TBL Template Selection: An Evolutionary Approach

Ruy Luiz Milidiú[1], Julio Cesar Duarte[2], and Cícero Nogueira dos Santos[1]

[1] Departamento de Informática, Pontifícia Universidade Católica, Rio de Janeiro, Brazil
{milidiu,nogueira}@inf.puc-rio.br
[2] Centro Tecnológico do Exército, Rio de Janeiro, Brazil
jduarte@ctex.eb.br

Abstract. Transformation Based Learning (TBL) is an intensively Machine Learning algorithm frequently used in Natural Language Processing. TBL uses rule templates to identify error-correcting patterns. A critical requirement in TBL is the availability of a problem domain expert to build these rule templates. In this work, we propose an evolutionary approach based on Genetic Algorithms to automatically implement the template selection process. We show some empirical evidence that our approach provides template sets with almost the same quality as human built templates.

1 Introduction

Transformation Based error-driven Learning (TBL) is a symbolic machine learning method introduced by Eric Brill [1]. The TBL technique builds an ordered set of rules that correct mistakes of a base line classifier. It has been used for several important linguistic tasks, such as part-of-speech (POS) tagging [1], parsing, prepositional phrase attachment [2] and phrase chunking [3,4], having achieved state-of-the-art performance in many of them.

Within the TBL framework, the generated rules must follow patterns called templates, which are meant to capture the relevant feature combinations. The accuracy of the TBL classifier is highly dependent on the template set used in the learning process. Unfortunately, the process of generating *good* templates is highly expensive and depends on the problem expert skills.

In this work, we address the problem of automatic TBL template selection through an evolutionary approach based on Genetic Algorithms (GAs). We show four genetic approaches, each one with a different degree of understanding of the problem. The better the understanding, the better is the accuracy of the generated classifier. Our experiments show that we can achieve the same quality as the best template set for some benchmark problems.

The remainder of this paper is organized as follows. Section 2 presents a brief overview of GAs and TBL. In Section 3, we describe our genetic approaches. Section 4 presents our experimental results. In the final section, we make some conclusions.

2 Techniques

2.1 Genetic Algorithms

Genetic Algorithms (GAs) [5] are a family of computational models inspired in the mechanisms of Evolution and Natural Selection. They model the solution of the problem into a data structure called **chromosome**, or *genotype* or *genome*, which represents the possible solutions, called **individuals**, or *creatures* or *phenotypes*. A series of genetic operators are applied to these chromosomes in order to achieve a high optimization of the problem.

Two components play an important role in the GA method: the problem codification and the evaluation function. The problem codification is the mapping that is made between the chromosomes and the individuals. Usually, the individuals are mapped into a string of 1's and 0's indicating the presence, or not, of some feature or characteristic. The evaluation function takes one individual and calculates its fitness. Usually, the fitness is a performance measure of the individual as a solution to the problem.

Normally, a genetic algorithm starts with a random population of individuals, which is influenced by the genetic operators over the generations. The main objective of a generation is to keep the *best* individuals, enhancing the overall fitness of the population, until some stopping criteria is achieved.

There are two kinds of genetic operators: selection and recombination. Selection operators use the evaluation function to decide which individuals have the highest potential. These individuals should persist in the population and be used by the other kind of operators.

The recombination operators are used to create new individuals using one or more high potential individuals. The most famous operators in this class are cross-over and mutation. The cross-over operator uses two or more fractions of high potential individuals to build a new individual which is appended to the next generation of the population. The mutation operator, on other hand, takes one high potential individual and makes a slight change in one of its components. The new individual is also appended in the next generation of the population.

2.2 Transformation Based Learning

Transformation Based error-driven Learning (TBL) uses a greedy error correcting strategy. Its main propose is to generate an ordered list of rules that correct classification mistakes in the training set, which have been produced by an initial classifier.

The requirements of the TBL algorithm are: a training corpus, a template set, an initial classifier and a score threshold. The learning method is a mistake-driven greedy procedure that iteratively acquires a set of transformation rules from the template set maximizing its score. The score from a rule can be defined as the number of corrections that it achieves in the training corpus in some iteration of the learning process, discounting the number of mistakes it makes in the same corpus. At each iteration, the rule with best score (better than the threshold) is chosen to be used in the generated classifier. The threshold value can be tuned to avoid overfitting to the training corpus. The classification process of a new sample can be done by simply applying the baseline classifier BC and the ordered rule set R. The pseudo-code of the TBL algorithm is shown in Algorithm 1

Algorithm 1. The TBL Algorithm Pseudo-Code

```
input  A training corpus $C_0$, a template set $T$, a baseline classifier $BC$ and an integer
  threshold $\tau$
  Apply $BC$ to $C_0$ generating $C_1$
  $R \leftarrow \{\}$
  $k \leftarrow 1$
  repeat
    Generate $CR_k$, instantiating all candidate rules from $T$ using $C_k$,
    for all r such that $r \in CR_k$ do
      score(r) $\leftarrow$ #(good corrections of r) - #(bad corrections of r) in $C_k$
    end for
    Choose $r_M$ from $CR_k$ with highest positive score above $\tau$
    if $r_M$ exists then
      Apply $r_M$ to $C_k$ generating $C_{k+1}$
      $R \leftarrow R + r_M$.
    end if
    $k \leftarrow k + 1$
  until not $r_M$ exists
output  $R$
```

TBL Templates. A TBL template can be any sequence of patterns that generates an error correction rule. For instance, in a Part-Of-Speech(POS) tagging process, we can write a template like *word[0] word[-1] pos[0]*, which tries to make rules based on bi-grams, correcting the current POS tag based on the current and previous words.

We define a template as being a sequence of *Atomic Terms* (ATs). An AT is the smallest template unit which indicates the feature and conditions to be instantiated in a template. It is meant to identify one peace of the context that a TBL rule needs to test when applying to the target token. Some examples of ATs are:

1. **f[ds]**, which checks the feature *f* of a token, located *ds* tokens to the left or right (depending of the sign) of the target token. For example: word[-1];
2. **f[ds,de]**, which checks the feature *f* in an interval of tokens positioned between *ds* and *de* (included), in relation to the target token. For example: word[-1,1];
3. **f[ds,de]_where(f'=v')**, which checks the feature *f* of the token nearest to the target token, within the closed interval of *ds* and *de*, for which the feature *f'* equals *v'* [6]. For example: word[-1,-5]_where(pos=VBD).

More complex atomic terms can be defined in order to create more specialized rules.

3 Approaches

In this section, we show the genetic coding used in our experiments. The use of genetic algorithms in conjunction with TBL has already been examined in [7], where they are used in the TBL training process to generate the instantiated rules and to provide an adaptive ranking. Nevertheless, they have not been used in the evaluation of template sets what is our proposal. In all codings, the template ordering is not taking into account, since it is the last criteria to be used when two or more rules have the same score.

3.1 Genetic Coding

Fixed Context Window. In this approach, the *chromosome* is composed by several sequences of possible atomic terms (ATs) of the simplest form $f[ds]$. The value in the *chromosome* determines the presence or absence of the corresponding AT in the template. The input for this coding is composed by the following items: the list of possible features to be used, an integer value maxOffset, the number of templates to be generated and an expected number of atomic terms in each template. The generated templates are sequences of atomic terms of the form $f[ds]$, where $ds \in$ {-maxOffset, +maxOffset}. An example of this coding is given in Table 1, showing two templates with expected size 3, using 2 features, f_1 and f_2, and maxOffset equals to 1. The *chromosome* shown in the Table 1 generates the following two templates: f_1[-1] f_1[+1] f_2[-1] f_2[+1] and f_2[-1] f_2[0].

Table 1. Example of the Fixed Context Window Approach

	Template 1						Template 2					
	f_1[-1]	f_1[0]	f_1[+1]	f_2[-1]	f_2[0]	f_2[+1]	f_1[-1]	f_1[0]	f_1[+1]	f_2[-1]	f_2[0]	f_2[+1]
C_1	1	0	1	1	0	1	0	0	0	1	1	0

Fixed List of Atomic Terms. Usually, it is easier to identify candidate atomic terms by looking at the output errors of a Machine Learning Algorithm. In Fixed List of Atomic Terms, the *chromosome* is very similar to the previous one, but it can be composed by sequences of a given set of atomic terms. The *chromosome* value also indicates the presence or the absence of the corresponding atomic term in the template. The input for this coding is the list of possible atomic terms to be used, and, as well, the number of templates to be generated and the expected number of atomic terms. An example of this coding is given in Table 2, showing two templates with expected size 3, using 6 different possible atomic terms

Table 2. Example of the Fixed List of Atomic Terms Approach

	Template 1						Template 2					
	AT_0	AT_1	AT_2	AT_3	AT_4	AT_5	AT_0	AT_1	AT_2	AT_3	AT_4	AT_5
C_1	0	1	1	0	0	1	1	0	1	0	1	0

$f_1[-1]$, $f_1[-2]$, $f_2[0]$, $f_2[1]$, $f_1[0,2]$ and $f_2[-2,-0]\_where\{f_1 = v_1\}$. The *chromosome* shown in the Table 2 generates the following two templates: $f_1[-2]$ $f_2[0]$ $f_2[-2,-0]\_where\{f_1 = v_1\}$ and $f_1[-1]$ $f_2[0]$ $f_1[0,2]$.

Maximum Template Size. In this approach, the *chromosome* is quite similar to the previous one, but instead of having an expected template size we establish a maximum size for all templates. The *chromosome* value indicates the position of the corresponding atomic term in the list. A value -1 indicates the absence of an atomic term. The repetition of atomic terms in the same template is now a possibility, but they are discarded. The input for this coding is the list of possible atomic terms to be used, the number of templates to be generated and the maximum template size. An example of this coding is given in Table 3, showing three templates with maximum size 4, using the same six possible previous atomic terms. The *chromosome* shown in the Table 3 generates the following three templates: $f_1[-1]$ $f_1[-2]$ $f_2[1]$, $f_1[-2]$ $f_2[0]$ $f_2[1]$ $f_2[-2,-0]\_where\{f_1 = v_1\}$ and $f_1[-2]$ $f_2[0]$ $f_2[1]$.

Table 3. Example of the Maximum Template Size Approach

	Template 1				Template 2				Template 3			
	AT_1	AT_2	AT_3	AT_4	AT_1	AT_2	AT_3	AT_4	AT_1	AT_2	AT_3	AT_4
C_1	1	3	-1	0	5	1	3	2	1	2	1	3

Template List. In this approach, the *chromosome* is composed of a sequence of predefined templates. The idea here is to find a better subset of templates than the one provided by an expert. Since TBL is a greedy algorithm, using all templates may not lead to better results than using just one of its subsets. The *chromosome* value indicates the presence or absence of the corresponding template. The input for this coding is the list of possible templates to be used and the expected number of templates to be used. An example of this coding is given in Table 4, showing templates from the fixed template list, $\{\tau_{00}, \tau_{01}, \tau_{02}, \tau_{03}, \tau_{04}, \tau_{05}, \tau_{06}, \tau_{07}, \tau_{08}, \tau_{09}, \tau_{10}, \tau_{11}\}$, with an expected number of seven templates. The *chromosome* shown in the Table 4 generates the following template set: $\{\tau_{00}, \tau_{02}, \tau_{05}, \tau_{06}, \tau_{08}, \tau_{09}, \tau_{10}\}$.

Table 4. Example of the Template List Approach

	τ_{00}	τ_{01}	τ_{02}	τ_{03}	τ_{04}	τ_{05}	τ_{06}	τ_{07}	τ_{08}	τ_{09}	τ_{10}	τ_{11}
C_1	1	0	1	0	0	1	1	0	1	1	1	0

3.2 Fitness Function

Using a training set, we train a TBL classifier for each individual. The F-measure of the generated classifier for a validation set is used as the fitness value of the individual.

3.3 Cross-Over Operator

The cross-over operator generates a new *chromosome* by breaking apart two *chromosomes* in a random point and combining them. Table 5 shows an example of the cross-over operator for the *chromosome* described in the Fixed Context Window approach.

3.4 Mutation Operator

The mutation operator generates a new *chromosome* by changing the value of the atomic term in a template. Table 5 shows an example of the mutation process for the *chromosome* described in the Fixed Context Window approach.

Table 5. Examples of the Cross-over and Mutation operator

	Template 1							Template 2				
	f_1[-1]	f_1[0]	f_1[+1]	f_2[-1]	f_2[0]	f_2[+1]	f_1[-1]	f_1[0]	f_1[+1]	f_2[-1]	f_2[0]	f_2[+1]
C_1	**1**	**0**	**1**	**1**	**0**	**1**	**0**	0	0	*1*	1	0
C_2	1	1	0	0	0	1	1	**1**	**0**	**1**	**0**	**0**
$C_1 \otimes C_2$	1	0	1	1	0	1	0	1	0	1	0	0
$\odot C_1$	1	0	1	1	0	1	0	0	0	*0*	1	0

For the Maximum Template Size approach, instead of changing the value from 0 to 1 and vice-versa, the value is changed to another value in the interval [-1, number of atomic terms - 1].

4 Experiments

We have chosen the *English Base Noun Phrase Chunking* to demonstrate the quality of our genetic approaches. Base Noun Phrase Chunking consists in recognizing non-overlapping text segments that correspond to noun phrases (NPs).

The data used in the base NP chunking is the one of Ramshaw & Marcus [3]. This corpus contains sections 15-18 and section 20 of the Penn Treebank, and is pre-divided into a 8936-sentence (211727 tokens) training set and a 2012-sentence (47377 tokens) test set. This corpus is tagged with POS tags and with base NP tags.

A small excerpt of the training corpus is used by the genetic approach. Two corpora are built: a GA-training set and a validation set. The GA-training and validation sets are used by the genetic algorithm to, respectively, train and evaluate the performance of the individuals. The *best* individual returned by the genetic algorithm is applied to the whole training corpus, generating a TBL classifier. The classifier is, then, applied to the test corpus and its performance is evaluated.

We use F-measure as our key statistics to evaluate the performance of the generated classifiers. **F-measure** is the harmonic mean between precision and recall. **Precision** informs how many good classifications the model predicted

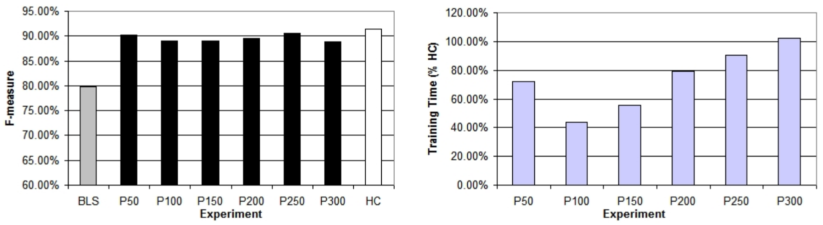

Fig. 1. Results for the Fixed Context Window approach

amongst all predictions made. **Recall** informs how many good classifications were predicted amongst all true classifications.

For the four genetic formulations, we report the performance of the classifier trained with the *best* template set produced by the use of different slices of the GA-training set in the genetic approach. These results are compared with the Baseline System (BLS), the same used by [3], and the handcrafted templates (HC). Although, we fixed the τ parameter used in all experiments to the same value used by the handcrafted templates, it could also be encoded and determined by the genetic algorithm without considerable loss of performance, since its set of optimal values is very limited (usually, $0 \leq \tau \leq 2$). We start with 50 sentences for the genetic training process, increasing with 50 more examples in each experiment. We also report the training time for each approach, in terms of percentage of the training time for the *handcrafted* templates. The reported training time includes both the selection of the best template set by the genetic algorithm and the training of the TBL classifier. The BLS training time is not reported since it is very small. Due to space constraints, we do not show the performance of the population in the validation set over the ten fixed generations, but it shows a consistent increase for all approaches.

The results for the Fixed Context Window (FCW) approach are reported in Figure 1. The experiment is conducted using the three possible features (word, POS and NP tag) with a window size of five ([-2, +2]). The genetic algorithm generated 20 templates with an expected atomic term size of 3. As we can see, the results are very good since we generate only 20 templates with the simplest atomic term. The loss of F-measure is smaller than 1% in the best ga-training sets. Also the genetic approaches takes less training time, since the templates are very simple.

Figure 2 shows the results for the Maximum Template Size (MTS) approach. The atomic term list used is {npt[0], npt[-1], npt[-2], npt[1], npt[2], pos[0], pos[1], pos[2], pos[-2], pos[-1], pos[$-3, -1$], pos[$1, 3$], word[0], word[1], word[2], word[-1], word[-2], word[$-3, -1$], word[$1, 3$]}. The results are almost the same. We do not use very complex atomic terms in order to maintain the simplicity of the approaches, avoiding the need of a specialist to determine the atomic term list. The genetic algorithm generated 20 templates with maximum atomic term size of 5. The overall training time is increased, since we added atomic terms that may instantiate more candidate rules.

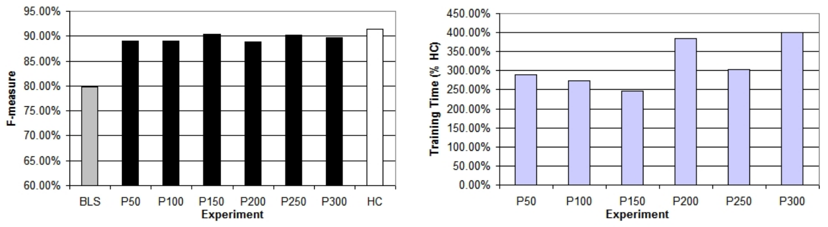

Fig. 2. Results for the Maximum Template Size approach

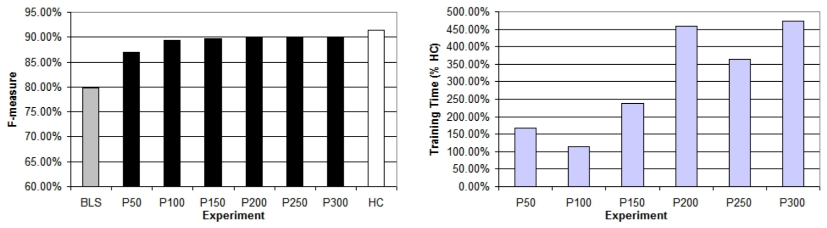

Fig. 3. Results for the Fixed List of Atomic Terms approach

The experiment using the Fixed List of Atomic Terms (FLAT) approach is quite similar to the previous one, with same main parameters, and is reported in Figure 3. The only difference is that we define the expected template size, which was fixed in 4. We can see that the results are very similar to the previous one, in terms of F-measure and training time, since the two approaches are quite equivalent.

The last conducted experiment uses the Template List (TL) approach. In this experiment, we try to find out a better combination of templates than the one provided by a specialist. Here, we use the template set proposed in [3]. The genetic generations are started with 80% of the templates activated. Figure 4 shows the results for this experiment. We can see that the template combination found by our approach achieve better results than the template set proposed by the specialist. However, this achievement implies in an increase of the overall training time.

We conducted other experiments with the English text chunking (CK) and Portuguese named entities (NE) tasks. The text chunking corpus is the same used in [3] and in the Base NP experiments, with the text chunking tags. The named entities corpus used is the same reported in [8]. The NE corpus was divided into a 1722-sentence (27055 tokens) training set and a 378-sentence (6084 tokens) test set. This corpus is tagged with POS tags and NE tags.

Due to space constraints, we show only the results of the best generated classifiers for each approach. The overall results in terms of F-measure and training time are similar to the ones reported for the base NP chunking. Figure 5 shows the results for the two experiments. The only aspect to except is that much more relative training time was needed in the NE problem since the TBL template designers managed to build very compact light templates with very short training

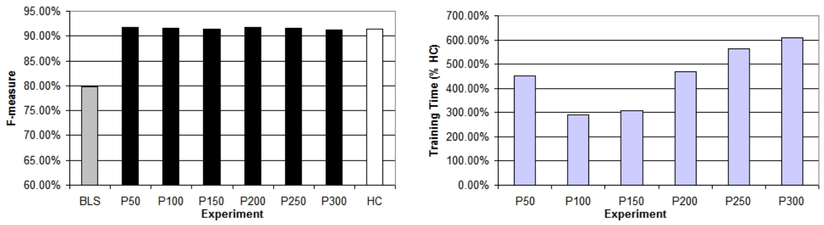

Fig. 4. Results for the Template List approach

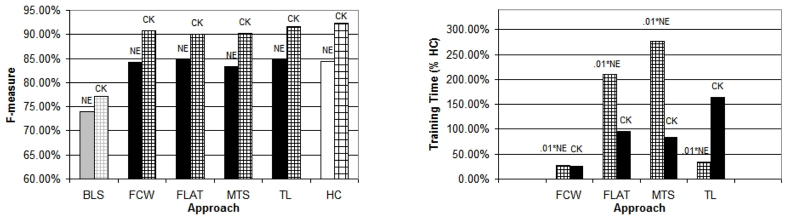

Fig. 5. Results for English Text Chunking and Portuguese Named Entities Extraction

times. That is why these relative training times are scaled by a factor of 1% in Figure 5.

5 Conclusions

TBL Template construction is a highly expensive process with strong impact in the classifier's accuracy. In this paper, we presented an evolutionary approach to help the creation of TBL templates. Our schemes use simple template design and very little training data to develop a set of templates.

We show a set of experiments that demonstrate the applicability and the effectiveness of the proposed method. The experimental results indicate that our approach achieves much better accuracy than the base line algorithm. Moreover, in many cases, our method slightly outperformed the F-measures obtained by the handcrafted templates with compatible training time since the domain expert was removed of most of the process.

References

1. Brill, E.: Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. Computational Linguistics 21, 543–565 (1995)
2. Brill, E., Resnik, P.: A rule-based approach to prepositional phrase attachment disambiguation. In: Proceedings of COLING 1994, Kyoto, Japan (1994)
3. Ramshaw, L., Marcus, M.: Text chunking using transformation-based learning. In: Yarovsky, D., Church, K. (eds.) Proceedings of the Third Workshop on Very Large Corpora, New Jersey, Association for Computational Linguistics, pp. 82–94 (1995)

4. Megyesi, B.: Shallow parsing with pos taggers and linguistic features. Journal of Machine Learning Research 2, 639–668 (2002)
5. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
6. dos Santos, C.N., Oliveira, C.: Constrained atomic term: Widening the reach of rule templates in transformation based learning. In: Bento, C., Cardoso, A., Dias, G. (eds.) EPIA 2005. LNCS (LNAI), vol. 3808, pp. 622–633. Springer, Heidelberg (2005)
7. Wilson, G., Heywood, M.: Use of a genetic algorithm in brill's transformation-based part-of-speech tagger. In: GECCO 2005. Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 2067–2073. ACM Press, New York (2005)
8. Milidiú, R.L., Duarte, J.C., Cavalcante, R.: Machine learning algorithms for portuguese named entity recognition. In: Fourth Workshop in Information and Human Language Technology (TIL 2006), Ribeirão Preto, Brazil (2006)