Rule Induction for Sentence Reduction

João Cordeiro[1], Gaël Dias[2], and Pavel Brazdil[3]

[1] University of Beira Interior, DI-Hultig,
Rua Marquês d'Ávila e Bolama, Covilhã 6200-001, Portugal,
jpaulo@di.ubi.pt

[2] University of Caen Basse-Normandie, GREYC Hultech,
Campus Côte de Nacre, F-14032 Caen Cedex, France,
gael.dias@unicaen.fr

[3] University of Porto, INESC-TEC,
Rua Dr. Roberto Frias, 378, 4200 - 465 Porto, Portugal
pbrazdil@inescporto.pt

Abstract. Sentence Reduction has recently received a great attention from the research community of Automatic Text Summarization. Sentence Reduction consists in the elimination of sentence components such as words, part-of-speech tags sequences or chunks without highly deteriorating the information contained in the sentence and its grammatical correctness. In this paper, we present an unsupervised scalable methodology for learning sentence reduction rules. Paraphrases are first discovered within a collection of automatically crawled Web News Stories and then textually aligned in order to extract interchangeable text fragment candidates, in particular *reduction cases*. As only positive examples exist, *Inductive Logic Programming* (ILP) provides an interesting learning paradigm for the extraction of sentence *reduction rules*. As a consequence, reduction cases are transformed into first order logic clauses to supply a massive set of suitable learning instances and an ILP learning environment is defined within the context of the *Aleph* framework. Experiments evidence good results in terms of irrelevancy elimination, syntactical correctness and reduction rate in a real-world environment as opposed to other methodologies proposed so far.

1 Introduction

The task of Sentence Reduction (or Sentence Compression) can be defined as summarizing a single sentence by removing information from it [14]. Therefore, the compressed sentence should retain the most important information and remain grammatical. But a more restricted definition is usually taken into account and defines sentence reduction as dropping any subset of words from the input sentence while retaining important information and grammaticality [6]. This formulation of the task provided the basis for the noisy-channel and decision tree based algorithms presented in [6], and for virtually most follow-up work on data-driven sentence compression [8] [13] [2]. One exception can be accounted for [15] [18], who consider sentence compression from a more general perspective and generate abstracts rather than extracts.

Sentence reduction has recently received a great deal of attentions from the Natural Language Processing research community for the number of applications where sentence compression can be applied. One of its (direct) applications is in automatic summarization [14] [17] [16]. But there exist many other interesting issues for sentence reduction such as automatic subtitling [21], human-computer interfaces [20] or semantic role labeling [19].

In this paper, we present an unsupervised scalable methodology for learning sentence reduction rules following the simplest definition of sentence compression. This definition makes three important assumptions: (1) only word deletions are possible and no substitutions or insertions allowed, (2) the word order is fixed and (3) the scope of sentence compression is limited to isolated sentences and the textual context is irrelevant. In other words, the compressed sentence must be a subsequence of the source sentence, which should retain the most important information and remain grammatical.

The proposed methodology is based on a pipeline. First, Web news stories are crawled and topically clustered. Then, paraphrase extraction and alignment are respectively performed based on text surface similarity measures and biologically-driven alignment algorithms. Then, *reduction cases*, which can be seen as interchangeable text fragments are extracted and transformed into first order logic clauses, eventually enriched with linguistic knowledge, to supply a massive set of suitable learning instances for an *Inductive Logic Programming* framework (*Aleph*). Experiments evidence good results in terms of irrelevancy elimination, syntactical correctness and reduction rate in a real-world environment as opposed to existing methodologies proposed so far.

2 Related Work

One of the first most relevant work is proposed by [6], who propose two methods. The first one is a probabilistic model called the noisy channel model in which the probabilities for sentence reduction $P(t|s)$ (where t is the compressed sentence and s is the original sentence) are estimated from a training set of 1035 (s,t) pairs, manually crafted, considering lexical and syntactical features. The second approach learns syntactic tree rewriting rules, defined through four operators: SHIFT, REDUCE, DROP and ASSIGN. Sequences of these operators are learned from a training set and each sequence defines a transformation from an original sentence to its compressed version. The results are interesting but the training data set is small and the methodology relies on deep syntactical analysis.

To avoid language dependency, [8] propose two sentence reduction algorithms. The first one is based on template-translation learning, a method inherited from the machine translation field, which learns lexical reduction rules by using a set of 1500 (s,t) pairs selected from a news agency and manually tuned to obtain the training data. Due to complexity difficulties, an improvement is proposed through a stochastic Hidden Markov Model, which is trained to decide the best sequence of lexical reduction rules that should be applied for a specific case. This work proposes an interesting issue but the lack of a large data set of learning pairs can not really assess its efficiency.

To avoid supervised learning, a semi-supervised approach is presented in [13], where the training data set is automatically extracted from the Penn Treebank corpus to fit a noisy channel model. Although it proposes an interesting idea to automatically provide new learning instances, it is still dependent upon a manually syntactically-labeled data set, in this case the Penn Treebank corpus.

[2] propose an hybrid system, where the sentence compression task is defined as an optimization of an integer programming problem. For that purpose, several constraints are defined, through language models, linguistic and syntactical features. Although this is an unsupervised approach, without using any parallel corpus, it is completely knowledge driven, as a set of hand-crafted rules and heuristics are incorporated into the system to solve the optimization problem.

More recently, [15] [18] propose a tree-to-tree transduction method for sentence compression. Their model is based on synchronous tree substitution grammar, a formalism that allows local distortion of the tree topology and can thus naturally capture structural mismatches. Their experimental results bring significant improvements over a state-of-the-art model, but still rely on supervised learning and deep linguistic analysis.

In this paper, we propose a real-world scalable unsupervised ILP learning framework based on paraphrase extraction and alignment. Moreover, only shallow linguistic features are introduced and propose an improvement over surface text processing.

3 The Overall Architecture

Our system autonomously clusters topically related Web news stories, from which paraphrases are extracted and aligned. From these aligned sentence pairs, structures called reduction cases are extracted and transformed into learning instances, which feed an ILP framework called *Aleph* appropriately configured for our problem. The learning cycle is then computed and a set of sentence reduction rules is generated, as a result.

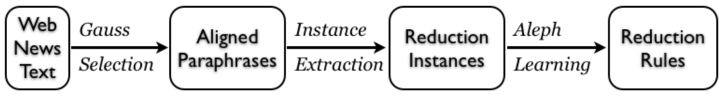

Fig. 1. The main architecture of our system.

A reduction rule obtained from the learning process states a number of conditions over a candidate elimination segment (X), as well as its relative left (L) and right (R) contextual segments[4]. In particular, the input Web news stories are splitted into sentences, which are then part-of-speech tagged and shallow parsed[5]. For that purpose, the *OpenNLP* library[6] has been used. So, given sentence (1),

```
Pressure will rise for congress to enact a massive fiscal stimulus package (1)
```

its shallow parsed counterpart is defined in sentence (2).

[4] A segment means just a sequence of words.
[5] A shallow parser is a non-hierarchical sentence parser.
[6] http://opennlp.apache.org/

```
[NP Pressure/NNP] [VP will/MD rise/VB] [PP for/IN] [NP congress/NN]
[VP to/TO enact/VB] [NP a/DT massive/JJ fiscal/JJ stimulus/NN package/NN] (2)
```

In the learning process, three types of learning features are considered: words, part-of-speech tags and chunks[7]. As a consequence, a learned rule may state conditions involving these three types. Formula 1, expresses such rule in a high level notation.

$$eliminate(X) \ <= |X| = 2 \ \wedge \ L_c = \texttt{VP} \ \wedge \ X_2 = \texttt{NN} \ \wedge R_1 = \texttt{to} \wedge R_c = \texttt{VP} \qquad (1)$$

The rule should be read as follows: *eliminate segment X if its size is two words long, its second word is a noun (*`NN`*), the left and right context segments are verb phrases (*`VP`*), and the first word of the right context is the word "to"*. We can see that this rule applies to sentence (2), resulting in the elimination of the "`for congress`" segment thus giving rise to the compressed sentence (3).

```
Pressure will rise to enact a massive fiscal stimulus package (3)
```

3.1 Paraphrase Extraction and Alignment

The first process called the *Gauss Selection* consists in the automatic identification and extraction of paraphrases from topically related Web news stories. It is based on lexical unigram exclusive overlaps in a similar way to some approaches in the literature [1, 5, 4]. In the work of [4], a significant comparison of paraphrase identification functions is proposed and results over standard test corpora reveal that some of their proposed functions achieve good performances. After several experiments, we noticed that the Gaussian functions could be adequately parametrized for the extraction of assymetrical paraphrases, thus satisfying our practical goal. A Gaussian function has the general form described in Equation 2.

$$f(x) = a \cdot e^{-\frac{(x-b)^2}{2 \cdot c^2}} \qquad (2)$$

In our case, the Gaussian function computes the likelihood of two sentences being paraphrases. So, (x) is the relative number of lexical unigrams in exclusive overlap between a pair of sentences. So, the more exclusive links two sentences share, the more likely they will be paraphrases. As the measure must be a real value in the unit interval a is settled to 1. Moreover, in order to learn reduction rule we must find paraphrases where one sentence is smaller than the other one. As a consequence, the b parameter allows to tune this dissimilarity degree, which in our case is equal to 0.5. An example is given below where sentences (4) and (5) are paraphrases.

```
Pressure will rise for congress to enact a massive fiscal stimulus package (4)
Pressure will rise to enact a fiscal package (5)
```

After paraphrase extraction, a combination of Biology-based sequence alignment algorithms [9] [11] is proposed in [3] to word-align any paraphrasic sentence pair. As a consequence, the alignment process over sentences (4) and (5) gives rise to the aligned sentences (6) and (7). For that purpose, a specific mutation matrix based on a modified version of the edit distance is computed as in [3].

[7] A sentence segment of related words grouped by a shallow parser.

```
Pressure will rise for congress to enact a massive fiscal stimulus package (6)
Pressure will rise ___ ________ to enact a _______ fiscal ________ package (7)
```

3.2 Learning Instance Selection

After identifying relevant paraphrases and aligning them at word level, specific text segments that evidence local sentence reduction cases, which can then be used as reduction instances in the learning process, must be defined. We call these structures *reduction cases*. For example, by looking at sentences (6) and (7), we can observe three reduction cases: (a) "`for congress`", (b) "`massive`", and (c) "`stimulus`".

So, first, in order to *consider* a reduction case, one must have a segment aligned with an empty sequence (lets say a middle segment, represented by *X*), surrounded by left (*L*) and right (*R*) contexts of commonly aligned words. For example, in segment (a) we have *L* = "`Preasure will rise`" and *R* ="`to enact a`", and so as a consequence, the triple $\langle L, X, R\rangle$ is selected as a reduction case. In that same alignment, we have two more reduction cases, with *X*="`massive`" and *X*="`stimulus`".

Then, in order to *select* a relevant reduction case, an evaluation function is defined as in Equation 3. In particular, the $value(\langle L, X, R\rangle)$ function gives preference to reduction cases having long contexts relatively to the misaligned segment X. The longer the contexts the higher the linguistic evidence indicating a true reduction case. A threshold must be set for the selection decision. We have pick one ensuring that the length of the contexts outweighs the length of the misaligned segment, i.e. $|X| \leq |L| + |R|$.

$$value(\langle L, X, R\rangle) = 1 - \frac{|X|}{|L| + |R| + \frac{1}{2}} \tag{3}$$

Following our example, the first selected reduction case is given in sentences (8) and (9). Note that all the other reduction cases would be selected although with different strength values.

```
Pressure will rise for congress to enact (8)
Pressure will rise ___ ________ to enact (9)
```

3.3 ILP Learning

After extracting sentence reduction cases, our final step is to transform them into learning instances in order to build a learning model. Within this context, one interesting advantage of ILP[8] is the possibility to learn exclusively from positive instances, contrarily to what is required by most supervised learning models. In our problem, this turns out to be a key aspect, since negative examples are difficult to obtain or even not available.

Another important characteristics of ILP is the way in which learning features can be defined, normally through first-order logic predicates. Indeed, most learning paradigms require a complete and exact feature set specification, before starting the learning process. With ILP, we can afford to simply define a broad set of "possible

[8] Inductive Logic Programming

features" that can be selected by the system during the learning process. This characteristics is particularly interesting as the set of all the exact learning features can be huge and as consequence lead to data sparseness in a classical learning paradigm.

We have considered three feature categories: *words*, *part-of-speech tags* and *chunks*. Each reduction case is transformed into a first-order logic representation, involving these features. An example related with the reduction case of section 3.2 is given below:

```
reduct(1, t(2,0),
    [pressure/nnp/np, will/md/vp, rise/vb/vp],
    [for/in/pp, congress/nn/np] ---> [],
    [to/to/vp, encat/vb/vp, a/dt/np]).
```

Each reduction case is represented by a 5-ary *Prolog* term "`reduct/5`". The first argument is a sequential number[9]. The second one contains a 2-ary term, which represents the reduction dimensionality, where its first argument is the misaligned segment size ($|X|$) and the second argument the kind of transformation that is applied, e.g. 0 means that there is a complete deletion of the misaligned segment. The third, fourth and fifth arguments contain *Prolog* lists representing respectively the L, X, and R segments. Each list element is a triple with the form of "`WORD/POS/CHUNK`".

4 The Aleph Framework

The learning process has been perfomed with an ILP system called *Aleph* [12], that we have specifically configured for our task. Aleph is a machine learning system written in *Prolog* and was initially designed to be a prototype for exploring ILP ideas. It has become a mature ILP implementation used in many research projects ranging form Biology to Natural Language Processing. In fact, Aleph is the successor of several and "more primitive" ILP systems such as: Progol [7] and FOIL [10], among others, and may be appropriately parametrized to emulate any of those older systems.

4.1 Configuring *Aleph*

Before starting any learning process in Aleph, a set of several specifications must be defined which will direct the learning process. Those involving the definition of the concept to be learned, the declaration of the predicates that can be involved in the rule formation, the definition of the learning procedure, optionally the definition of rule constraints and a set of learning parameters, among other details. In this subsection, we describe the most relevant settings, defined for our learning problem.

In Aleph, the learning instances are divided in three files: the background knowledge (BK) file (*.b) and two files containing the positive (*.f) and negative (*.n) learning instances . This last one is optional and was not used in our case, as explained before. Hence, our positive instances file contains the set of all sentence reduction cases extracted from the aligned paraphrasic sentences and transformed into first-order logic predicates.

The BK file contains the learning configurations including their associated predicates and parameters. We start by showing an excerpt of the head of our BK file, which contains the *settings*, *modes*, and *determinations*, for our learning task.

[9] It simply means the instance identifier

```
%-------------------------------------------
% SETTINGS
:- set(minpos, 5).
:- set(clauselength, 8).
:- set(evalfn, user).
:- set(verbosity, 0).
%-------------------------------------------
% DECLARATIONS
:- modeh(1, rule(+reduct)).
:- modeb(1, transfdim(+reduct, n(#nat,#nat))).
:- modeb(3, chunk(+reduct, #segm, #chk)).
:- modeb(*, inx(+reduct, #segm, #k, #tword)).
:- determination(rule/1, transfdim/2).
:- determination(rule/1, chunk/3).
:- determination(rule/1, inx/4).
```

The first block specifies the learning parameters we have been using, where `minpos` is the minimum coverage and `clauselength` is the maximum number of clauses (conditions) a rule can have. The `evalfn` parameter establishes that the rule evaluation function is defined by the "user", meaning that we are defining our own evaluation function in the BK file. The verbosity parameter is simply related with the level of output that is printed during the learning process.

The second block of the BK file header contains the main procedures for rule construction. The `modeh/2` function defines the "learning concept", which is called as `rule`. The `modeb/2` and `determination/2` directives establish the predicates that can be considered for rule construction, as well as the way they can be used, like number of times that a given predicate can occur in the rule (first argument of `modeb/2`).

In particular, we defined three predicates that can be used in the rule formation process: `transfdim/2`, `chunk/3`, and `inx/4`. The first one states the transformation dimensionality (e.g. a reduction from 2 to 0 words), the second one states the chunk type for a specific text segment and the third predicate states a positional[10] word or part-of-speech (POS) occurrence. Note that in the mode directives, the predicate arguments starting with # represent data types, which are also defined in the BK file. For instance, `#nat` and `#k` represent natural numbers, `#segm` a text segment, `#chk` a chunk tag and `#tword` represents either a word or a POS tag. In order to understand the kind of rules being produced, the following example codifies Formula 1 presented in section 3.

```
rule(A) :-
   transfdim(A,n(2,0)), chunk(A,left,vp),
   inx(A,center:x,2,pos(nn)),
   inx(A,right,1,to),
   chunk(A,right,vp).
```

From the rule body, we have `transfdim(A, n(2,0))` as the first literal, stating that it is a two word elimination rule. The second and fifth literals respectively state that the left and right textual contexts must be verb phrases (`vp`). The third literal states that the second word from the elimination segment (`center:x`) must be a noun (`pos(nn)`) and the fourth literal obliges that the first word in the right context must be "`to`". With this example we can see that different linguistic aspects (lexical, morpho-syntactical and shallow-syntactical) can be mingled into a single rule.

[10] In a relative index position (third argument: `#k`).

It is important to point at the fact that special concern has been dedicated to the misaligned segment i.e. literals of the form `chunk(A, center:x, *)`, as it can be formed by multiple chunks. Thus, only for this segment (`center:x`), we let rules with multiple chunk types to be generated. Two structures can be formed: `XP-YP` and `XP*YP`, with `XP` and `YP` representing chunk tags. In particular, the first structure means a sequence of exactly two chunks and the second structure represents a sequence of three or more chunks, with the first one being `XP` and the last one `YP`. For example, `pp*np` represents a sequence of three or more chunks starting with a prepositional phrase (`pp`) and ending with a noun phrase (`np`). This would match chunk sequences like "`pp np np`" or "`pp np vp np`".

We have set a *user-defined cost function* and a number of *integrity constraints* as a strategy to better shape and direct the learning process [12]. The cost function shown as follows combines the rule coverage with a given distribution length, giving preference to rules having four and five literals. The 17047 value is the number of learning instances used. For each training set, this value is automatically defined by the *Java* program that generates the *Aleph* learning files.

```
cost(_, [P,_,L], Cost) :-
    value_num_literals(L, ValueL),
    Cost is P/17047 * ValueL.

value_num_literals(1, 0.10). %     |
value_num_literals(2, 0.25). % 1.0 -                      _
value_num_literals(3, 0.50). %     |                      _     _
value_num_literals(4, 1.00). %     |                _     _     _     _
value_num_literals(5, 0.60). %     |          _     _     _     _     _     _
value_num_literals(6, 0.40). %     |    _     _     _     _     _     _     _
value_num_literals(7, 0.20). %   ----------------------------------------------->
value_num_literals(_, 0.00). %          1     2     3     4     5     6     7
```

The set of integrity constraints was designed to avoid certain undesired rule types, such as reduction rules without any condition over one of the three textual segments (left, center:x and right). This is achieved through the constraint shown below, where the `countSegmRestr/5` predicate counts the number of conditions on each segment.

```
false :-
    hypothesis(rule(_), Body, _),
    countSegmRestr(Body, NL, NX, NY, NR),
    not_valid(NL, NX, NY, NR).

not_valid( _,  0,  _,  _).  %--> the center:x segment is free
not_valid( 0,  _,  _,  _).  %--> left segment is free.
not_valid( _,  _,  _,  0).  %--> right segment is free.
```

As a consequence of several experimental iterations taken, we have decided that it would be better to include constraints for avoiding the generation of a kind of overgeneral rules, which are likely to yield bad reduction results. For example, rules that just constrain on chunks. This is exactly what the following two integrity constraints are stating:

```
false :-
    hypothesis(rule(_), Body, _),
    Body = (chunk(_,_,_), chunk(_,_,_), chunk(_,_,_)).
false :-
    hypothesis(rule(_), Body, _),
    Body = (transfdim(_,_), chunk(_,_,_), chunk(_,_,_), chunk(_,_,_)).
```

4.2 Learned Rules

The output of a learning run produces a set of sentence reduction rules similar to the ones illustrated in section 4.1. In particular, we will discuss the results of the quality of the set of learned reduction rules by applying them on new raw sentences and measuring their correctness with different measures in section 5. It is important to keep in mind that the learning model can generate thousands of reduction rules and in Table 1 we show only four of them, as well as their application on new sentences[11].

1	$L_1 = \texttt{IN} \wedge X_1 = \texttt{DT} \wedge R_1 = \texttt{NN} \wedge \lvert X\rvert = 4$	*for all the iraqi people and for* ***all those who love*** *iraq*	✓
2	$L_1 = \texttt{NNS} \wedge X_3 = \texttt{NN} \wedge R_1 = \texttt{IN}$	*we need new faces* ***and new blood*** *in politics*	✓
3	$L_c = \texttt{VP} \wedge X_1 = \texttt{NN} \wedge R_1 = \texttt{to} \wedge \lvert X\rvert = 1$	*my comment has* ***everything*** *to do with the way the*	✓
4	$L_1 = \texttt{NNS} \wedge X_2 = \texttt{NN} \wedge R_1 = \texttt{IN}$	shia and kurdish parties **took control** of parliament	×

Table 1. Four examples of learned rules applied to new text.

From these four examples, we can see that three rules were positively applied and the rule from case 4 was badly applied. This case illustrates one of the main difficulties that still persists: the generation of too general rules. Indeed, a good learning model must be balanced in terms of specificity and generality. In fact, specific rules may be very precise but seldom apply, while general rules may have high coverage but low precision. These issues can be evidenced by the kind of extracted rules. For example, rules 2 and 4 are similar and both state constraints only on morpho-syntactical information. As such, they are general rules On the contrary, rule 3 is more specific by stating that the right context R of a possible deletion of size one ($|X| = 1$) must contain the word "to" immediately after the deleted segment (R_1 = $\texttt{to}$). Therefore, it is much less error prone.

5 Experimental Results

To estimate the quality of the produced reduction rules, we followed an empirical experiment using a data set of Web news stories collected along a period of 90 days over Google News API. This data set is called T90Days and contains 90 XML files, one per day, covering the most relevant news events from each day.

In each given file, the news are grouped by events or topics, where each one contains a bunch of related documents[12]. The T90Days corpus contains a total of 474MB of text data and a total number of $53\ 986$ aligned paraphrases extracted through the method described in subsection 3.1. From these aligned paraphrases, a total of $13\ 745$ reduction cases were selected and transformed into learning instances following the methodology described in subsection 3.2. Finally, the induction process yielded an amount of 2507 sentence reduction rules. It is important to notice that all these data sets and results are

[11] To fit space constraints, we only show the relevant sentence fragment and not the overall sentence. Moreover, the marked segment is the deleted one.

[12] Usually from 20 to 30 Web news stories.

freely available[13] in order to provide the research community with a large scale golden data set compared to the existing ones so far.

The evaluation of the induced rule set was performed over news set, different from the one used for learning. For the sake of this evaluation, we applied the best rule to each sentence and compared it with the baseline, which consists in directly applying the reduction cases to the sentences i.e. only lexical information is taken into account.

In particular, we had to define what is the "best" rule to apply to a given sentence. For that purpose, rule optimality was computed by combining rule support, number of eliminated words and the application of a syntactical 4-gram language model applied to the context of the reduction segment. While rule support guarantees some confidence in the rule application and the number of eliminated words must be maximized, the idea of the syntactical language model is to guarantee the syntactical correctness of the sentence after the application of the deletion. As a consequence, only the reduction rules, which can guarantee that the compressed sentence will be more syntactically correct than the longer one will be applied. For that purpose, we trained a syntactical 4-gram language model over a part-of-speech tagged corpus to evaluate the syntactical complexity of any given sentence by a sequence probability as defined in Equation 4. Here, $F = [t_1, t_2, ..., t_m]$ is the sequence of part-of-speech tags for a given sentence with size m. In particular, $P(t) > P(s)$ is the condition that triggers the application of the sentence reduction rule, where t is the compressed version of sentence s.

$$P(F) = \Big(\prod_{k=4}^{m-4} P(t_k \mid t_{k-1}, ..., t_{k-4}) \Big)^{\frac{1}{m}} \tag{4}$$

So, for each one of the two evaluations (baseline and best rule application), a random sample of 100 reductions was manually cross-evaluated by two human annotators. In order to take into account irrelevancy elimination and syntactical correctness, each reduction had to be scored with a value of 1 (incorrect reduction), 2 (semantically correct but incorrect syntactically) and 3 (semantically and syntactically correct). Additionally, each score was weighted by the number of eliminated words in order to give more importance to longer reductions. The results are presented in Table 2 for a *Cohen's K* value for inter-rater agreement of 0.478, meaning "moderate agreement".

Test	**Mean Rank**	**Precision**	**Mean $\lvert X \rvert$**	**Rules/Sentence**
Baseline	1.983	66.09%	1.15	0.042
ILP Rules	2.056	68.54%	1.78	5.639

Table 2. Results with four evaluation parameters.

In particular, column 2 (**Mean Rank**) presents the average value of both annotators. Column 3 contains the average size of the eliminated segment (in words) and column 4 evaluates the ratio of the number of rules applied by sentence. In fact, columns 3 and 4 evidence the *utility* of a rule set in terms of the number of eliminated words and the

[13] `http://www.di.ubi.pt/~jpaulo/competence/`

number of reduced sentences. As stated before, on one hand the baseline test consists in the direct use of the reduction cases, as they are in T90Days, on sentence reduction, meaning that no induction was used. This approach resembles to what is done in [8]. On the other hand, the ILP Rule test implies the application of the best learned rule and shows an improvement both in terms of quality and reduction size, although both results still need to be improved. In the final section, we will propose new perspectives to improve our approach.

6 Conclusions and Future Directions

In this paper, we described an ILP learning strategy that learns sentence reduction rules from massive textual data automatically collected from the Web. After paraphrase identification based on Gaussian functions and alignment through a combination of biology-based sequence alignment algorithms, sentence reduction cases are selected and prepared to be used in the learning process, handled by an ILP framework called *Aleph* [12]. Different aspects distinguish our system from existing works. First, it relies on its exclusive automation. Each step takes place in a pipeline of tasks, which is completely automatic. As a result, the system can process huge volumes of data compared to existing works. Second, it is based on shallow linguistic processing, which can easily be adapted to new languages. Finally, we propose a freely available golden data set to the research community in order to apply existing techniques to larger data sets than existing ones [6].

However, improvements still need to be performed. As the overall strategy is based on a pipeline, different errors tend to accumulate step after step. So, each stage must be individually improved. In particular, we noticed from the results that many errors were due to incorrect text tokenization. As a consequence, we believe that the identification of multiword units will improve the quality of rule induction. Moreover, we will propose to automatically tune the generalization process so that we can avoid the induction of over-generalized reduction rules.

Acknowledgements

This work is funded by the ERDF – European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project “FCOMP - 01-0124-FEDER-022701”.

References

1. Barzilay R, Lee L. 2003. Learning to Paraphrase: An Unsupervised Approach using Multiple-Sequence Alignment. *Proceedings of the 4th North American Chapter of the Association for Computational Linguistics Conference* (NAACL 2003).
2. Clarke J, Lapata M. 2006. Constraint-based Sentence Compression: An Integer Programming Approach. *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics* (ACL 2006).

3. Cordeiro J, Dias G, Cleuziou G, Brazdil P. 2007. Biology Based Alignments of Paraphrases for Sentence Compression. *Proceedings of the Workshop on Textual Entailment and Paraphrasing associated to the 45th Annual Meeting of the Association for Computational Linguistics Conference* (ACL 2007).
4. Cordeiro J, Dias G, Brazdil P. 2007. New Functions for Unsupervised Asymmetrical Paraphrase Detection. Anonymous. Anonymous.
5. Dolan WB, Quirck C, Brockett C. 2004. Unsupervised Construction of Large Paraphrase Corpora: Exploiting Massively Parallel News Sources. *Proceedings of 20th International Conference on Computational Linguistics* (COLING 2004).
6. Knight K., Marcu D. 2002. Summarization beyond sentence extraction: A probabilistic approach to sentence compression. *Artificial Intelligence*, 139(1):91-107.
7. Muggleton S. 1999. Inductive Logic Programming: Issues, Results and the Challenge of Learning Language in Logic. *Artificial Intelligence*, 114(1-2):283-296.
8. Le Nguyen M, Horiguchi S, Ho BT. 2004. Example-based Sentence Reduction using the Hidden Markov Model. *ACM Transactions on Asian Language Information Processing*, 3(2):146-158.
9. Needleman SB, Wunsch CD. 1970. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology*, 48(3):443-453.
10. Quinlan JR. 1990. Learning Logical Deinitions from Relations. *Machine Learning.*, 5(3):239-266.
11. Smith TF, Waterman MS. 1981. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147:195-197.
12. Srinivasan A. 2000. *The Aleph Manual*, Technical Report. Computing Laboratory, Oxford University, UK.
13. Turner J, Charniak E. 2005. Supervised and Unsupervised Learning for Sentence Compression. *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics Conference* (ACL 2005).
14. Hongyan H, McKeown KR. 2000. Cut and Paste based Text Summarization. *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference* (NAACL 2000).
15. Cohn T, Lapata M. 2008. Sentence Compression Beyond Word Deletion. *Proceedings of the 22nd International Conference on Computational Linguistics* (COLING 2008).
16. Zajic DM, Dorr BJ, Lin J. 2008. Single-Document and Multi-Document Summarization Techniques for Email Threads using Sentence Compression. *Information Processing and Management*, 44(4):1600–1610.
17. Siddharthan A, Nenkova A, McKeown K. 2004. Syntactic Simplification for Improving Content Selection in Multi-Document Summarization. *Proceedings of the 20th International Conference on Computational Linguistics* (COLING 2004).
18. Cohn T, Lapata M. 2009. Sentence Compression as Tree Transduction. *Journal of Artificial Intelligence Research*, 34(1):637-674.
19. Vickrey D, Koller D. 2008. Sentence Simplification for Semantic Role Labeling. *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics Conference* (ACL 2008).
20. Corston-Oliver S. 2001. Text Compaction for Display on Very Small Screens. *Proceedings of the Workshop on Automatic Summarization associated to the 2nd North American Chapter of the Association for Computational Linguistics Conference* (NAACL 2001).
21. Vandeghinste V, Pan Y. 2004. Sentence Compression for Automated Subtitling: A Hybrid Approach. *Proceedings of the Workshop on Text Summarization Branches Out associated to the 44th Annual Meeting of the Association for Computational Linguistics Conference* (ACL 2004).