Notes on inductive logic programming methods in natural language processing (European work)

James Cussens
University of York

September 21, 1998

1 Introduction

The aim of these notes is to analyse ILP methods which have been applied to NLP, drawing exclusively on work conducted in Europe. The paper is organised thematically, examining how the following are treated in existing work:

1. Representation
2. Using expert knowledge
3. Hybrid approaches
4. Recursion
5. Large data sets

As a consequence of examining particular themes we examine a number of pieces of work. Appropriately it is in Section 2 on representation, where the bulk of the description of the various pieces of research is to be found. The interested reader should read the cited papers for the full story. Email addresses for all researchers are also given at the end.

This document arose from notes made in preparation for a tutorial on European ILP work on NLP given at the ILP tutorial day which took place immediately before ILP'98 in Madison, Wisconsin, USA. It does not cover American work in this area—this was handled by Ray Mooney at the tutorial day. (Go to `http://www.cs.utexas.edu/users/ml/` to find a number of ILP+NLP papers from Mooney's group.) Also it is narrowly focussed on ILP, and does not attempt a discussion of related work involving logic and/or machine learning in an NLP context.

2 Representation

Attribute-value ML approaches rest on the assumption that items of data can be adequately represented by fixed-length vectors of attribute vectors. Consequently, ILP is well-motivated where data can not, or can not easily, be represented as such. This is often the case in NLP.

2.1 Conceptual clustering

In [8], one goal is to cluster words into concepts and so to form an ontology of a domain. ASIUM, the tool used, begins by clustering words into "basic classes" to give a first level of clusters. ASIUM continues by building second level clusters from these, then third level clusters from the second level and so on. The final output is an acyclic directed graph (not necessarily a tree) representing a generality hierarchy between concepts.

Non-ILP clustering algorithms such as COBWEB or AUTOCLASS were rejected:

> Attributes of the input vector would be head words and values, their frequencies. As attributes would need to be the same in all vectors, very large vectors representing a whole dictionary would be required (about 2000 words in our experimentation) and most of their values would be equal to zero.

Here we see the "exploding attribute space phenomenon" which occurs when unsuitable data is crow-barred into attribute value form.

Instead, Faure and Nédellec use *instantiated subcategorization frames* as input to ASIUM. These have the general form:

```
<verb> <preposition | syntactical role: head word>*
```

and here are a couple of examples (produced originally from the clauses "My father travels by car" and "His neighbour travels by car"):

```
<to travel> <subject: father> <by: car>
<to travel> <subject: neighbour> <by: car>
```

ASIUMS's first step is to create a *synthetic frame* for each verb. For example, if the two examples above were our total training data, we would get this synthetic frame:

```
<to travel> <subject: [father(1), train(1)]> <by: [car(2)]>
```

Clearly, both instantiated subcategorization frames and synthetic frames are essentially terms, and could be represented, respectively, in Prolog like this:

```
travel([subject(father),by(car)])
travel([subject([father(1),neighbour(1)]),by([car(2)])]
```

2.2 Transfer rule learning

In [2] the goal is learn *transfer rules* which translate, for example, English *quasi-logical forms* (QLFs) to French or Swedish QLFs and vice-versa. QLFs are semantic representations of sentences or phrases which are represented as complex Prolog terms. As Boström and Zemke point out, transfer rule learning is a particular case of term rewriting. Here are two examples of transfer rules between English and Swedish QLFs:

```
trans(flight_AirplaneTrip,flygning_Flygresa).

trans([and,X1,form(_,verb(no,A,yes,M,D),V,Y1,_)],
      [and,X2,[island,form(_,verb(pres,A,no,M,D),V,Y2,_)]]) :-
              trans(X1,X2),
              trans(Y1,Y2).
```

In this application a logic-based representation seems inescapable since QLFs themselves are of this form. However, in [13] Milward and Pulman examine "flattening" QLFs into "Minimal Recursion Structures" the better to get at subterms buried deep within the QLF. The basic idea is to replace one complex term with a list of subterms which are indexed to encode the original structure. For example $f(g(a))$ would become $[0 : f(1), 1 : g(2), 2 : a]$. This flatter representation is superficially closer to an attribute-value representation but is not that close since (i) there is not a *fixed* number of attributes (hence a list) and (ii) the value of one 'attribute' can include the name of another.

2.3 Tagging

Tagging involves classifying individual words. In part-of-speech tagging words are tagged with their part-of-speech such as "singular common noun" or "3rd person present tense verb". More complex tags are also used. For example, in [11], Eineborg and Lindberg use (a pre-release of) the Stockholm-Umeå Corpus (SUC) where words are tagged with part-of-speech and a set of morphological features.

Tagging is non-trivial, since many words are ambiguous. For example, "table" can be a noun or a verb, although the latter reading is rare. To overcome ambiguity taggers use the context surrounding the focus word to tag it correctly. Here we examine two ways of representing context in ILP approaches to learning taggers from pre-tagged data.

2.3.1 Using a grammar

In [3, 4], the left and right contexts are represented as two lists of tags, where the list for the left context has the order of the tags reversed, so that the head of the list is the tag of the word immediately to the left of the focus word. Using lists instead of attribute-value vectors is more natural since contexts are not of a fixed length. The approach taken in [3, 4] rests on the assumption that, in many cases, the contexts, as represented by tags, are sufficient to determine that some potential tag for the focus word is in fact not the correct one. Taking each tag in turn, the ILP system Progol was used to find those contexts which allowed elimination of the tag. Contexts were characterised using a (very!) rough hand-crafted grammar. For example, adjectival phrase was defined as follows:

```
adj([jj|S],S) :- !.
adj([jjr|S],S) :- !.
adj([jjs|S],S).

adjp_rest([cma|L4],L2) :- !, adjp(L4,L2).
adjp_rest([cc|L4],L2) :- !, adjp(L4,L2).
adjp_rest(L3,L2) :- adjp(L3,L2).

adjp(L1,L2) :- adj(L1,X), !,  (X=L2 ; adjp_rest(X,L2)).
adjp(L1,L2) :- advp(L1,L3), adjp(L3,L2).
```

`jj`, `jjr` and `jjs` are the tags for normal, comparative and superlative adjectives. So, for example, any sequence of adjective-tagged words with, optionally, commas or conjunctions between them, counts as an adjectival phrase. Such predicates were used to define elimination rules. For example, the following states that a word can not be tagged as a conjunction (`cc`) if it is followed by a noun, which is followed by a verb phrase which is followed by an adjectival phrase, where verb phrase and adjectival phrase are defined in the background grammar.

```
rmv(A,B,cc) :- noun(B,C), vp(C,D), adjp(D,E).
```

Returning to the issue of representation we find that the combination of recursive data structures (the lists), intensional definitions and variable chains makes it:

1. easy for the user to define those features (tag sequences) which he/she judges likely to be relevant
2. easy for the tool (here Progol) to bolt these features together to form rule bodies

An advantage of representing, say, adjectival phrases intensionally is that their presence is checked for ‘on the fly’. Essentially we are doing ‘lazy feature construction’. Features, defined by connected literals, are only constructed at a particular point in the search if they might be useful in discriminating between positive and negatives. In the vast majority of cases Progol never examines tags far away from the focus word even if the declarative bias allows it. A relevance ordering which focuses on tags near the focus word drops out as a natural consequence of Progol’s top-down search. A Progol-specific disadvantage is that, unless some sort of caching is done, considerable re-computation and reconstruction of features may be done

2.3.2 Pulling out features

In [7, 11], Lindberg and Eineborg also learn tag elimination rules using Progol. Their approach differs from that of Cussens in that

1. words, as well as tags, are represented in the context
2. no grammar is used
3. the context is restricted to a window of length 5
4. the tags include morphological features as well as parts-of-speech; these features are used for eliminating tags

Here is an example of an induced rule:

```
remove(vb,A) :-
   context(A,left_target(word(att),featlist([imp,akt]))).
```

which says that focus (target) words can not have verb (`vb`) tags if they have the `imp` (imperative) and `akt` (active voice) features and the word to the left is "att". So instead of partially parsing the contexts, Lindberg and Eineborg use a fixed number of function symbols such as `left/1` and `right_right/2` which the single background predicate `context/2` uses to access and examine tagged words in the context. Each clause thus has only a single body literal at the expense of a complex term inside that literal.

2.3.3 Using a database

In [5] the pre-tagged Wall Street Journal corpus is represented by a single relation in an Oracle7^{TM} database. Each record has the following fields

1. Sentence ID – natural number
2. Word ID – natural number
3. Word – string
4. Tag – string

so the record $(77, 1, champagne, nn)$ represents that the first word in the 77th sentence is "champagne" and is tagged as "nn" (singular common noun).

Given the size of NL corpora and the close relations between logic programming and relational databases the use of database technology is well motivated. Similarly to Cussens, an elementary grammar was used as background knowledge to build constituents over tag sequences. The WARMR algorithm was used to find all 783 constituents sequences that occur at least 500 times in a 100000 word corpus—this took five days. For example, the sequence

```
np(WID,B), vp(B,C), np(C,D), prep(D,E), np(E,F)
```

was found 644 times. These constituents were constructed with a view to later treating them as binary features in a non-ILP approach to inducing a tagger.

From a representational point-of-view it is interesting that, due to the use of the database, lists have been rejected in favour of indexing sentences and words. On the plus side this allows the use of the database, on the minus side the size of the data is increased.

2.4 Morphology

ILP approaches to learning morphology can be found in [6, 12, 9]. In all of these, words are represented as lists of letters and a single list-processing predicate is used to perform morphological analysis.

In [6], inflectional paradigms of Slovene nouns are learned. From examples such as:

```
nxmsg([t,e,s,l,a],[t,e,s,l,e]).
```

rules such as:

```
nxsmg(A,B) :- split(A,C,[a]), split(B,C,[e]).
```

are learned where `split(A,B,C)` splits `A` into two non-empty lists `B` and `C`. `nxsmg/2` takes a base form ("lemma") as input and produces ("synthesises") the masculine genitive form as output. "Analysis" rules which produce the lemma from the masculine genitive form were also induced. Rules were learned for a large number of forms, not just masculine genitive. The work in [12] has a similar representation design except the single predicate `mate/6` is used where `mate(W1,W2,P1,P2,S1,S2)` is true if `P1` and `S1` is a prefix and suffix of `W1` and similarly for `P2`, `21` and `W2`. Similarly in [9], the single background predicate `append/3` is used to learn segmentation rules such as

```
    seg(A,B) :-
           append([é,b,l,o,u,i,s,s],B,A),
           append(C,[o,n,s],A),!.
```

from examples such as:

```
        seg([é,b,l,o,u,i,s,s,o,n,s],[o,n,s]).
```

The representational features here are the use of lists and list processing instead of fixed-length vectors and the appropriately highly limited use of background knowledge.

2.5 Learning phonetic rules

In [1], vowel recognition rules are learned from features of local maxima of the spectral decomposition of the wave signals produced by a speaker. The local maxima are found using a GA and then used as input to the IMPUT ILP algorithm.

Each local maximum is represented by a 6-tuple: size, frequency, frequency fluctuation, left angle, right angle, intensity. The 'angles' measure the left and right steepness of the 'hill' leading up to the maxima. Each vowel sound could then be represented by a list of these tuples representing the sequence of maxima produced. Here is an example:

```
sound([
   (fulllength,125,veryveryshort,85,110,loud),
   (fulllength,2350,veryshort,75,110,noise),
   (fulllength,3550,wide,75,120,noise)
]).
```

For a given vowel, samples of that vowel were used as positive examples and all of the other samples were used as negatives. The IMPUT system was then used to refine an initially over-general program. IMPUT uses unfolding and clause removal to refine an overly general program. An interactive debugger is used to find the best place for unfolding. Here we see a mixture of fixed-length and non-fixed-length representation: the local maxima are represented by 6 features, the sounds are represented by a list of local maxima.

2.6 The Logic, Language and Learning (LLL) challenge

The LLL challenge [10] is a learning problem set up by the ongoing ILP2 project with assistance from SRI International. Competitors are given *part* of a unification grammar and lexicon developed during the FraCaS project (see `http://www/cogsci.ed.ac.uk/~fracas/deliverables.html` for details). They are also given examples of a predicate `parse/2` which maps input sentences to the QLFs produced from the *complete* grammar. The task is to find the missing grammar rules and lexical items.

Here is the lexical entry for "need" which expresses syntactic and semantic information for that word in a form which allows efficient parsing rather than human comprehensibility:

```
cmp_synword(need, v(f(0,0,1,1,1,1,1,1,1),
        sem(need(A,B,C),A,B,C,_,pos),
        n,
        n,
        [np(_,_,_,f(0,_,1,1),nonsubj)])).
```

Grammar rules are also unit clauses of the form: `cmp_synrule(RuleID,LHS,RHS).` Again we see the use of complex terms to represent linguistic information. No results from this data are currently available since it has only just been released.

2.7 Summary of representational issues

With the exception of transfer rule learning, the structure of induced theories and clauses are remarkably simple, although the number of clauses in an induced theory can be quite large. It is principally in the representation of possibly quite complex terms that the logical representation is being exploited. Even in the case of transfer rule learning, where recursion is central, the induced clauses follow the term structure of the QLFs in a very direct manner. Also a successful response to the LLL challenge will comprise unit clauses whose complex term structure will represent the lexical entries and grammar rules.

This fits in well with lexical approaches. Consider categorial grammar which consists of a highly structured lexicon and only two grammar rules. Also, pushing the representational work into the terms makes sense computationally. We have fewer more complicated unifications (compare appending lists and difference lists).

3 Using expert knowledge

Failing to exploit existing grammatical expertise would clearly be wasteful. All ILP approaches incorporate expert knowledge to some degree via the choice of background predicates. In some ILP/NLP approaches more interaction with the expert is enabled.

This is most notable in the ASIUM system. "Clustering steps are intertwined with co-operative validation steps where a domain expert assesses and refines the learning results" [8]. The user also names the induced clusters. This interaction is facilitated by a graphical user interface.

In other cases the goal is to induce a theory to complete what an expert has provided already. In the case of transfer rule learning, the experts provided the non-recursive transfer rules and the job was to build the recursive ones. A previous alternative approach was to hand-craft non-recursive rules by inspecting the data.

The LLL challenge is an exercise is grammar completion. Although the LLL challenge is artificial, the problem of grammar completion is very real given the continued difficulty in producing wide-coverage grammars purely by hand. Being able to build on an existing hand-crafted grammar (perhaps in co-operation with an expert) to achieve wider coverage is a very important application area for ILP.

4 Hybrid approaches

Given that ILP is not a 'silver bullet' it is often used in conjunction with other methods. Leaving aside the non-trivial task of pre-processing data to get it into Prolog form, we see a number of hybrid approaches:

- The ASIUM system takes the output of a syntactic parser (SYLEX) to provide its basic input.
- In [3] lexical statistics are used to resolve ambiguities left pending by the induced contextual rules.

- In [4] induced rules are parameterised and used to provide probabilities to a Gibbs sampling based tagger.
- In [9] a genetic algorithm is used to search for a (suitably defined) optimal segmentation of initially unsegmented words in the training data. These segmented words are then used as training data for the ILP algorithm CLOG, which looks for general segmentation rules using these examples of particular segmentations.

5 Recursion

Although ILP systems are unusual in their ability to induce recursive rules, this facility has very rarely been used in real-world applications. (The sole exception is an application in finite element mesh design[1].) It is worth noting that recursively defined *background predicates* have been used extensively in tagging, morphological and phonetic work to provide operations on lists of tags, words, letters, etc.

It seems likely that, with some applications to NLP, recursion will appear in induced clauses rather than just background clauses. This would necessarily be so when (directly) learning or completing recursively defined grammars. Learning recursive clauses is generally harder, but the work on transfer rule learning already provides an example of an application where the structure of the terms in the data can 'push' the search for a suitable recursive theory in the right direction.

6 Large data sets

The corpus-based tagging work uses data sets which are considerably larger than is usual in ILP. [5] used a 100,000 word corpus. Lindberg and Eineborg, as well as Cussens used training sets of up to 6000 examples. In all these cases training times were long: WARMR took 5 days on the 100,000 word corpus, in [3] 6000 examples usually took in the region of 20,000-30,000 seconds (albeit on a slow Sparc 5). Lindberg and Eineborg give one example of 493 rules being induced from 1284 positives and 419 negatives in 11,785 seconds.

If the results of ILP induction are valuable enough, then a wait of several days may be a small price to pay. However, it seems likely that corpus based ILP will focus effort on increasing the efficiency of ILP algorithms. This has already occurred to a limited extent; for example caching and a more efficient representation of examples in P-Progol were motivated by work on tagging. CLOG was developed to learn decision lists (for morphology) more efficiently than FOIDL. In the future it seems likely that there will be greater use of database techniques in ILP: the work with WARMR is an example of this.

[1]Luc de Raedt, personal communication