$(LP)^2$, an Adaptive Algorithm for Information Extraction from Web-related Texts

Fabio Ciravegna
Department of Computer Science, University of Sheffield
Regent Court, 211 Portobello Street,
S1 4DP Sheffield, UK
F.Ciravegna@dcs.shef.ac.uk

in Proceedings of the IJCAI-2001 Workshop on Adaptive Text Extraction and Mining to be held in conjunction with the 17th International Conference on Artificial Intelligence (IJCAI-01), Seattle, August, 2001

Abstract

(LP)² is an algorithm for adaptive Information Extraction from Web-related text that induces symbolic rules by learning from a corpus tagged with SGML tags. Induction is performed by bottom-up generalisation of examples in a training corpus. Training is performed in two steps: initially a set of tagging rules is learned; then additional rules are induced to correct mistakes and imprecision in tagging. Shallow NLP is used to generalise rules beyond the flat word structure. Generalization allows a better coverage on unseen texts, as it limits data sparseness and overfitting in the training phase. In experiments on publicly available corpora the algorithm outperforms any other algorithm presented in literature and tested on the same corpora. Experiments also show a significant gain in using NLP in terms of (1) effectiveness (2) reduction of training time and (3) training corpus size. In this paper we present the machine learning algorithm for rule induction. In particular we focus on the NLP-based generalisation and the strategy for pruning both the search space and the final rule set.

1. Introduction

By general agreement the main barriers to wide use and commercialization of Information Extraction from text (IE) are the difficulties in adapting systems to new applications. Classical IE systems often rely on approaches based on Natural Language Processing (NLP) (e.g. using parsing) [Humphreys *et al.* 1998; Grishman 1997]. Most current IE systems require the involvement of IE experts for new applications development [Ciravegna 2000]. This is a serious limitation for the wider acceptance of IE, especially in the Internet realm: most small/medium enterprises (i.e. the backbone of the New Economy) cannot afford to hire specialists in IE. For this reason there is an increasing interest in applying machine learning (ML) to IE in order to build adaptive systems. Up to now, the use of ML has been approached mainly in an NLP-oriented perspective, i.e. in order to reduce the amount of work to be done by the NLP experts in porting systems across free text based scenarios [Cardie 1997; Miller *et al.* 1998; Yangarber *et al.* 2000].

In the last few years Information Extraction from texts (IE) has been focusing progressively on the Web, i.e. away from the newspaper-based text IE analysed in the MUC conferences. This is due both to the reduction of strategic funds available for research, and to the increase in the potential IE applications in the Web realm. The Web emphasises the central role of texts such as emails, Usenet posts and Web pages. In this context, extralinguistic structures (e.g. HTML tags, document formatting, and stereotypical language) are elements used to convey information. Linguistically intensive approaches as used in classical IE systems [Grishman 1997, Humphreys *et al.* 1998] are difficult or unnecessary or ineffective in such cases. For this reason a new research stream on adaptive IE has arisen at the convergence of NLP, Information Integration and Machine Learning. The goal is to produce IE algorithms and systems adaptable to new Internet-related applications/scenarios by using only an analyst's knowledge (i.e. knowledge on the domain/scenario itself) [Kushmerick 1997; Califf 1998; Muslea *et al.* 1998; Freitag and McCallum 1999; Soderland 1999; Freitag and Kushmerick 2000]. As result successful commercial products have been created and there is an increasing interest in IE in the Internet market. Currently available technology is very effective when applied to highly structured HTML pages, but less effective with unstructured texts (e.g. free texts). In our opinion, this is because most successful algorithms tend to avoid any generalisation over the flat word sequence. When they are applied to unstructured texts, data sparseness becomes a problem. Data sparseness is relevant for: (1) the size of the training data, the more sparse the data are, the more examples are needed for training; (2) quality of results, sparse data cause the generated rules to be applicable to a limited number of cases, overfitting the training examples, and therefore affecting effectiveness on unseen cases.

This paper presents **(LP)²** (Learning Pattern by Language Processing), an adaptive IE algorithm designed in this new stream of research that makes use of shallow NLP in order to overcome data sparseness when confronted with NL texts, while keeping effectiveness on highly structured texts. Experimentally the algorithm outperforms any other algorithm presented in the literature on a number of testbeds. In particular we focus on the machine learning algorithm for rule induction, on the NLP-based generalisation and the strategy for pruning both the search space and the final rule set.

2 Types of Induced Rules

(LP)² learns rules by generalising over a set of examples marked via SGML tags in a training corpus. It induces two types of symbolic rules: **tagging rules** and **correction rules.** This section

presents the types of rules the algorithm induces, while section 4 focuses on rule generalisation.

2.1 Tagging Rules

A tagging rule is composed of a left hand side, containing a pattern of conditions on a connected sequence of words, and a right hand side that is an action inserting an SGML tag in the texts. Each rule inserts a single SGML tag, e.g. *</speaker>*. This makes **(LP)²** different from many adaptive IE algorithms, whose rules recognize whole slot fillers (i.e. insert both *<speaker>* and *</speaker>*, [Califf 1998, Freitag 1998] or even multi slots, [Soderland 1999]. As positive examples the tagging rule induction algorithm uses SGML tags inserted by a user in a training corpus. The rest of the corpus is considered a pool of negative examples. For each positive example the algorithm: (1) builds an initial rule, (2) generalizes the rule and (3) keeps the *k* best generalizations of the initial rule.

In particular **(LP)²**'s main loop starts by selecting a tag in the training corpus and extracting from the text a window of ***w*** words to the left and ***w*** words to the right. Each information stored in the 2****w*** word window is transformed into a condition in the initial rule pattern, e.g. if the third word is "seminar", a condition `word3="seminar"` is created.

Each initial rule is then generalised (see next sections) and the *k* best generalisations are kept: retained rules become part of the **best rules pool**. When a rule enters such pool, all the instances covered by the rule are removed from the positive examples pool, i.e. they will no longer be used for rule induction (**(LP)²** is a sequential covering algorithm). Rule induction continues by selecting new instances and learning rules until the pool of positive examples is void.

2.2 Contextual Rules

When applied to the test corpus, the best rules pool provides good results in terms of precision, but limited effectiveness in terms of recall. This means that such rules insert few tags (low recall), and that such tags are generally correct (high precision). This

Condition	Action
Word=?	Insert Tag
the	
seminar	
at	**stime**
4	
pm	

Figure 1: Starting rule (with associated NLP knowledge) inserting <stime> in the sentence ``the seminar at <stime> 4 pm …"

condition		action
word	wrong tag	move tag to
at		
4	*</stime>*	
pm		*</stime>*

Figure 2: A a correction rule. The action (not shown) shifts the tag from the wrong to the correct position**.**

is because just a limited number of rules are able to match the absolute reliability condition required for selection. In order to reach acceptable effectiveness, it is necessary to identify additional rules able to raise recall without affecting precision. **(LP)²** recovers some of the non-best rules and tries to constrain their application to make them reliable. Interdependencies among tags are exploited to constrain the application of such rules. As mentioned, **(LP)²** learns rules for inserting tags (e.g., *<speaker>*) independently from other tags (e.g., *</speaker>*). But tags are not independent: (1) tags represent slots, therefore $<tag_x>$ always requires $</tag_x>$; (2) slots can be concatenated into linguistic patterns and therefore the presence of a slot can be a good indicator of the presence of another, e.g. *</speaker>* can be used as "anchor tag" for inserting *<stime>*[1]. (1) and (2) can be summarized as: $<tag_x>$ can be used as indicator of the presence of $<tag_y>$. Considering that single tag rules are not able to model such dependencies, the context is reintroduced in **(LP)²** as an external constraint used to improve the reliability of some rules. In particular, **(LP)²** reconsiders low precision non-best rules for application in the context of tags inserted by the best rules only. For example some rules will be used only to close slots when the best rules were able to open it, but not to close it. Selected rules are called **contextual rules**. As example consider a rule inserting a `</speaker>` tag between a capitalized word and a lowercase word. This is not a best rule as it reports high recall/low precision on the corpus, but it is reliable if used only to close an open `<speaker>`. Thus it will only be applied when the best rules have already recognized an open `<speaker>`, but not the corresponding `</speaker>`. "Anchor tags" used as contexts can be found either to the right of the rule space application (e.g. when anchor tag is `<speaker>`), or to the left as in the opposite case (anchor tag is `</speaker>`). Reliability for contextual rules is computed by using the same error rate used for best rules.

In conclusion the tagging rule set is composed of both the best rule pool and the contextual rules.

[1] Here we use examples related to the first case as it is more intuitive.

2.3 Correction Rules

Tagging rules when applied on the test corpus report some imprecision in slot filler boundary detection. A typical mistake is for example "at <time> 4 </time> pm", where "pm" should have been part of the time expression. For this reason **(LP)²** induces rules for shifting wrongly positioned tags to the correct position. It learns from the mistakes made in tagging the training corpus. Correction rules are identical to tagging rules, but (1) their patterns match also the tags inserted by the tagging rules and (2) their actions shift misplaced tags rather than adding new ones. An example of an initial correction rule for shifting *</stime>* in "at <*stime*> 4 *</stime>* pm" is shown in Figure 2. The induction algorithm used for the best tagging rules is also used for shift rules: initial instance identification, generalisation, test and selection. Positive (correct shifts) and negative (wrong shifts of correctly assigned tags) are counted. Shift rules are accepted only if they report an acceptable error rate.

3. Rule Application for IE

In the testing phase, information is extracted from the test corpus in four steps: initial tagging, contextual tagging, correction and validation. The best rule pool is initially used to tag the texts. Then contextual rules are applied in the context of the introduced tags. They are applied until new tags are inserted, i.e. some contextual rules can match also tags inserted by other contextual rules. Then correction rules correct some imprecision. Finally each tag inserted by the algorithm is validated. There is no meaning in producing a start tag (e.g. <*speaker*>) without its corresponding closing tag (*</speaker>*) and vice versa, therefore uncoupled tags are removed in the validation phase.

4. The Rule Induction Algorithm

The types of rule mentioned above are all induced by the same algorithm. As mentioned, the initial rule pattern matches conditions on word strings as found in a window *w* around each instance. Then each rule is generalised. Generalisation is important in in analysing natural language input, because of data sparseness due to the high flexibility of natural language forms. Avoiding generalisation actually means producing a big rule set composed of rules covering a limited number of cases each. Such rule set is very likely to produce very good results on the training corpus, but very limited accuracy on the test corpus. This is the well known problem of overfitting the training corpus: on the one hand the system learns a number of rules for covering unrelated cases: if such cases are not found as they are, the rule will not apply at testing time (leading to low recall). On the other hand the rule set is sensible to errors in the training data: such errors can either prevent some rules derived from correct examples from being accepted as they report errors (low recall again during test) or can produce spurious results at testing time (low precision).
It is therefore important on the one hand to generalise over the plain word surface of the training example in order to produce rules able to overcome data sparseness. On the other hand it is necessary to prune the resulting rule set in order to reduce overfitting, as explained in the remainder of the section.

word index	**Condition**	**Additional Knowledge**				**Action**
	Word	Lemma	LexCat	case	SemCat	Tag
1	The	the	Art	low		
2	Seminar	Seminar	Noun	low		
3	at	at	Prep	low		**stime**
4	4	4	Digit	low		
5	pm	pm	Other	low	timeid	
6	will	will	Verb	low		

Figure 3: The rule in figure 1 with associated NLP knowledge.

4.1 Rule Generalisation

There are two ways in which the algorithm generalises the initially induced rules: on the one hand constraints in the initial pattern are dropped (e.g. patterns are reduced in length or some forms of wildcards are used) this allows to model cases that slightly differ (e.g. « at 4 pm » and « at 5 pm » can be modelled by the rule « word=at, word=*, word=pm »). On the other hand conditions on single elements are relaxed by using NLP-based information. Shallow Natural Language Processing is used to associate additional knowledge to each word in the initial pattern via a morphological analyser (providing lemma + case information), a POS tagger (lexical category, e.g. noun) and a user-defined dictionary (or a gazetteer, if available) (Figure 3). Conditions on each element in the rule pattern are relaxed by substituting constraints on words with constraints on some parts of the additional knowledge (Figure 4). In the example mentioned above (« at 4 pm » and « at 5 pm ») the

Word index	**Condition**					**Action**
	Word	Lemma	LexCat	Case	SemCat	Tag
3		at				**Stime**
4			Digit			
5					timeid	

Figure 4: One generalisation for the rule above.

rule « word=at, lexCat=DIGIT, word=pm » is able to better generalise over the two cases than the rule using a wildcard.

We have implemented different strategies for rule generalisation. The naïve version of the algorithm [Ciravegna 2001] [Ciravegna2001b] generates all the possible rules in parallel. Each generalisation is then tested separately on the training corpus and an error score *E=wrong/matched* is calculated. For each initial rule, the *k* best generalisations are kept that: (1) report better accuracy; (2) cover more positive examples; (3) cover different parts of input. The naïve generalisation is quite expensive in computational terms. Here we describe a more efficient version of the algorithm that uses a general to specific beam search for the best *k* rules in a way similar to AQ [Michalski 1986]. It starts by modelling a specific instance with the most general rule (the empty rule matching every instance) and specialises it by greedily adding constraints. Constraints are added by incrementing the length of the rule pattern, i.e. by adding conditions on terms. Figure 6 shows the search space for an example of this type of generalisation for the case in which conditions are set only on words. The induction algorithm is shown in figure *5*. This algorithm is as effective as the naïve one, but it is more efficient, as it allows a very efficient rule testing. As a matter of fact the matches of a specialised rule can be

```
InduceRuleFromInstance(instance)
 RuleSet=startRulesFromEmptyRule(instance);
 Loop while notEmpty(RuleSet) {
  Loop for  rule1 in butLast(RuleSet){
       for rule2 in butFirst(RuleSet)
   ruleSet=combineRules(rule1, rule2);
   add(ruleSet, finalRuleSet);
   }  }
return finalRuleSet;
}

startRulesFromEmptyRule (instance){
 tag=instance.tag
 loop for distance from 0 to 2*w {
  do word=instance.pattern[distance]
     /* w-position is the distance between
        the current word and the tag to be
        inserted */
  collect generateRule(word,tag,w-position)
 } }
```

Figure 5: the basic algorithm for non NLP-based generalisation

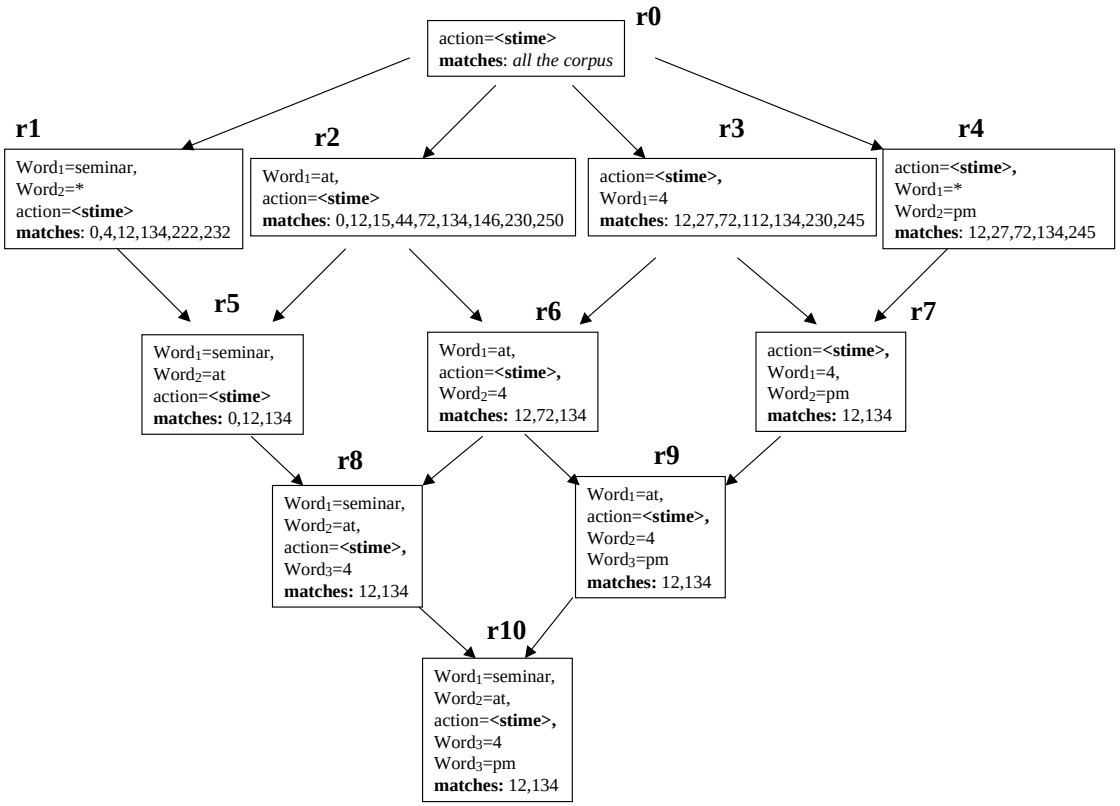

Figure 6: pattern generalisation for "the seminar at **<*stime*>** 4 pm will" with window *w*=2 (conditions on words only). The rule action is inserted in the part of the pattern where the tag is inserted. For example r6 matches “at” + “4” and inserts a tag after “at”. The “matches” fields contain indeces in the corpus where rules apply.

computed as the intersection of the matches of two more general rules, making testing an order of magnitude more efficient (see figure 6). The actual algorithm is more complex than shown, as it also relaxes conditions on words in rule patterns by using conditions on some parts of the NLP-based additional knowledge associated to each word. In order to introduce this type of generalisation the algorithm in figure 6 is modified so that `RuleSet` does no longer contain rules, but sets of rules derived by concurrent generalisation of the lexical items. The final algorithm is shown in figure 7.

4.2 Rule Set Pruning

Pruning is performed both for efficiency reasons (so to reduce the search space during generalisation) and to reduce overfitting. Pruning removes rules that either are unreliable, or whose coverage overlaps with those of other rules.

There are two types of unreliable rules: those with a high error rate (i.e. performing poorly on the training corpus) and those reporting a very limited number of matches on the training corpus, so that it is not easy to foresee what their behaviour at test time will be.

In order to prune the final rule set from such elements, rule accuracy is tested against a maximum error threshold set by the user before running the algorithm. Rules that do not pass the test are discarded as soon as they are generated and tested. They are no longer considered for IE, but they are considered for further specialisation, as they could become reliable after the addition of some more constraints. At the end of training the algorithm tries to optimise the error threshold by restricting the rule selection condition, therefore excluding some other rules. It tests the results of the reduced rule set on the training corpus and stops pruning the moment in which there is a reduction of accuracy (seen as the mean of precision and recall).

Rules are also pruned at induction time when they cover too few cases in the training corpus and therefore they cannot be safely used at test time because the amount of training data does not allow to guarantee a safe behaviour at run time. Such rules are detected at generation time and any further specialisation depending from them is stopped. In this way the search space for the rule is reduced and the algorithm is more efficient. For example with a threshold on minimum coverage set to 3 cases, rules 9 and 10 would have never been generated in the example in figure 6. Again the user can set a priori the minimum coverage threshold and such threshold will be optimised at the end of training in a way similar to the error threshold optimisation.

```
InduceRuleFromInstance(instance)
 RuleSet=startRulesFromEmptyRule(instance);
 Loop while notEmpty(RuleSet) {
  NewRuleSetS={}
  Loop for  ruleSet1 in butLast(RuleSet){
       for ruleSet2 in butFirst(RuleSet)
   ruleSet=combineRuleSet(ruleSet1, ruleSet2);
   pruneRuleSet(ruleSet);
   add(ruleSet, newRuleSetS);
   add(ruleSet, finalRuleSets);
   }
  RuleSet=NewRuleSetS;
 }
 PruneRuleSets(finalRuleSets)
}

startRulesFromEmptyRule (instance)
  tag=instance.tag
  Loop for distance from 0 to 2*w {
    word=instance.pattern[distance]
        /* w-position is the distance between
        the current word and the tag to be
        inserted */
    generateRuleSet (word, tag, w-position)
 }
```

```
generateRuleSet(word, tag, position){
 Loop for condit in NLPGeneralisations(word)
  Collect generateRule(condit, tag, position)
}

PruneRuleSet (ruleSet){
  For rule in ruleSet
   if (not(subsumedRule(rule, RuleSet)))
   collect rule
}

subsumedRule (rule, ruleSet){
  loop for ru in ruleSet
   if (ru!=rule)
    if((ru.score<=rule.score) and
       (includes(ru.coverage,rule.coverage)))
        return true;
 return false;
}
```

Figure 7: the final algorithm for rule generalisation by starting from an initial rule.

There is a further level of pruning based on overlapping rule coverage: all the rules whose coverage is subsumed by that of others more general ones are removed from the selected rule set. Again the goal is to determine the minimum subset of rules that maximises the accuracy of IE on the training corpus. Rules whose coverage is subsumed by those of other rules can be safely removed, as their contribution to the final results is irrelevant. Such type of pruning requires comparing not only coverage, but also the reliability of the involved rules (otherwise the algorithm would only produce one rule, i.e. the empty initial rule!). In general rules subsumed by other rules with the same (or minor) error rate can be safely removed during rule induction. Rules subsumed by ones with worse error rate are pruned when the final error and covering threshold have been determined, as it is necessary to see if the subsuming rule will survive such rule pruning process.

A problem arises in pruning at every level when two rules cover the same set of examples with the same error rate. In this case the following heuristic is used. If the number of covered cases is limited, then the one with most specific conditions is chosen (e.g. one using condition on words). This is because the training corpus does not provide enough evidence that the rule is reliable and a rule requiring a sequence of words is less likely to produce spurious results at test time than one requiring sequences of conditions on the additional knowledge (e.g. on lexical categories). Otherwise, the rule with the most generic conditions is selected (e.g. testing lexical categories), as it is more likely to provide coverage on the test corpus.

5. Experimental Results

(LP)² was tested in a number of tasks in two languages: English and Italian. Here we report results on two standard tasks for adaptive IE: the CMU seminar announcements and the Austin job announcements[2]. The first task consists of uniquely identifying speaker name, starting time, ending time and location in 485 seminar announcements, [Freitag 1998]. Figure 5 shows the overall accuracy obtained by **(LP)²**, and compares it with that obtained by other state of the art algorithms. **(LP)²** definitely outperforms other NLP-based approaches (+8.7% wrt Rapier [Califf 1998], +21% wrt to Whisk [Soderland 1999]), but it also outperforms non-NLP approaches (+2.1% wrt BWI, [Freitag and Kushmerick 2000], and +4% wrt HMM, [Freitag and McCallum 1999]). Moreover **(LP)²** is the only algorithm whose results

	(LP)²	**BWI**	**HMM**	**SRV**	**Rapier**	**Whisk**
speaker	**77.6**	67.7	76.6	56.3	53.0	18.3
location	75.0	76.7	**78.6**	72.3	72.7	66.4
stime	99.0	**99.6**	98.5	98.5	93.4	92.6
etime	95.5	93.9	62.1	77.9	**96.2**	86.0
All Slots	**86.0**	**83.9**	**82.0**	**77.1**	**77.3**	**64.9**

Figure 10: results (F-measure β=1) obtained on CMU seminars in a 10 experiments using 1/2 corpus for training. See [Ciravegna 2001b] for details on the experiments.

Slot	**(LP)²**	Rapier	BWI
id	**100**	97.5	100
title	**43.9**	40.5	50.1
company	**71.9**	69.5	78.2
salary	62.8	**67.4**	
recruiter	**80.6**	68.4	
state	84.7	**90.2**	
city	**93.0**	90.4	
country	81.0	**93.2**	
language	**91.0**	80.6	

Slot	**(LP)²**	Rapier
Platform	**80.5**	72.5
Application	**78.4**	69.3
Area	**66.9**	42.4
Req-years-e	**68.8**	67.1
Des-years-e	60.4	**87.5**
Req-degree	**84.7**	81.5
des-degree	65.1	**72.2**
post date	**99.5**	**99.5**
All Slots	**84.1**	75.1

Figure 8: F-measure (β=1) for misc.jobs.offered using 1/2 corpus for training. Whisk obtained lower accuracy than Rapier, [Califf 1998]. We cannot compare (LP)² with BWI as the latter was tested on a limited subset of slots

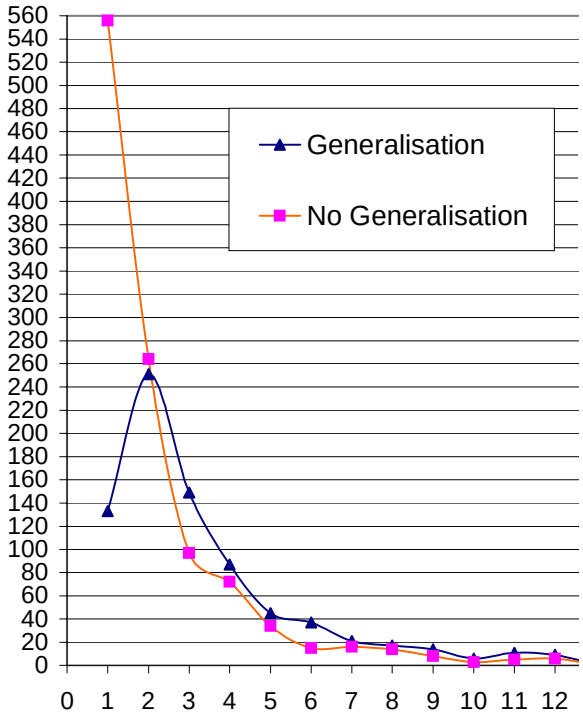

Figure 9: the effect of generalisation in reducing data sparseness: number of rules (on y) covering number of cases (on x : we show up to 12 cases)

never go down 75% on any of the slots.

A second task concerned IE from 300 Job Announcements taken from misc.jobs.offered, [Califf 1998]. **(LP)^2** outperforms both Rapier and Whisk (Figure 8).

6. Discussion

(LP)^2's main features that are most likely to contribute to the excellence in the experiments are: (1) the use of single tag rules, (2) the use of a correction phase, and (3) rule generalisation via shallow NLP processing. Points 1 and 2 have been discussed in [Ciravegna 2001b]. Here we focus on the effect of NLP-based generalisation. (LP)^2 induces rules by instance generalisation through shallow NLP processing. Generalisation of examples in the training corpus allows reducing data sparseness by capturing some general aspects beyond the simple flat word structure. Morphology allows overcoming of data sparseness due to number/gender word realisations (an aspect very relevant in morphologically rich languages such as Italian), while POS tagging information allows generalisation over lexical categories. In principle such types of generalisation produce rules of better quality than those matching the flat word sequence; rules that tend to be more effective on unseen cases. This is because both morphology and POS tagging are generic NLP processes performing equally well on unseen cases; therefore rules relying on their results apply successful on unseen cases. This intuition was confirmed experimentally: (LP)^2 with generalisation ((LP)^2_G) definitely outperforms a version without NLP generalisation (but with pattern length generalisation) ((LP)$^2_{NG}$) on the test corpus (Figure 11), while having comparable results on the training corpus. Moreover

Slot	**(LP)^2_G**	**(LP)$^2_{NG}$**
speaker	72.1	14.5
location	74.1	58.2
stime	100	97.4
etime	96.4	87.1
All slots	**89.7**	**78.2**

	(LP)^2_G	**(LP)$^2_{NG}$**
Average rule coverage	10.2	6.2
Selected rules	887	1136
Rules covering 1 case	133 (14%)	560 (50%)
Rules covering >50 cases	37	15
Rules covering >100 cases	19	3

Figure 11: comparison between the NLP-based generalisation version (LP)^2_G with the version without generalisation (LP)$^2_{NG}$

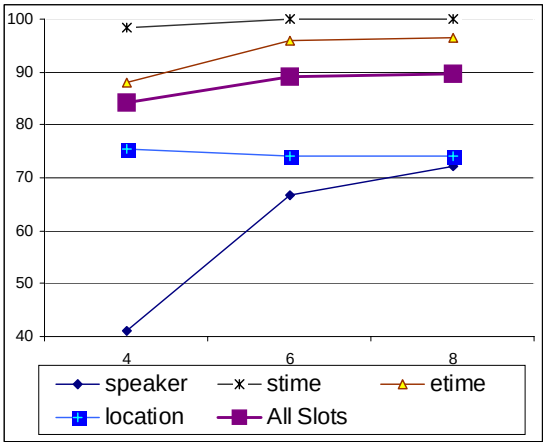

Figure 12: the effect of different window sizes in learning tagging rules : some fields are not affected (e.g. location), while others (speaker) are sensible to window size.

(LP)^2_G produces more general rules, i.e. rules covering more cases (Figures 10 and 11). NLP reduces the risk of overfitting the training examples. Figure 13 shows experimentally how (LP)^2_G avoids overfitting on the *speaker* field (constant rise in f-measure when the training corpus is increased in size), while (LP)$^2_{NG}$ present a clear problem of overfitting (reductionof effectiveness when more examples are provided). Producing more general rules also implies that the covering algorithm converges more rapidly because its rules tend to cover more cases. This means that (LP)^2_G needs less examples in order to be trained, i.e., rule generalisation also allows reducing the training corpus size. Figure 13 shows how on *etime* (LP)^2_G is

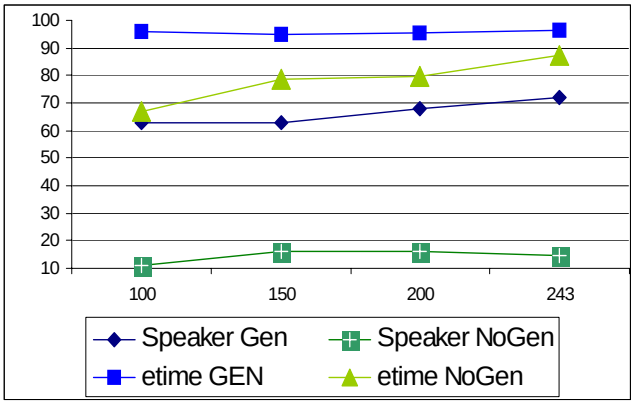

Figure 13: effect of training data quantity in the CMU seminar task : the generalisation-based version converges immediately to the optimal value for <*etime*> and shows a positive trend for <*speaker*>. The non NLP based version shows overfitting on <*speaker*> and slowly converges on <*etime*>

able to converge at optimal values with 100 examples only, while accuracy in $(LP)^2_{NG}$ slowly increases with more training material, even if it is not able to reach the same accuracy as $(LP)^2_G$, even when using half of the corpus for training.

Not surprisingly the role of shallow NLP in the reduction of data sparseness is more relevant for semi-structured or free texts (e.g. the CMU seminars) than on highly structured documents (e.g. HTML pages). Note that during the rule selection phase **(LP)²** is able to adopt the right level of NLP information for the task at hand: in an experiment on texts written in mixed Italian/English we used an English POS tagger that was completely unreliable on the Italian part of the input. $\textbf{(LP)}^2_G$ reached an effectiveness analogous to $\textbf{(LP)}^2_{NG}$'s, because the rules using the unreliable NLP information were automatically discarded.

Finally it is interesting to note the way rule pattern length affects accuracy. Some slots are insensible to it (e.g. *location*), while others definitely need a longer pattern. This is an important information to monitor as the window size strongly influences training time.

7. Conclusion and Future Work

(LP)² is a successful algorithm. On the one hand, it outperforms the other state of the art algorithms in two scientific experiments on two very popular IE tasks. In particular **(LP)²** definitely outperforms other algorithms making use of NLP information such as Rapier and Whisk (more than 9% and 20% in terms of accuracy respectively in the CMU experiment). This is mainly due to the tagging and correction based learning algorithm, as all the algorithms are able to exploit NLP information. Rapier uses a tagging approach equivalent to the best tagging step of **(LP)²**. Moreover, even if it is able to induce more expressive rules (e.g. making use of more powerful wild cards), it uses a randomly based compression rule mechanism for generalisation that tends to produce rules providing spurious results, [Ciravegna 2000b].

Whisk is able to use some more sophisticated NLP information, i.e., it is able to work also on the output of a parser. Unfortunately it uses multi-slot rules that increase data sparseness in the training phase, leading to very low recall [Ciravegna 2001b].

Concerning the non NLP-based algorithms, their tagging approach is generally equivalent to the best rule based tagging phase in **(LP)²**, but they tend to use more sophisticated machine learning approaches. BWI for example learns single tag rules that are largely equivalent to the best rules in **(LP)²**. Boosting, [Schapire 1998], is used to emphasize examples on which the learner is doing poorly in order to derive additional rules. The rule formalism includes a number of wildcards that contribute radically to the algorithm's experimental results (up to 85% of its effectiveness). BWI does not use any type of NLP preprocessing. **(LP)²** slightly outperforms BWI in the experiments. In our opinion this is due to the combined effect of the contextual tagging phase, of the correction phase and of the use of NLP for generalization, the latter – we believe - being the most relevant of the three.

For a more complete comparison with the state of the art algorithms, [Ciravegna 2000].

(LP)² is also very successful because it was the basis for building *Learning*Pinocchio, a tool for adaptive IE applications that is having a considerable commercial success. A number of applications have been developed for commercial companies and a number of licenses have been released to industrial companies for further application development, [Ciravegna 2001].

Future work on **(LP)²** will involve both the improvement of rule formalism expressiveness (use of wild cards) and the further use of shallow NLP for generalisation. Concerning the latter, from the one hand **(LP)²** will be used in a cascade with a Named Entity Recogniser (also implemented by using **(LP)²**). This will allow further generalisation over named entity classes (e.g., the speaker is always a person, therefore it is possible to generalise over such class in the rules). On the other hand **(LP)²** is compatible with forms of shallow parsing such as chunking. It would be possible to preprocess the texts with a chunker and to insert tags only at the chunk borders. This is likely to improve precision in border identification, reducing the need of correction.

Acknowledgments

I developed the naive version of $(LP)^2$ and *Learning*Pinocchio at ITC-Irst, Centro per la Ricerca Scientifica e Tecnologica, Trento, Italy. *Learning*Pinocchio is property of ITC-Irst, see http://ecate.itc.it:1025/cirave/LEARNING/home.html. Some parts of this paper have been previously published in [Ciravegna 2001b]. For the common parts: copyright of the Association for Computational Linguistics.

References

Califf M. E. (1998), *Relational Learning Techniques for Natural Language Information Extraction*, Ph.D. thesis, Univ. Texas, Austin, 1998 www/cs/utexas.edu/users/mecaliff

Claire Cardie (1997), `Empirical methods in information extraction', *AI Journal*, 18(4), 65-79, 1997.

Ciravegna F., Lavelli, A. and Satta, G. (2000), `*Bringing information extraction out of the labs: the Pinocchio Environment*', in ECAI2000, Proceeding of the 14th European Conference on Artificial Intelligence, W. Horn, ed., IOS Press.

Ciravegna, F., (2000b)'*Learning to tag for Information Extraction from Text'*, in F. Ciravegna, R. Basili, R. Gaizauskas (eds.) ECAI2000 Workshop on Machine Learning for IE, Berlin, (www.dcs.shef.ac.uk/~fabio/ecai-workshop.html)

Ciravegna, F., (2001) *'Adaptive Information Extraction from Text by Rule Induction and Generalisation'* Proceedings of 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), Seattle, August 2001.

Ciravegna, F., (2001b) *'Using Shallow NLP in Adaptive Information Extraction from Web-related Texts'* Proceedings of 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP 2001), Pittsburgh, June 2001.

Freitag D. (1998), `*Information Extraction from HTML: Application of a general learning approach*', Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98), 1998.

Freitag D. and McCallum, A. (1999): `*Information Extraction with HMMs and Shrinkage'*, AAAI-99 Workshop on Machine Learning for IE, Orlando, www.isi.edu/~muslea/RISE/ML4IE/

Freitag D., Kushmerick, N. (2000) `*Boosted wrapper induction'*, in F. Ciravegna, R. Basili, R. Gaizauskas (eds.) ECAI2000 Workshop on Machine Learning for IE, Berlin, 2000, (www.dcs.shef.ac.uk/~fabio/ecai-workshop.html)

Grishman R. (1997), `*Information Extraction: Techniques and Challenges*. In Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology, in M.T. Pazienza, (ed.), Springer Verlag.

Humphreys, K., Gaizauskas, R., Azzam, S., Huyck, C., Mitchell, B., Cunningham, H. and Wilks, Y. (1998): `*Description of the University of Sheffield LaSIE-II System as used for MUC-7*'. In Proc. of the 7th Message Understanding Conference.

Kushmerick N., Weld, D. and Doorenbos, R.,(1997) `*Wrapper induction for information extraction*', Proc. of 15th International Conference on Artificial Intelligence, IJCAI-97.

Mickalski, R. S., Mozetic, I., Hong, J., Lavrack, H. (1986): *The multi purpose incremental learning system AQ15 and its testing application to three medical domains*', in Proceedings of the 5th National Conference on Artificial Intelligence, Philadelphia: Morgan Kaufmann publisher.

S. Miller, M. Crystal, H. Fox, L. Ramshaw, R. Schwartz, R. Stone and R. Weischedel, (1998) `BBN: Description of the SIFT system as used for MUC-7', In *Proc. of the 7th Message Understanding Conference*, 1998 (www.muc.saic.com).

Muslea, I., Minton, S, and Knoblock, C, (1998) `*Wrapper induction for semi-structured, web-based information sources*', in Proc. of the Conference on Autonomous Learning and Discovery.

Schapire R. and Singer Y., (1998), `*Improved boosting algorithms using confidence-rated predictions*', in Proc. Eleventh Annual Conference on Computational Learning Theory.

Soderland, S., `*Learning information extraction rules for semi-structured and free text*', Machine Learning, (1), 1-44, 1999.

Yangarber, R., Grishman, R., Tapanainen, P. and Huttunen, Silja: `*Automatic Acquisition of Domain Knowledge for Information Extraction''* In Proc. of COLING 2000: The 18th International Conference on Computational Linguistics, Saarbrücken, 2000.