XHAIL algorithm for ILP which is based on Answer Set Programming and we evaluate our extensions using the Natural Language Processing application of sentence chunking. With respect to processing natural language, ILP can cater for the constant change in how we use language on a daily basis. At the same time, ILP does not require huge amounts of training examples such as other statistical methods and produces interpretable results, that means a set of rules, which can be analysed and tweaked if necessary. As contributions we extend XHAIL with (i) a pruning mechanism within the hypothesis generalisation algorithm which enables learning from larger datasets, (ii) a better usage of modern solver technology using recently developed optimisation methods, and (iii) a time budget that permits the usage of suboptimal results. We evaluate these improvements on the task of sentence chunking using three datasets from a recent SemEval competition. Results show that our improvements allow for learning on bigger datasets with results that are of similar quality to state-of-the-art systems on the same task. Moreover, we compare the hypotheses obtained on datasets to gain insights on the structure of each dataset.

1. Introduction

Inductive Logic Programming (ILP) (Muggleton & De Raedt, 1994) is a formalism where a set of logical rules is learned from a set of examples and a background knowledge theory. By combining rule-based and statistical artificial intelligence, ILP overcomes the brittleness of pure logic-based approaches and the lack of interpretability of models of most statistical methods such as neural networks or support vector machines. We here focus on ILP that is based on Answer Set Programming (ASP) as our underlying logic programming language because we aim to apply ILP to Natural Language Processing (NLP) applications such as Machine Translation, Summarization, Coreference Resolution, or Parsing that require nonmonotonic reasoning with exceptions and complex background theories.

In our work, we apply ILP to the NLP task of sentence chunking. Chunking, also known as 'shallow parsing', is the identification of short phrases such as noun phrases which mainly rely on Part of Speech (POS) tags. In our experiments on sentence chunking (Tjong Kim Sang & Buchholz, 2000) we encountered several problems with state-of-the-art ASP-based ILP systems XHAIL (Ray, 2009), ILED (Katzouris, Artikis, & Paliouras, 2015), and ILASP2 (Law, Russo, & Broda, 2015). XHAIL and ILASP2 showed scalability issues already with 100 sentences as training data. ILED is designed to be highly scalable but failed in the presence of simple inconsistencies in examples. We decided to investigate the issue in the XHAIL system, which is open-source and documented well, and we made the following observations:

(i) XHAIL only terminates if it finds a provably optimal hypothesis,
(ii) the hypothesis search is done over all potentially beneficial rules that are supported by at least one example, and
(iii) XHAIL contains redundancies in hypothesis search and uses outdated ASP technology.

In larger datasets, observation (i) is unrealistic, because finding a near-optimal solution is much easier than proving optimality of the best solution, moreover in classical machine learning suboptimal solutions obtained via non-exact methods routinely provide state-of-the-art results. Similarly, observation (ii) makes it harder

* Corresponding author.
E-mail addresses: mishalkazmi@sabanciuniv.edu (M. Kazmi), peter.schuller@marmara.edu.tr, schueller.p@gmail.com (P. Schüller), ysaygin@sabanciuniv.edu (Y. Saygın).

http://dx.doi.org/10.1016/j.eswa.2017.06.013
0957-4174/

to find a hypothesis, and it generates an overfitting hypotheses which contains rules that are only required for a single example. Observation (iii) points out an engineering problem that can be remedied with little theoretical effort.

To overcome the above issues, we modified the XHAIL algorithm and software, and we performed experiments on a simple NLP chunking task to evaluate our modifications.

In detail, we make the following contributions.

- We extend XHAIL with best-effort optimisation using the newest ASP optimisation technology of unsat-core optimisation (Andres, Kaufmann, Matheis, & Schaub, 2012) with stratification (Alviano, Dodaro, Marques-Silva, & Ricca, 2015; Ansótegui, Bonet, & Levy, 2013) and core shrinking (Alviano & Dodaro, 2016) using the WASP2 (Alviano, Dodaro, Faber, Leone, & Ricca, 2013; Alviano, Dodaro, Leone, & Ricca, 2015) solver and the Gringo (Gebser, Kaminski, König, & Schaub, 2011) grounder. We also extend XHAIL to provide information about the optimality of the hypothesis.
- We extend the XHAIL algorithm with a parameter *Pr* for pruning, such that XHAIL searches for hypotheses without considering rules that are supported by fewer than *Pr* examples.
- We eliminate several redundancies in XHAIL by changing its internal data structures.
- We describe a framework for chunking with ILP, based on preprocessing with Stanford Core NLP (Manning et al., 2014) tools.
- We experimentally analyse the relationship between the pruning parameter, number of training examples, and prediction score on the sentence chunking (Tjong Kim Sang & Buchholz, 2000) subtask of iSTS at SemEval 2016 (Agirre et al., 2016).
- We discuss the best hypothesis found for each of the three datasets in the SemEval task, and we discuss what can be learned about the dataset from these hypotheses.

Only if we use all the above modifications together, XHAIL becomes applicable in this chunking task. By learning a hypothesis from 500 examples, we can achieve results competitive with state-of-the-art systems used in the SemEval 2016 competition.

Our extensions and modifications of the XHAIL software are available in a public fork of the official XHAIL Git repository (Bragaglia & Schüller, 2016).

In Section 2 we provide an overview of logic programming and ILP. Section 3 gives an account of related work and available ILP tools. In Section 4 we describe the XHAIL system and our extensions of pruning, best-effort optimisation, and further improvements. Section 5 gives details of our representation of the chunking task. In Section 6 we discuss empirical experiments and results. We conclude in Section 7 with a brief outlook on future work.

2. Background

We next introduce logic programming and based on that inductive logic programming.

2.1. Logic programming

A logic programs theory normally comprises of an alphabet (variable, constant, quantifier, etc.), vocabulary, logical symbols, a set of axioms and inference rules (Lloyd, 2012). A logic programming system consists of two portions: the logic and control. Logic describes what kind of problem needs to be solved and control is how that problem can be solved. An ideal of logic programming is for it to be purely declarative. The popular Prolog (Clocksin & Mellish, 2003) system evaluates rules using resolution, which makes the result of a Prolog program depending on the order of its rules and on the order of the bodies of its rules. Answer Set Programming (ASP) (Brewka, Eiter, & Truszczyński, 2011; Gebser, Kaminski, Kaufmann, & Schaub, 2012) is a more recent logic programming formalism, featuring more declarativity than Prolog by defining semantics based on Herbrand models (Gelfond & Lifschitz, 1988). Hence the order of rules and the order of the body of the rules does not matter in ASP. Most ASP programs follow the Generate-Define-Test structure (Lifschitz, 2002) to (i) generate a space of potential solutions, (ii) define auxiliary concepts, and (iii) test to invalidate solutions using constraints or incurring a cost on non-preferred solutions.

An ASP program consists of rules of the following structure:

$$a \leftarrow b_1, \ \ldots, b_m, \mathbf{not}\, b_{m+1}, \ldots, \mathbf{not}\, b_n$$

where a, b_i are atoms from a first-order language, a is the head and $b_1, \ldots, \mathbf{not}\, b_n$ is the body of the rule, and **not** is negation as failure. Variables start with capital letters, facts (rules without body condition) are written as 'a.' instead of '$a \leftarrow$ '. Intuitively a is true if all positive body atoms are true and no negative body atom is true.

The formalism can be understood more clearly by considering the following sentence as a simple example:

Computers are normally fast machines unless they are old.

This would be represented as a logical rule as follows:

$$\mathit{fastmachine(X)} \leftarrow \mathit{computer(X)}, \mathbf{not}\, \mathit{old}(X).$$

where X is a variable, *fastmachine, computer*, and *old* are predicates, and *old(X)* is a negated atom.

Adding more knowledge results in a change of a previous understanding, this is common in human reasoning. Classical First Order Logic does not allow such non-monotonic reasoning, however, ASP was designed as a commonsense reasoning formalism: a program has zero or more answer sets as solutions, adding knowledge to the program can remove answer sets as well as produce new ones. Note that ASP semantics rule out self-founded truths in answer sets. We use the ASP formalism due to its flexibility and declarativity. For formal details and a complete description of syntax and semantics see the ASP-Core-2 standard (Calimeri et al., 2012). ASP has been applied to several problems related to Natural Language Processing, see for example Mitra and Baral (2016), Schwitter (2012), Schüller (2013, 2014, 2016) and Sharma, Vo, Aditya, and Baral (2015). An overview of applications of ASP in general can be found in Erdem, Gelfond, and Leone (2016).

2.2. Inductive logic programming

Processing natural language based on hand-crafted rules is impractical because human language is constantly evolving, partially due to the human creativity of language use. An example of this was recently noticed on UK highways where they advised drivers, 'Don't Pokémon Go and drive'. Pokémon Go is being informally used here as a verb even though it was only introduced as a game a few weeks before the sign was put up. To produce robust systems, it is necessary to use statistical models of language. These models are often pure Machine Learning (ML) estimators without any rule components (Manning & Schütze, 1999). ML methods work very well in practice, however, they usually do not provide a way for explaining why a certain prediction was made, because they represent the learned knowledge in big matrices of real numbers. Some popular classifiers used for processing natural language include Naive Bayes, Decision Trees, Neural Networks, and Support Vector Machines (SVMs) (Dumais, Platt, Heckerman, & Sahami, 1998).

In this work, we focus on an approach that combines rule-based methods and statistics and provides interpretable learned models: *Inductive Logic Programming* (ILP). ILP is differentiated from ML techniques by its use of an expressive representation language and