Meta-Interpretive Learning of Data Transformation Programs

Andrew Cropper, Alireza Tamaddoni-Nezhad, and Stephen H. Muggleton

Department of Computing, Imperial College London

Abstract. Data Transformation is an important part of *data curation*, which involves the maintenance of research data on a long-term basis with the aim of allowing its re-use. This activity, wide-spread in commercial and academic data analytics projects, is labour intensive, involving the manual construction and debugging of large numbers of small, special purpose data transformation programs. This paper investigates learning data transformation programs from examples provided by the user. Though small, such programs can be relatively complex, involving problem decomposition, repetition and recognition of context. These features together with the need to learn from a handful of user examples make this a hard task for standard ILP systems. We use a Meta-Interpretive Learning approach which supports these various requirements through predicate invention and the learning of recursive programs from small numbers of examples. In our approach we develop a general technique for applying $Metagol_{DF}$ to both semi-structured and unstructured data. Our experiments are conducted on tasks involving semi-structured medical patient data and un-structured ecological text. The results indicate that high levels of predictive accuracy can be achieved in these tasks from small number of training examples.

1 Introduction

Suppose you are given a large number of patient records in a semi-structured format and are required to transform them to a given structured format as shown in Fig. 1. In order to avoid manually editing the transformations, you might decide to write a small program to extract the relevant fields from the input and produce them in the output. Fig. 1c illustrates the kind of Prolog transformation program for this task which can be generated automatically from a small number of examples such as those in Fig. 1a (see Section 5). In this program, predicate invention is used to introduce $f1 - f2$ and the primitive background predicates *find_patient_id/2*, *find_int/2*, and *open_interval/4* are used to identify the various fields to be transposed from input to output.

P_001
67 year
lung disease: n/a, Diagnosis: Unknown
80.78
P_003
56
Diagnosis: carcinoma, lung disease: unknown
20.78
P_013
70
Diagnosis: pneumonia
55.9

(a) Input

P_001	67	Unknown
P_003	56	carcinoma
P_013	70	pneumonia

(b) Output

```
f(A,B):- f2(A,C), f1(C,B).
f2(A,B):- find_patient_id(A,C) , find_int(C,B) .
f1(A,B):- open_interval(A,B,[':',' '],[';'n']).
f1(A,B):- open_interval(A,B,[':',' '],[',',' ']).
```

(c) Learned program

Fig. 1: Transformation of medical records from semi-structured format

This paper uses the recently developed Meta-Interpretive Learning (MIL) framework [9, 3, 4]. MIL differs from most state-of-the-art ILP approaches by supporting predicate invention for problem decomposition

and the learning of recursive programs. MIL has [10] been shown to be capable of learning a simple robot *strategy* for building a stable wall from a small set of initial/final state pairs. The work in this paper uses the same general approach as that employed in [7] for building string transformation programs. Here we look at the more general problem of learning data transformation programs.

The paper is arranged as follows. Section 2 describes related work. Section 3 describes the theoretical framework. Section 4 describes the framework implementation. Section 5 describes experiments in ecological and medical domains. Finally, Section 6 concludes the paper and details future work.

2 Related work

In [2] the authors compared statistical and relational methods to extract facts from a MEDLINE corpus. The primary limitation of the statistical approach, they state, is its inability to express the linguistic structure of the text; by contrast, the relational approach allows these features to be encoded as background knowledge, as parse trees, specifically. The relational approach used an ILP system similar to FOIL [11] to learn extraction rules. Similar works include [5], who used the ILP system Aleph [13] to learn rules to extract relations from a MEDLINE corpus; and [1], who used the ILP system FOIL to learn rules to extract relations from Nature and New Scientist articles. These works focused on constructing the appropriate problem representation, including determining the necessary linguistic features to be included in the background knowledge. In contrast to these approaches, we use the state-of-the-art ILP system Metagol$_{DF}$, which supports predicate invention, the learning of recursive theories, and positive-only learning, none of which is supported by FOIL nor Aleph.

FlashExtract [6] is a framework to extract fields from documents, such as text files and web pages. In this framework, the user highlights one or two examples of each field in document and FlashExtract attempts to extract all other instances of such fields, arranging them in a structured format, such as a table. FlashExtract uses an inductive synthesis algorithm to synthesise the extraction program using a domain-specific language built upon a pre-specified algebra of few core operators (map, filter, merge, and pair). In contrast to FlashExtact, our approach allows for the inclusion of background knowledge.

Wu and Knoblock [15] recently describe a Programming-by-Example technique which iteratively learns data transformation programs by example. While the overall aims of their approach are similar to ours, their approach does not support automatic problem decomposition using predicate invention.

3 Framework

MIL [9, 10] is a form of ILP based on an adapted Prolog meta-interpreter. Whereas a standard Prolog meta-interpreter attempts to prove a goal by repeatedly fetching first-order clauses whose heads unify with a given goal, a MIL learner attempts to prove a set of goals by repeatedly fetching higher-order metarules (Fig. 2b) whose heads unify with a given goal. The resulting meta-substitutions are saved in an abduction store, and can be re-used in later proofs. Following the proof of a set of goals, a hypothesis is formed by applying the meta-substitutions onto their corresponding metarules, allowing for a form of ILP which supports predicate invention and the learning of recursive theories.

General formal framework In the general framework for data transformation we assumes that the user provides examples E of how data should be transformed. Each example $e \in E$ consists of a pair $\langle d1, d2 \rangle$ where $d1 \in D1$ and $d2 \in D2$ are input and output data records respectively. Given background knowledge B, in the form of existing transformations and the user-provided examples E the aim of the learning is to generate a transformational function $\tau : D1 \rightarrow D2$ such that $B, \tau \models E$.

4 Implementation

Fig. 2a shows Metagol$_{DF}$[7], an implementation of the MIL framework, similar in form to a standard Prolog meta-interpreter. The metarule set (Fig. 2b) is defined separately.

```
prove([], Prog, Prog).
prove([Atom|As], Prog1, Prog2) :-
  metarule(Name, MetaSub, (Atom :- Body), Order),
  Order,
  abduce(metasub(Name, MetaSub), Prog1, Prog3),
  prove(Body, Prog3, Prog4),
  prove(As, Prog4, Prog2).
```

(a) Prolog code for generalised meta-interpreter

Name	**Metarule**	**Order**
Base	$P(x,y) \leftarrow Q(x,y)$	$P \succ Q$
Chain	$P(x,y) \leftarrow Q(x,z), R(z,y)$	$P \succ Q, P \succ R$
Curry	$P(x,y) \leftarrow Q(x,y,c_1,c_2)$	$P \succ Q$
TailRec	$P(x,y) \leftarrow Q(x,z), P(z,y)$	$P \succ Q, x \succ z \succ y$

(b) Metarules with associated ordering constraints, where $\succ$ is a pre-defined ordering over symbols in the signature. The letters P, Q, and R denote existentially quantified higher-order variables; x, y, and z denote universally quantified first-order variables; and c_1 and c_2 denote existentially quantified first-order variables.

Fig. 2: Metagol$_{DF}$ meta-interpreter (a) and metarules (b)

4.1 Transformation language

The transformation framework[1] includes three predicates: find_sublist/3, closed_interval/4, and open_interval/4. These predicates transform an input state to an output state. The find_sublist/3 predicate finds a complete sublist in an input and writes it to the output. The closed_interval/4 and open_interval/4 predicates find a sublist denoted by start and end delimiters, and write it to the output. For example, if the input = [i,n,d,u,c,t,i,o,n], start delimiter = [n,d], and end delimiter = [t,i], then the predicate open_interval/4 predicate writes [u,c] to the output, and the predicate closed_interval/4 writes [n,d,u,c,t,i] to the output. The delimiters are the set of all sublists of length k in an input. The value of k is pre-specified as part of the background knowledge, and, as the experiments demonstrate in Section 5, affects the predictive accuracy of the learner.

5 Experiments

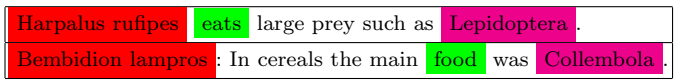
This section details experiments in learning data transformation programs. We test the following null hypothesis: *Metagol$_{DF}$ cannot learn data transformation programs with higher than default predictive accuracies*. To test this hypothesis, we apply our framework in two domains: ecological scholarly papers, and medical patient records. In all experiments, we use the *Chain* and *Curry* metarules (Fig. 2b).

5.1 Ecological scholarly papers

In this experiment, the aim is to learn programs to extract relations from natural language taken from ecological scholarly papers.

Materials The dataset contains 32 positive real-world examples taken from ecological scholarly papers (adopted from [14]) of a natural language sentence paired with a list of relations to be extracted. Fig. 3 shows two example pairings, and the partial background knowledge used in the experiment, where natural language sentences are represented as lists of terms. We provide a complete set of known species, but we do not provide predation terms in the background knowledge, so Metagol must use the general purpose background predicates, described in Section 4, to learn a solution. To generate negative testing examples, we create a frequency distribution over input lengths from the training examples and a frequency distribution over words and punctuation in the training examples. To create a negative testing example input, we randomly select a length n from the length distribution and then randomly select with replacement n words or punctuation characters. To create the negative testing output, we randomly select without replacement three words from the input.

[1] Full code and experimental data are available at http://ilp.doc.ic.ac.uk/datatransformation



(a) Input examples

(b) Output examples

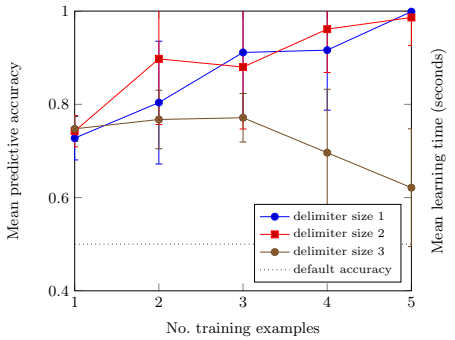
```
find_species(A,B):-
   known_species(Species),
   find_sublist(A,B,Species).
known_species([L,o,r,i,c,e,r,a, ,p,i,l,i,c,o,r,n,i,s]).
known_species([H,a,r,p,a,l,u,s, ,r,u,f,i,p,e,s]).
```

(c) Partial background knowledge for the ecological experiment

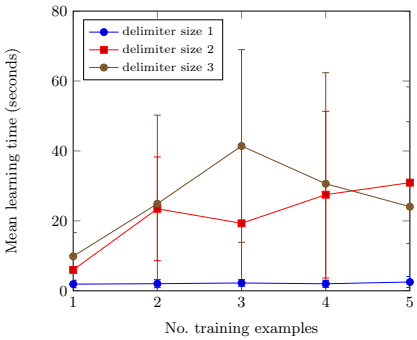
Fig. 3: Ecological input (a) and output (b) examples with the background knowledge (c)

Methods For each m in the interval [1,5], we randomly select without replacement m positive training examples, 20 positive testing examples, and 20 negative testing examples. We learn using the positive training examples only, but we test with both positive and negative examples, and thus the default predictive accuracy is 50%. We average predictive accuracies and learning times over 20 trials. We repeat the experiment for delimiter sizes 1, 2, and 3.

Results Fig. 4 shows that predictive accuracies improve with an increasing number of training examples, with over 80% predictive accuracy from two examples. In all cases, the predictive accuracies of learned solutions are greater than the default accuracy, and thus the null hypothesis is rejected. The results show that in this scenario there is no significant difference in predictive accuracy between delimiter sizes 1 and 2. For a delimiter size of 3, however, the predictive accuracy tails off because Metagol could not learn sufficiently discriminative delimiters under the size constraints. Fig. 4c show an example learned solution.



(a) Predictive accuracies



(b) Learning times

```
f(A,B):- f3(A,C), find_species(C,B).
f3(A,B):- find_species(A,C), f2(C,B).
f2(A,B):- closed_interval(A,B,[f,o],[o,d]).
f3(A,B):- find_species(A,C), f1(C,B).
f1(A,B):- closed_interval(A,B,[e,a],[t,s]).
```

(c) Learned program

Fig. 4: Ecological learning results when varying number of training examples

5.2 Patient medical records

In this experiment, the aim is to learn programs to extract values from patient medical records.

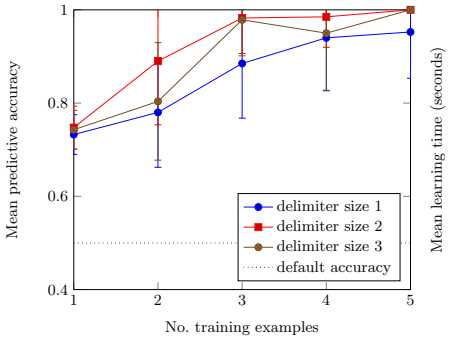
Materials The dataset contains 16 positive patient medical records, modelled on real-world examples[2], paired with a list of values to be extracted. Figs. 1a and 1b show simplified input/output example

[2] http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE8581

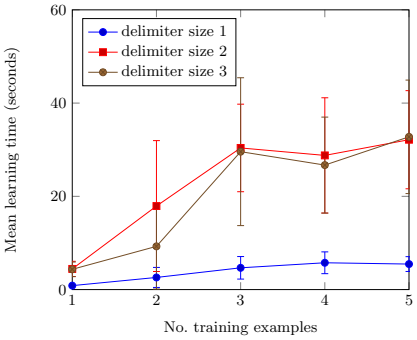
pairings. The experimental dataset, however, contains one additional input and output value, which is a floating integer value. We provide the following background predicates: find_int/2, find_float/2, and find_patient_id/2. We do not, however, provide a predicate to identify the diagnosis field, so Metagol must use the general purpose background predicates, described in Section 4, to learn a solution. To generate negative testing examples, we use an almost identical approach to the ecological experiment, and details are omitted for brevity.

Methods The methods are almost identical to ecological experiment, but we use 40 testing examples, half positive and half negative.

Results Fig. 5 shows that predictive accuracies improve with an increasing number of training examples, with over 80% predictive accuracy from a single example. In all cases, the predictive accuracies of learned solutions are greater than the default accuracy, and thus the null hypothesis is rejected. The results suggest a difference in predictive accuracy between delimiter sizes 1, 2, and 3, with a delimiter size of two achieving the highest predictive accuracy. Fig. 5c shows an example solution for a delimiter size of 2.



(a) Predictive accuracies



(b) Learning times

```
f(A,B):- f2(A,C), f2(C,B).
f2(A,B):- find_patient_id(A,C), find_int(C,B).
f2(A,B):- f1(A,C), find_float(C,B).
f1(A,B):- open_interval(A,B,[':',' '],[';'n']).
f1(A,B):- open_interval(A,B,[':',' '],[',',' ']).
```

(c) Learned program

Fig. 5: Medical record learning results when varying number of training examples

6 Conclusion and further work

We have investigated learning programs which transform data from one format to another. A general framework for the problem is presented and experiments conducted on ecological and medical data sets indicate that MIL is capable of generating accurate data transformation programs from small numbers of user-provided examples.

Further work This paper provides an initial investigation into an important new group of ILP applications relevant to real-world big data problems. Further work remains to achieve this potential. In particular, in an extended paper we hope to perform comparisons with alternative ILP and other machine learning techniques on this problem.

Although recursion and predicate invention appear important for data transformation problems, it is unclear how critical these are in either of the data sets studied. We would therefore like to study a broader set of datasets to understand this issue better.

Several issues need to be studied further in scaling-up the work reported. To begin with, although training data required will be small, owing to the requirements of user-provided annotation, test data

will typically consist of large numbers of instances. Running Prolog hypotheses on such data will be time-consuming, and we would like to investigate generating hypotheses in a scripting language, such as Python.

For data transformation problems such as the ecological data set we would also like to investigate the value of large scale background knowledge, which might provide deeper natural language analysis based on dictionaries, tokenisation, part-of-speech tagging, and specialised ontologies.

We would also like to investigate Probabilistic ILP approaches [12], such as [8], which have the potential to not only provide probabilistic preferences over hypothesised programs, but also the potential of dealing with issues such as noise which are ubiquitous within real-world data. In the context of free-text data these approaches might also be integrated with finding highest probability parses.

References

1. James S Aitken. Learning information extraction rules: An inductive logic programming approach. In *ECAI*, pages 355–359, 2002.
2. Mark Craven, Johan Kumlien, et al. Constructing biological knowledge bases by extracting information from text sources. In *ISMB*, volume 1999, pages 77–86, 1999.
3. A. Cropper and S.H. Muggleton. Learning efficient logical robot strategies involving composable objects. In *Proceedings of the 24th International Joint Conference Artificial Intelligence (IJCAI 2015)*, 2015. In press.
4. A. Cropper and S.H. Muggleton. Logical minimisation of meta-rules within meta-interpretive learning. In *Proceedings of the 24th International Conference on Inductive Logic Programming*. Springer-Verlag, 2015. In press.
5. Mark Goadrich, Louis Oliphant, and Jude Shavlik. Learning ensembles of first-order clauses for recall-precision curves: A case study in biomedical information extraction. In *Inductive logic programming*, pages 98–115. Springer, 2004.
6. Vu Le and Sumit Gulwani. Flashextract: A framework for data extraction by examples. In *ACM SIGPLAN Notices*, volume 49, pages 542–553. ACM, 2014.
7. D. Lin, E. Dechter, K. Ellis, J.B. Tenenbaum, and S.H. Muggleton. Bias reformulation for one-shot function induction. In *Proceedings of the 23rd European Conference on Artificial Intelligence (ECAI 2014)*, pages 525–530, Amsterdam, 2014. IOS Press.
8. S.H. Muggleton, D. Lin, J. Chen, and A. Tamaddoni-Nezhad. Metabayes: Bayesian meta-interpretative learning using higher-order stochastic refinement. In Gerson Zaverucha, Vitor Santos Costa, and Aline Marins Paes, editors, *Proceedings of the 23rd International Conference on Inductive Logic Programming (ILP 2013)*, pages 1–17, Berlin, 2014. Springer-Verlag. LNAI 8812.
9. S.H. Muggleton, D. Lin, N. Pahlavi, and A. Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. *Machine Learning*, 94:25–49, 2014.
10. S.H. Muggleton, D. Lin, and A. Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Machine Learning*, 2015. Published online: DOI 10.1007/s10994-014-5471-y.
11. J.R. Quinlan and R.M Cameron-Jones. FOIL: a midterm report. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 3–20. Springer-Verlag, 1993.
12. L. De Raedt, P. Frasconi, K. Kersting, and S.H. Muggleton, editors. *Probabilistic Inductive Logic Programming*. Springer-Verlag, Berlin, 2008. LNAI 4911.
13. Ashwin Srinivasan. *The Aleph Manual*. University of Oxford, 2007.
14. KD Sunderland. The diet of some predatory arthropods in cereal crops. *Journal of Applied Ecology*, 12(2):507–515, 1975.
15. Bo Wu and Craig A. Knoblock. An iterative approach to synthesize data transformation programs. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.