μ-TBL Lite: A Small, Extendible Transformation-Based Learner

Torbjörn Lager
Department of Linguistics
Uppsala University
SWEDEN
Torbjorn.Lager@ling.uu.se

Abstract

This short paper describes – and in fact gives the complete source for – a tiny Prolog program implementing a flexible and fairly efficient Transformation-Based Learning (TBL) system.

1 Introduction

Transformation-Based Learning (Brill, 1995) is a well-established learning method in NLP circles. This short paper presents a 'light' version of the **μ-TBL system** – a general, logically transparent, flexible and efficient transformation-based learner presented in (Lager, 1999). It turns out that a transformation-based learner, complete with a compiler for templates, can be implemented in less than one page of Prolog code.

2 μ-TBL Rules & Representations

The point of departure for TBL is a tagged *initial-state corpus* and a correctly tagged *training corpus*. Assuming the part-of-speech tagging task, corpus data can be represented by means of three kinds of clauses:

`wd(P,W)` is true iff the word `W` is at position `P` in the corpus

`tag(P,A)` is true iff the word at position `P` in the corpus is tagged `A`

`tag(A,B,P)` is true iff the word at `P` is tagged `A` and the correct tag for the word at `P` is `B`

Although this representation may seem a bit redundant, it provides exactly the kind of indexing into the data that is needed.[1] A decent Prolog system can deal with millions of such clauses.

[1]Assuming a Prolog with first argument indexing. The μ-TBL systems are implemented in SICStus Prolog.

The object of TBL is to learn an ordered sequence of *transformation rules*. Such rules dictate when – based on the context – a word should have its tag changed. An example would be "replace tag `vb` with `nn` if the word immediately to the left has a tag `dt`." Here is how this rule is represented in the μ-TBL rule/template formalism:

```
tag:vb>nn <- tag:dt@[-1].
```

Conditions may refer to different features, and complex conditions may be composed from simpler ones. For example, here is a rule saying "replace tag `rb` with `jj`, if the current word is "only", and if one of the previous two tags is `dt`.":

```
tag:rb>jj <- wd:only@[0] & tag:dt@[-1,-2].
```

Rules that can be learned in TBL are instances of *templates*, such as "replace tag `A` with `B` if the word immediately to the left has tag `C`", where `A`, `B` and `C` are variables. In the μ-TBL formalism:

```
t3(A,B,C) # tag:A>B <- tag:C@[-1].
```

Positive and negative instances of rules that are instances of this template can be generated by means of the following clauses:

```
pos(t3(A,B,C)) :-
    dif(A,B),tag(A,B,P),P1 is P-1,tag(P1,C).

neg(t3(A,B,C)) :-
    tag(A,A,P),P1 is P-1,tag(P1,C).
```

Tied to each template is also a procedure that will apply rules that are instances of the template:

```
app(t3(A,B,C)) :-
    (tag(A,X,P), P1 is P-1, tag(P1,C),
    retract(tag(A,X,P)), retract(tag(P,A)),
    assert(tag(B,X,P)), assert(tag(P,B)),
    fail ; true).
```

3 The μ-TBL Template Compiler

To write clauses such as the above by hand for large sets of templates would be tedious and prone to errors. Instead, Prolog's term expansion facility, and a couple of DCG rules, can be used to compile templates into Prolog code, as follows:

```
term_expansion((ID # A<-Cs),
        [(pos(ID) :- G1),
         (neg(ID) :- G2),
         (app(ID) :- (G3,fail;true))]) :-
    pos((A<-Cs),L1,[]), list2goal(L1,G1),
    neg((A<-Cs),L2,[]), list2goal(L2,G2),
    app((A<-Cs),L3,[]), list2goal(L3,G3).

pos((F:A>B<-Cs)) -->
    {G =.. [F,A,B,P]},[dif(A,B),G], cond(Cs,P).

neg((F:A>_<-Cs)) -->
    {G =.. [F,A,A,P]}, [G], cond(Cs,P).

app((F:A>B<-Cs)) -->
    {G1 =.. [F,A,X,P], G2 =.. [F,P,A],
     G3 =.. [F,B,X,P], G4 =.. [F,P,B]},
    [G1], cond(Cs,P), [retract(G1),
    retract(G2), assert(G3), assert(G4)].

cond((C&Cs),P) --> cond(C,P), cond(Cs,P).
cond(FA@Pos,P0) --> pos(Pos,P0,P), feat(FA,P).

pos(Pos,P0,P) -->
    [member(Offset,Pos), P is P0+Offset].

feat(F:A,P) --> {G =.. [F,P,A]}, [G].
```

4 The μ-TBL Lite Learner

Given corpus data, compiled templates, and a value for Threshold, the predicate tbl/1 implements the μ-TBL main loop, and writes a sequence of rules to the screen:

```
tbl(Threshold) :-
    (   setof(N-Rule,L^(bagof(.,pos(Rule),L),
            length(L,N), N >= Threshold),FL),
        reverse(FL,RevFL),
        bestof(RevFL,dummy,Threshold,Winner),
        dif(Winner,dummy)
    ->  write(Winner),nl,
        app(Winner),
        tbl(Threshold)
    ;   true
    ).
```

The call to the setof-bagof combination generates a frequency listing of all positive instances of all templates, based on which the call to bestof/4 then selects the rule with the highest score. tbl/1 terminates if the score for that rule is less than the threshold, else it applies the rule and goes on to learn more rules from there.

```
bestof(FL0,Leader,HiScore,Winner) :-
    (   FL0 = [Pos-Rule|FL],
        Pos > HiScore
    ->  Max is Pos-HiScore,
        (   count0(neg(Rule),Max,Neg)
        ->  bestof(FL,Rule,Pos-Neg,Winner)
        ;   bestof(FL,Leader,HiScore,Winner)
        )
    ;   Winner = Leader
    ).
```

To compute the rule with the highest score, bestof/4 traverses the frequency listing, keeping track of a leading rule and its score. The score of a rule is calculated as the difference between the number of its positive instances and its negative instances. When the list of rules is empty or the number of positive instances of the most frequent rule in what remains of the list is less than the leading rules score, the leader is declared winner.

The following procedure implements the counting of negative instances in an efficient way:

```
count0(G,M,N) :-
    (   bb_put(c,0), G, bb_get(c,N0),
        N is N0+1, bb_put(c,N), N > M
    ->  fail
    ;   bb_get(c,N)
    ).
```

5 μ-TBL Lite Performance

The learner was benchmarked on a 250Mhz Sun Ultra Enterprise 3000, training on Swedish corpora of three different sizes, with 23 different tags, and the 26 templates that Brill uses in his context-rule learner[2]. In each case, the accuracy of the resulting sequence of rules was measured on a test corpus consisting of 40k words, with an initial-state accuracy of 93.3%. The following table summarizes the results:

Size	Thrshld	Runtime	# of rules	Acc.
30k	2	15 min	99	95.5%
60k	4	24 min	85	95.7%
120k	6	60 min	92	95.8%

By comparison, it took Brill's C-implemented context-rule learner 90 minutes, 185 minutes, and 560 minutes, respectively, to train on these corpora, producing similar sequences of rules. Thus μ-TBL Lite is an order of magnitude faster than Brill's learner. The full μ-TBL system presented in (Lager, 1999) is even faster, uses less memory, and is in certain respects more general. Small is beautiful, however, and the light version may also have a greater pedagogical value. Both versions can be downloaded from http://www.ling.gu.se/~lager/mutbl.html.

References

Lager, Torbjörn. 1999. The μ-TBL System: Logic Programming Tools for Transformation-Based Learning, In: *Proceedings of CoNLL-99*, Bergen.

Brill, Eric. 1995. Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics*, December 1995.

[2]Available from http://www.cs.jhu.edu/~brill.