Natural Language Engineering **1** (1): 1–21. Printed in the United Kingdom

LearningPinocchio: Adaptive Information Extraction for Real World Applications

F. CIRAVEGNA

Department of Computer Science, University of Sheffield
Regent Court, 211 Portobello Street,
S1 4DP Sheffield, UK

A. LAVELLI

ITC-irst Centro per la Ricerca Scientifica e Tecnologica
via Sommarive 18, 38050 Povo (TN), Italy

(*Received 31 July 2002; revised 3 June 2003*)

Abstract

The new frontier of research on Information Extraction from texts is portability without any knowledge of Natural Language Processing. The market potential is very large in principle, provided that a suitable easy-to-use and effective methodology is provided. In this paper we describe LearningPinocchio, a system for adaptive Information Extraction from texts that is having good commercial and scientific success. Real world applications have been built and evaluation licenses have been released to external companies for application development. In this paper we outline the basic algorithm behind the scenes and present a number of applications developed with LearningPinocchio. Then we report about an evaluation performed by an independent company. Finally we discuss the general suitability of this IE technology for real world applications and draw some conclusion.

1 Introduction

Information Extraction from Text (IE) systems are generally used in real world applications as support for other technologies. For example in Knowledge Management (KM) applications, IE can provide support in a number of tasks (Maybury, 2001), including document annotation (Ciravegna *et al.*, 2002; Handschuh *et al.*, 2002) and ontology population (Bontcheva *et al.*, 2001). In this case the application environment poses a number of requirements on the characteristics of the needed IE tools, among them: portability with limited effort, portability across text types and possibility of tuning system results.

The first important requirement is **portability with minimum effort** and limited specific IE skills. The ideal situation is portability with an analyst's knowledge, i.e. knowledge on the domain. The use of Machine Learning (ML) has become a quite popular way to simplify porting. In MUC-oriented IE (MUC7, 1998), the use of ML has been approached mainly in an NLP-oriented perspective, i.e. in order

to reduce the amount of work done by IE experts in porting systems across free text based scenarios (Cardie, 1997; Yangarber *et al.*, 2000). In this case IE experts are still necessary. Other authors have addressed the problem of producing IE algorithms and systems adaptable by using only an analyst's knowledge (Kushmerick *et al.*, 1997; Califf, 1998; Muslea *et al.*, 1998; Freitag & McCallum, 1999; Soderland, 1999; Freitag & Kushmerick, 2000; Ciravegna, 2001a).

A second issue is **portability across text types**. In many application fields the increase in the use of Web technologies requires a wide range of text types to be considered: from free texts (technical reports, newspaper-like texts) to (semi)structured marked-up texts (e.g. partially marked-up or even highly structured HTML documents) and even a mixture of them (Ciravegna, 2001b). A system to be really adaptive should be portable across different text types without major recoding of system resources (i.e. again adaptable with minimum effort). Most IE literature has focused on IE from free newspaper-like texts (Cardie, 1997) and the technology developed is linguistically based. Such methodologies can be difficult to apply or even ineffective on highly structured marked-up documents. They are not able to cope with the extralinguistic elements (e.g. tags, document formatting) used to convey information in such documents. On the other hand, wrapper-like algorithms designed for highly structured documents (Kushmerick *et al.*, 1997; Freitag & McCallum, 1999; Muslea *et al.*, 1998; Freitag & Kushmerick, 2000) are largely ineffective on unstructured texts given their inability to overcome data sparseness due to linguistic variation (Ciravegna, 2001c). We believe that there is the need to develop methodologies able to fill the gap between the two approaches in order to cope with different text types.

A third issue in porting IE systems is **accuracy tuning**. Most of the current approaches are based on an adaptation phase in which users provide a set of texts with the associated relevant information manually extracted. Corpus annotation is just one part of the adaptation task, though, in building applications. Adaptation as a one-way process (from tagged examples to rules) is unlikely to provide optimised results for specific users, as different users will require results with different characteristics (e.g., high recall in some cases, high precision in others). There is the necessity of enabling users to modify the system behaviour so as to tailor the accuracy to the specific application needs. This requires two main features. On the one hand users should be enabled to evaluate results from both a quantitative and qualitative point of view. A test corpus with associated expected results is generally used as the basis for evaluation. The system is then run and statistics on both effectiveness and accuracy are presented to the user, together with details on correct matches and mistakes. Currently available technology is able to provide such information (Douthat, 1998; Cunningham *et al.*, 2002). On the other hand there is the necessity of actually influencing the system's behaviour effectively so to adapt it to the application needs. In case of occasional or inexperienced users, the issue arises of avoiding the use of technical or numerical concepts (such as precision and recall). This requires the ability of bridging the user's qualitative vision ("you are not capturing enough information") with the numerical concepts the system is able to manipulate (e.g. modifying error thresholds in order to obtain higher recall).

In this paper we describe LearningPinocchio, a system for adaptive IE whose design addresses the requirements mentioned above. In the paper we initially introduce the system and the used IE methodology, and describe a number of real world applications that we have developed. Then we present the experimental results obtained by LearningPinocchio both on standard testbeds and on real-world applications. Finally we discuss the system suitability to the requirements mentioned above and draw some conclusion and future work.

2 LearningPinocchio

LearningPinocchio is an adaptive system for IE, based on a kind of transformation-based like rule learning. Rules are learnt by generalising over a set of examples marked via XML tags in a training corpus. The system performs IE as tagging, i.e. the information is extracted by annotating texts using XML tags (e.g. the speaker in a seminar will be identified by surrounding it in the text with two tags: `<speaker>` and `</speaker>`).

IE is performed by applying a user-defined architecture. An **architecture** is always based on a **preprocessor** performing tokenization and - optionally - morphological analysis, part of speech tagging and gazetteer lookup (see Figure 1). Then the user can define a number of adaptive modules for IE. A **module** defines a scenario that summarizes the information to be extracted as a set of annotations (XML tags). A session of learning is associated with each module and a set of rules will be learnt for each module. Different modules in an architecture perform different steps in the IE process, for example we have developed modules for text zoning, named entity recognition and for more complex IE tasks.

The user interface of LearningPinocchio is web-based. All the interaction takes place via HTML forms.

Each module (and therefore each architecture) can work in three modes: training mode, test mode and production mode.

The **training mode** is used to induce rules, to learn how to perform IE in a specific application scenario. Input in training mode is:

- A module definition including a set of system parameters (e.g. accuracy thresholds; see below).
- A preprocessed training corpus tagged with XML tags identifying the information to be extracted by the system.

Output of the training phase is:

- A set of rules to be used when operating in production or testing mode.

A set of system parameters influences the learning algorithm in order to balance precision and recall. Among them, an accuracy threshold defines the maximum error rate a single induced rule must obtain in order to be considered valid. Roughly speaking a bigger error rate brings higher recall, a smaller one increases precision.

The **testing mode** is used to test a module on an unseen tagged corpus, so to understand how well it performs on a specific application. The provided corpus is

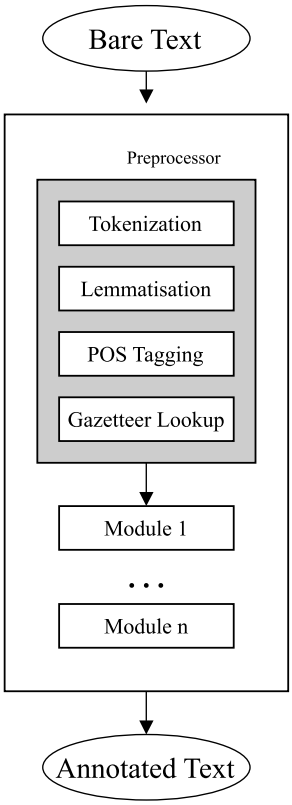

Fig. 1. An architecture in LearningPinocchio.

a collection of texts tagged by the user. During the test phase the module uses the set of rules induced during the training session to tag the provided corpus. Finally the system results are automatically compared with the user-provided results (i.e. the tags present in the corpus). Input is:

- A module with its induced rules.
- A test corpus tagged with XML tags identifying the information to be extracted by the system.

Output of the test phase is:

- The corpus tagged with XML tags by the module.
- A set of accuracy statistics on the test corpus: recall, precision and details on the mistakes the system makes.

During this phase it is possible to decide to retrain the module with different system parameters in order to tune its accuracy (e.g. to obtain more recall and/or more precision).

The **production mode** is used when an application is released. The module receives the text as tagged by the previous modules in the architecture and adds XML tags to the corpus.

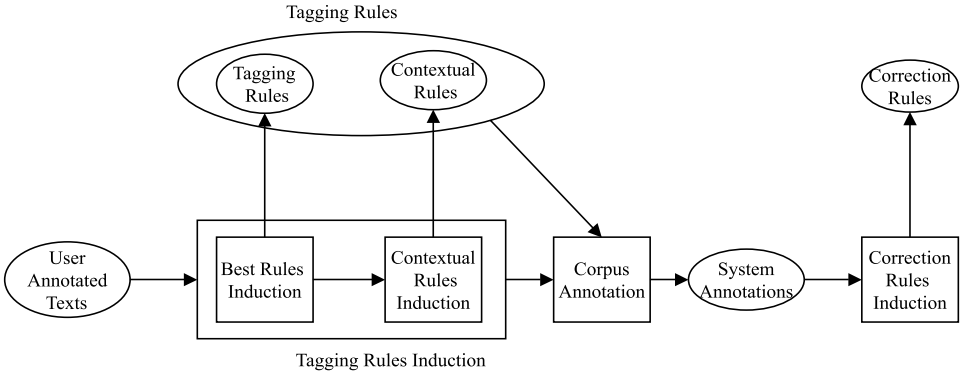

Fig. 2. Rule induction steps.

3 The Rule Induction Algorithm

The system is based on $(LP)^2$, a covering algorithm specifically designed for user-driven IE. $(LP)^2$ learns from a training corpus marked with XML tags. It induces symbolic rules that insert XML tags into texts in two steps (see Figure 2):

1. Sets of **tagging rules** are induced that insert a preliminary tagging.
2. **Correction rules** are induced that refine the tagging by correcting mistakes and imprecision.

All rules undergo a process of condition generalization based on shallow linguistic processing, as discussed below.

Sections 3.1 and 3.2 present and discuss the two steps mentioned above.

3.1 Inducing Tagging Rules

A tagging rule is composed of a left hand side, containing a pattern of conditions on a sequence of adjacent words, and a right hand side that is an action inserting an XML tag in the texts. Each rule inserts a single XML 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). The tagging rule induction algorithm uses positive examples from the training corpus for learning rules. Positive examples are the XML tags inserted by the user. All 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 piece of information stored in the $2*w$ word window is transformed into a condition in the initial rule pattern, e.g. if the third word in the window is "seminar", the condition on the third word in the pattern will be `Word=‘‘seminar’’`.

Index	**Condition**					**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			
5	pm	pm	Other	low	timeid	
6	will	will	Verb	low		

Fig. 3. Starting rule inserting `<stime>` in the sentence "the seminar at `<stime>` 4 pm will...".

Index	**Condition**					**Action**
	Word	Lemma	LexCat	Case	SemCat	Tag
3		at				`<stime>`
4			Digit			
5					timeid	

Fig. 4. A generalization for the rule in Figure 3. The pattern is relaxed in length (conditions on words 1, 2 and 6 have been removed) and only some of the constraints on the words 3, 4 and 5 are present.

Conditions are not limited to word matching. Conditions on linguistic information associated with each word are also inserted in the initial rule. Such information is provided by generic modules such as a morphological analyzer, a POS tagger and a user-defined dictionary (or a gazetteer). A *lexical item* (LexItem in the following) summarizes conditions on word (e.g., "companies"), lemma ("company"), lexical category (Noun), case information (lowercase), and a list of semantic classes from a user-defined dictionary or a gazetteer (if available). An initial rule with conditions on each LexItem is shown in Figure 3. The redundancy of the conditions in the initial rule is quite clear. `Word=''4''` always implies `LexCat=Digit`. An initial rule is never used as is. It is used as a seed to produce generalizations. Generalization consists in the generation of a set of rules derived by relaxing constraints in the initial rule pattern. Patterns are relaxed both by reducing their length and by removing constraints in the LexItems. Figure 4 shows one of the many possible generalizations for the rule in Figure 3. The last step of the algorithm is the selection of the best generalizations. Each generalization is tested on the training corpus and an `error rate` *E=wrong/matched* is calculated. Generalizations are accepted when they fall into one of the two groups described in the rest of the section: best rules and contextual rules.

3.1.1 Best Rules

Best rules are meant to be highly reliable annotation rules. A generalization becomes part of the set of best rules if:

1. it covers at least a minimum number of cases on the training corpus;
2. its error rate is less than a user-defined threshold.

The best rule list is kept constantly sorted so that better rules come first. Rules are sorted by decreasing number of matches (those with more matches come first). Rules with identical number of matches are sorted by increasing error rate (more precise rules come first). Rules with equal number of matches and equal error rate are sorted using the following criterion: if they report a number of matches below a user-defined threshold, then the one with less generic conditions is preferred (e.g. with conditions on words), otherwise the other one is preferred.

This heuristic has been chosen in order to favour rules supported by more evidence. Data sparseness, a well known phenomenon in Machine Learning for NLP, can produce rules with limited number of matches on the training corpus. When they are applied on the test corpus, they can produce unreliable results. Rules with more matches have more evidence supporting them and tend to be more reliable at testing time. This is also why we prefer rules with less generic conditions when they report a limited number of matches on the training corpus: matches on words tend to be more predictable than matches on other parts of the LexItem. For example a rule matching the sequence $\texttt{Word}_1$=‘‘in’’, $\texttt{Word}_2$=‘‘the’’, $\texttt{Word}_3$=‘‘afternoon’’ tends to overrecognize less than $\texttt{Case}_1$=low, $\texttt{LexCat}_2$=Art, $\texttt{Word}_3$=‘‘afternoon’’.

Rules fully subsumed by other better rules (i.e. their matches are also matched by another better rule) are removed from the list[1].

A maximum of k best rules are constantly kept. This means that when the list reaches size k, every new generalization is ignored unless it is able to enter the list in a position $p \leq k$. If it does, the rule previously in position k is removed from the list. The algorithm is summarized in Figure 5.

Rules retained at the end of the generalization process of one specific seed rule become part of the global *best rules pool*. When a rule enters such pool, all the instances covered by this rule are removed from the *positive examples pool*, i.e. covered instances will no longer be used for rule induction ($(LP)^2$ is a covering algorithm). Rule induction continues by selecting new instances and learning rules until the pool of positive examples is empty.

3.1.2 Contextual Rules

When applied on 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).

[1] Generalizations derived from the same seed rule cover the same portions of input when ineffective constraints are present.

```
method SelectRule(rule, currentBestPool)
  if (rule.matches≤MinimumMatchesThreshold)
     then return currentBestPool // i.e. reject(rule)
  if (rule.errorRate≥ErrorRateThreshold)
     then return currentBestPool // i.e. reject(rule)
  insert (rule, currentBestPool)
  sort(currentBestPool)
  removeSubsumedRules(currentBestPool)
  cutRuleListToSize(currentBestPool, k)
  return currentBestPool

method sort(ruleList)
  sort by decreasing number of matches
  if two rules have equal number of matches
     then sort by increasing error rate
  if two rules have same error rate and number of matches:
     then if one rule has more matches than a threshold
               then prefer the one with more generic conditions
          else prefer the other one
  return ruleList

method removeSubsumedRules(ruleList)
  loop for index1 from 0 to ruleList.size-1
        rule1=ruleList(index1)
        loop for index2 from index1+1 to ruleList.size
              rule2=ruleList(index2)
              if (subsumes(rule1, rule2))
                 then remove (rule2, ruleList)
  return ruleList

method subsumes(rule1, rule2)
  return (rule2.matches is a subset of rule1.matches)

method cutRuleListToSize(list, size)
  return subseq(list, 0, size)
```

Fig. 5. The algorithm for deciding if a generalization can be considered as a best rule.

Intuitively this is because the absolute reliability required for rule selection is strict, thus only a few of the induced rules will match it. 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 rules not selected as best rules and tries to constrain their application to make them more reliable. Constraints on rule application are derived by exploiting interdependencies among tags. As mentioned, $(LP)^2$ learns rules for inserting tags (e.g., `<speaker>`) independently from other tags (e.g., `</speaker>`). But tags are not independent. There are two ways in which they can influence each other: (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>`[2]. In general it is possible to use $\texttt{<tag}_x\texttt{>}$ to introduce $\texttt{<tag}_y\texttt{>}$. $(LP)^2$ as described so far is not able to use such contextual information, as it induces single tag rules. The context is reintroduced in $(LP)^2$ as an external constraint used to improve the reliability of inaccurate rules. In particular, low precision non-best rules are reconsidered for application in the context of tags inserted by the best rules. 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 (i.e., when the best rules are able to insert $\texttt{<tag}_x\texttt{>}$ but not $\texttt{</tag}_x\texttt{>}$). Such rules are called contextual rules. As an 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 tag `<speaker>`. Thus it will only be applied when the best rules have already recognized an open tag `<speaker>`, but not the corresponding `</speaker>`. The area of application is the part of the text following a `<speaker>` and within a distance less than or equal to the maximum length allowed for the present slot[3]. "Anchor tags" used as contexts can be found either to the right of the rule space application (as in the case above when the anchor tag is `<speaker>`), or to the left as in the opposite case (anchor tag is `</speaker>`). Acceptability for contextual rules is computed by using the same algorithm used for selecting best rules, but only matches in constrained contexts are counted.

In conclusion the sets of tagging rules $(LP)^2$ induces are both the best rules pool and the contextual rules. Figure 6 shows the whole algorithm for tagging rule induction[4].

3.2 Inducing Correction Rules

Tagging rules when applied on the training corpus report some imprecision in slot filler boundary detection. A typical mistake is for example "at `<time>` 4 `</time>` pm", where "pm" should be 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 applying tagging rules on the training corpus. Shift rules consider tags misplaced within a distance d from the correct position. Correction rules are similar to tagging rules, but (1) their patterns match also the tags inserted by 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 7. The induction and selection algorithm used for the best tagging rules is also used for shift rules: initial instance identification, generalization, test and selection of best k generalizations.

[2] In the following we just make examples related to the first case as it is more intuitive.
[3] The training corpus is used for computing the maximum filler length for each slot.
[4] Please note that in the presentation above the induction of contextual rules is described as separated from the best rule induction step for the sake of clarity. In the actual implementation it is interleaved with it, as detailed in Figure 6.

```
method InduceRules()
  loop for instance in initial-instances
        unless already-covered(instance)
        loop for rule in generalise(instance)
              test(rule)
              // The following is a a simplification of the SelectRule method
              // presented in Figure 5 where k=∞. The code for finite values
              // of k is trivial, but more complex to present.
              if best-rule?(rule)
                then insert(rule, bestrules)
                      cover(rule, initial-instances)
                else loop for tag in tag-list
                            if test-in-context(rule, tag, :right)
                              then select-contxtl(rule, tag, :right)
                            if test-in-context(rule, tag, :left)
                              then select-contxtl(rule, tag, :left)

method generalise(instance)
  currentRules={instance}
  loop for lexitem in instance.pattern
        generalizations=generalize(lexitem)
        newRules={}
        loop  for currentCondition in generalizations
               loop for rule in currentRules
                     newRule=copyRule(rule)
                     ReplaceCondition(newRule, currentCondition, lexitem)
                     newRules.add(newRule)
        currentRules.AddAll(newRules)
  return currentRules

method generalise(lexitem)  // it produces the allowed generalizations
  collect newCondition(lexitem.word)
  if (lexitem.lemma!=null)
    then collect newCondition(lexitem.lemma)
          if (lexitem.case!=null))
          then collect newCondition(lexitem.lemma, lexitem.case)
  if (lexitem.case!=null)
    then collect newCondition(lexitem.word, lexitem.case)
          collect newCondition(lexitem.case)
  if (lexitem.lexCat!=null)
    then collect newCondition(lexitem.lexCat)
  if (lexitem.semCat!=null)
    then collect newCondition(lexitem.semCat)
  return collected conditions
```

Fig. 6. The final version of the algorithm for rule tagging induction.

3.3 Extracting Information

In the testing phase information is extracted from the test corpus in four steps (Figure 8): initial tagging, contextual tagging, correction and validation. The best rules pool is initially used to tag the texts. Then contextual rules are applied in the

Word	Lemma	LexCat	Case	SemCat	Wrong Tag	Correct Tag
at	at	Prep	low			
4	4	Digit			</stime>	
pm	pm	Other	low	timeid		</stime>

Fig. 7. The condition of an initial correction rule. The action (not shown) shifts the tag from the wrong to the correct position.

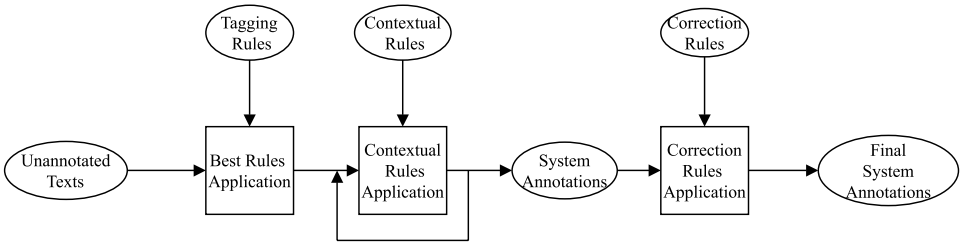

Fig. 8. The algorithm for extracting information.

context of the introduced tags. They are iteratively applied until no new tags are inserted, i.e. some contextual rules can match also tags inserted by other contextual rules. Then correction rules adjust some misplaced tags. Finally each tag inserted by the algorithm is validated. It is not meaningful to produce an open tag (e.g. <speaker>) without its corresponding closing tag (</speaker>) and vice versa, therefore uncoupled tags are removed in the validation phase.

4 Experimental Results

$(LP)^2$ was tested in a number of tasks in two languages: English and Italian. In each experiment $(LP)^2$ was trained on a subset of the corpus (some hundreds of texts, depending on the corpus) and the induced rules were tested on unseen texts. Here we report about results on two standard tasks for adaptive IE: the CMU seminar announcements and the Austin job announcements[5]. The first task consists of uniquely identifying speaker name, starting time, ending time and location in 485 seminar announcements (Freitag, 1998). We tested the system on a 10 run experiment using for each run a randomly selected half of the corpus for training and the rest of the corpus for testing. The system needed 56 minutes per run for training on a Sun Ultra5 workstation with 256M RAM using a window size of 4*2 words. 5 cycles of learning (performed on a partition not used for testing) were necessary for manually selecting the best error thresholds for rule selection. For the experiments we used a preprocessor that performed tokenization, part of speech tagging (we did not use gazetteer lookup here as no other researcher that tested

[5] Corpora available at `http://www.isi.edu/muslea/RISE/index.html`.

Table 1. *Results obtained on CMU seminar announcements.*

	$(LP)^2$			**Rapier**			**BWI**		
Slot	Prec	Rec	F(1)	Prec	Rec	F(1)	Prec	Rec	F(1)
speaker	87.0	70.0	77.6	80.9	39.4	53.0	79.1	59.2	67.7
location	87.0	66.0	75.1	91.0	60.5	72.7	85.4	69.6	76.7
stime	99.0	99.0	99.0	96.5	95.3	95.9	99.6	99.6	99.6
etime	94.0	97.0	95.5	95.8	96.6	96.2	94.4	94.9	93.9
All slots			86.0			77.3			83.9

	SRV			**Whisk**			**HMM**		
Slot	Prec	Rec	F(1)	Prec	Rec	F(1)	Prec	Rec	F(1)
speaker	54.4	58.4	56.3	52.6	11.1	18.3	77.9	75.2	76.6
location	74.5	70.1	72.3	83.6	55.1	66.4	83.0	74.6	78.6
stime	98.6	98.4	98.5	86.2	100.0	92.6	98.5	98.5	98.5
etime	67.3	92.6	77.9	85.0	87.2	86.1	45.7	97.0	62.1
All slots			77.1			64.9			82.0

their system on this corpus did use it: this makes the comparison fairer). Table 1 shows the results obtained by $(LP)^2$, and compares it with those obtained by other state-of-the-art algorithms. $(LP)^2$ scores the best results in the task. It outperforms both symbolic approaches (+8.7% wrt Rapier (Califf, 1998), +21% wrt to Whisk (Soderland, 1999), +2.1% wrt BWI (Freitag & Kushmerick, 2000)) and statistical approaches (+4% wrt HMM (Freitag & McCallum, 1999)). Moreover $(LP)^2$ is the only algorithm whose results never go below 75% on any slot (second best is BWI: 67.7%). The results shown in Table 1 for algorithms other than $(LP)^2$ are taken from (Freitag & Kushmerick, 2000). We added the comprehensive ALL SLOTS figure, as it allows better comparison among algorithms. It was computed by:

$$\frac{\sum_{slot}(F\text{-}measure * NumberOfPossibleSlotFillers)}{\sum_{slot} NumberOfPossibleSlotFillers} * 100$$

A second task concerned IE from 300 Job Announcements taken from the newsgroup `misc.jobs.offered` (Califf, 1998). The task consists of identifying for each announcement: message id, job title, salary offered, company offering the job, recruiter, state, city and country where the job is offered, programming language, platform, application area, required and desired years of experience, required and desired degree, and posting date. We performed the same type of 10 run based experiments using the same preprocessor. The results obtained on such task are reported in Table 2. $(LP)^2$ outperforms both Rapier and Whisk (Whisk obtained lower accuracy than Rapier (Califf, 1998)). We cannot compare $(LP)^2$ with BWI as

Table 2. *Results obtained on the Jobs domain using half of the corpus for training.*

	$(LP)^2$			**Rapier**			**BWI**		
Slot	Prec	Rec	F(1)	Prec	Rec	F(1)	Prec	Rec	F(1)
id	100.0	100.0	100.0	98.0	97.0	97.5	100.0	100.0	100.0
title	54.0	37.0	43.9	67.0	29.0	40.5	59.6	43.2	50.1
company	79.0	66.0	71.9	76.0	64.8	70.0	88.4	70.1	78.2
salary	77.0	53.0	62.8	89.2	54.2	67.4			
recruiter	87.0	75.0	80.6	87.7	56.0	68.4			
state	80.0	90.0	84.7	93.5	87.1	90.2			
city	92.0	94.0	93.0	97.4	84.3	90.4			
country	70.0	96.0	81.0	92.2	94.2	93.2			
language	92.0	90.0	91.0	95.3	71.6	80.6			
platform	81.0	80.0	80.5	92.2	59.7	72.5			
application	86.0	72.0	78.4	87.5	57.4	69.3			
area	70.0	64.0	66.9	66.6	31.1	42.4			
req-years-e	79.0	61.0	68.8	80.7	57.5	67.1			
des-years-e	67.0	55.0	60.4	94.6	81.4	87.5			
req-degree	90.0	80.0	84.7	88.0	75.9	81.5			
des-degree	90.0	51.0	65.1	86.7	61.9	72.2			
post date	99.0	100.0	99.5	99.3	99.7	99.5			
All slots			84.1			75.1			

the latter was tested on a very limited subset of slots. In summary, $(LP)^2$ reaches the best results on both the tasks.

5 Developing applications

LearningPinocchio was initially developed as a research prototype, but it quickly turned out to be suitable for real world applications. Recently it has been used in a number of industrial applications. Moreover evaluation licenses have been released to external companies for further testing and application building. In this section we report about some applications developed by us and on the evaluation accomplished by an independent IT company. In all the applications the architecture used was composed of a preprocessor performing tokenization, POS tagging and gazetteer lookup, and a module modeling the scenario.

5.1 IE from Résumés

The first application consisted in extracting information from professional résumés written in English. The application was developed for a Canadian company. LearningPinocchio is used on the results of a spider surfing the Web to retrieve professional résumés. The spider classifies résumés by topics (e.g. computer science). LearningPinocchio extracts the relevant information and outputs it to a database. Table 3 shows the results obtained in a blind test on 50 texts after training on 250. Note that such scenario does not imply a simple named entity recognition

Table 3. *Results of a blind test on 50 résumés.*

Tag	**Prec**	**Rec**	**F(1)**
Name	97	82	88.9
Street	96	71	81.6
City	90	90	90
Province	97	92	94.4
Email	92	71	80.1
Telephone	93	75	83.0
Fax	100	50	66.6
Zip code	100	90	94.7

task. A résumé may contain multiple names and addresses (e.g. previous work addresses, name of referees or thesis supervisors and their addresses). The system had to recognize the correct ones. The most recurrent errors were due to the presence of multiple addresses for a person. Sometimes the different addresses were wrongly merged (e.g. the street was assigned to the wrong city). This is a known limitation, as explained in Section 6.

Application development time for the IE task required about 24 person hours for scenario definition and revision (the scenario was refined by tagging some texts in different ways and discussing among annotators). Further 10 person hours were needed for tagging about 250 texts. The rule induction process took 72 hours on a Sun Ultra5 workstation. Finally the validation of system results required 4 person hours.

5.2 Financial News

This application was developed for Kataweb, one of the main Italian Internet portals (`http://www.kataweb.it`). It concerned the extraction of information from short financial news written in Italian and published on the portal pages.

The corpus consisted of 618 texts tagged with relevant labels. The texts were taken from the financial news published on the Kataweb portal in the period between 20 June 2000 and 29 June 2000. One half of the corpus was used for training the system and the other half was used as a test set. The relevant texts were tagged by Kataweb and we ran the system on the corpus and tuned the parameters in order to improve system performances. The average text length was about 200 words. The task was to extract all the occurrences of entities belonging to a set of 10 relevant categories: geographic areas related to stock markets (e.g. Far East), currencies, the citation of stock exchanges names and indices, name of organizations and companies, share names and their type and the market category of the company (e.g. telecommunication, blue chip, etc.). The overall number of relevant entities present in the corpus is about 17,100. The different categories are unevenly populated, ranging from a couple of hundreds for geographical areas to four thousands for company names. This task is somehow equivalent to a named entity recognition task in MUC (MUC7, 1998). In Figure 9 the application architecture is shown. The IE architec-

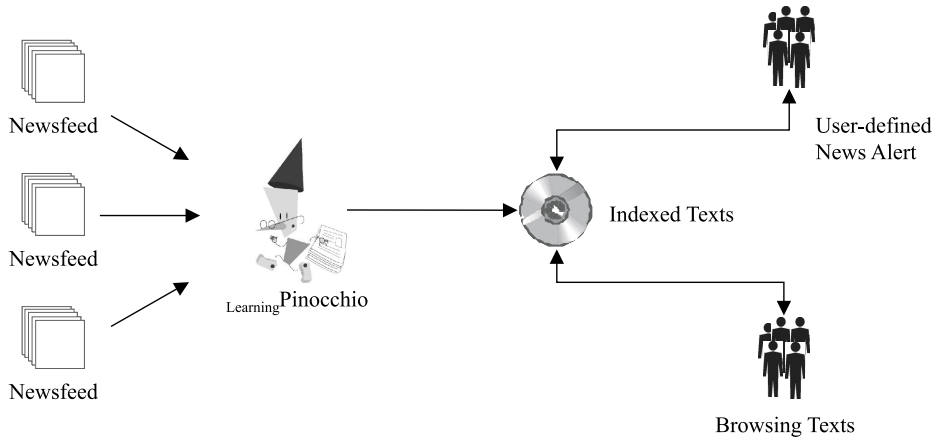

Fig. 9. The application on financial news.

Table 4. *Results of test on financial news.*

Tag	Prec	Rec	F(1)
Geographical area	74	67	71
Currency	84	85	85
Stock exchange name	90	86	88
Stock exchange index	98	94	96
Organization	91	81	86
Company name	89	80	84
Company type	91	91	91
Market category	84	85	84
All tags	90	84	87

ture used was composed of a preprocessor performing tokenization and gazetteer lookup, and a module modeling the scenario. The training of the system took about one day. The results on the test set are shown in Table 4. Kataweb is evaluating the performance of the system on the continuous flow of news published on the portal pages and the possibility of providing a service based on LearningPinocchio.

5.3 Classified Ads

A second application for Kataweb concerned IE from classified ads written in Italian and published on the portal pages. The corpus consisted of 2,214 texts tagged with relevant labels. The texts were taken from the classified ads published on the Kataweb portal on 29 August 2000. One half of the corpus was used for the training of the system and the other half was used has a test set. People from Kataweb tagged the relevant texts and we ran the system on the corpus and tuned the parameter settings in order to improve system performances. The average text length was about 35 words. The task was to extract the following information: name, email

Table 5. *Results of test on classified ads.*

Tag	Prec	Rec	F(1)
Advertiser	100	70	82
City	90	92	91
Telephone	88	97	92
Email	100	43	60
Ad category	99	100	99
Car model	91	95	93
Optional features	96	73	83
Company type	78	21	33
Job offered	87	56	68
House type	96	73	83
All tags	90	85	87

address and telephone number of advertisers; ad's category (e.g. house selling, car selling), city of reference (e.g. where the house is located). Then according to the type of ad, the system had to look for: brand model and optional features for cars, type of houses, kind of job offered by what type of company, etc. The overall number of relevant entities present in the corpus is about 13,400. The different categories are unevenly populated, ranging from a dozen email addresses to a couple of thousands of telephone numbers. The architecture used was composed of a preprocessor performing tokenization and gazetteer lookup, and a module modeling the scenario. The results of a blind test are shown in Table 5. This application is currently under final test at the customer's site.

5.4 Comparing with other systems

Bolesian (currently CGEY), a Dutch IT company, performed a survey of three IE state-of-the-art tools in a number of practical tasks: two adaptive and one that required manual development of rules. The purpose of the survey was not the selection of the best tool. They wanted to gain more insight in the use and working of such tools and to get a feeling of the kind and amount of effort involved in their use. In such evaluation they used a list of criteria ranging from ease of use of a tool and suitability to product costs and supplier characteristics (Joosen *et al.*, 2001). For the evaluation they used two IE tasks on documents about deeds of conveyance from the Dutch land-registration office. The data set was split into three parts:

- a training set of 225 texts, used as training input (split in four to determine the effect of the amount of training data on the accuracy of a tool);
- a tuning set of 50 texts, used to tune the learning algorithm and/or the rules developed;
- a test set of 62 texts, used to determine the actual precision and recall in the given domain.

Table 6. *Results of test accomplished by Bolesian.*

Tag	**Prec**	**Rec**	**F(1)**
Notarial Section	100	100	100
Date	91	100	95
Notary	69	97	81
Substitute	33	100	50
Place	93	98	95

The results of the evaluation for LearningPinocchio are shown in Table 6. In Figure 10 the results of the comparison of the three IE tools are shown. It appears that a person was able to model the task using 50 examples only; this is probably why the learning curve for the manual system is quite flat and accuracy does not improve significantly considering more texts. The two systems based on Machine Learning needed more examples.

After learning on 225 texts, LearningPinocchio definitely outperformed the other adaptive tool and it was slightly outperformed by the one requiring handcrafted rules. The system however showed a steady learning curve at the increase of the dimension of the training set (see Figure 10). With a limited increase in the training set size the systems would have probably reached equivalent accuracy. The other adaptive system did not reach the half of precision and recall reached by Learning-Pinocchio.

Positive aspects of LearningPinocchio were the fast and relatively easy way of setting up a task and the limited amount of training data needed to get reasonable results. The requirements in terms of time and skill needed for setting up new applications were definitely reduced with respect to the manual system. On the downside, they noted the lack of means for manipulating the learning process and that LearningPinocchio provided little insight in the learning process. Manually adjusting or extending the extraction process afterwards was impossible. The fact that LearningPinocchio performs IE as tagging was considered a limitation with respect to the manual tool that performed template filling. All the development was done at the customer's site, where the customer itself installed LearningPinocchio and ported it to Dutch by connecting a Dutch preprocessor. This (among other experiences) shows that the tool is usable also by people different from the developers.

6 Suitability of Technology

The experience described above allows us to draw some conclusion on the suitability for real world applications of the adaptive IE technology behind LearningPinocchio. First and foremost we were surprised by the positive reaction by the market when the system became available. Initially we had just in mind a research prototype, we submitted a paper for a workshop, presented the system to people visiting our

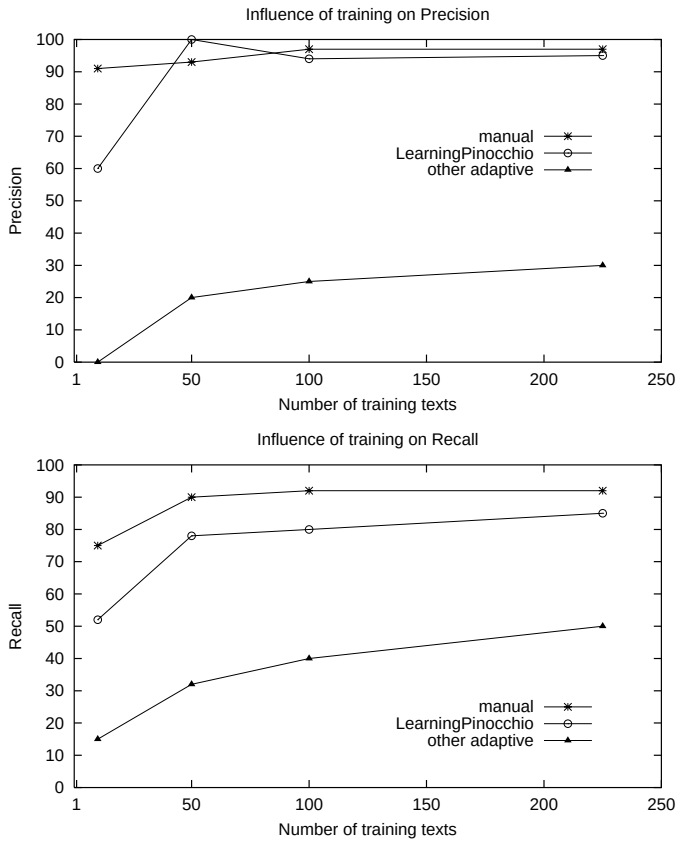

Fig. 10. Comparison between LearningPinocchio and two other IE systems.

institute and set up pages on our web server[6] to present the system. In two months (and far before the publication of the scientific paper) we were contacted by a dozen companies and started two of the applications above. All the activities cited in this paper were completed within eight months from the availability of the first release of the system. In our opinion this shows that the market is looking for easy-to-use IE tools . All the companies involved were Small Medium Enterprises (SMEs) and in our opinion this is not a surprise: blue chips can afford to buy expensive systems and to hire experts in IE, while SMEs cannot and are looking for systems that are easy to adapt.

The system proved to be easy to port to new applications, as other companies used the system and built applications without our support. No specific training was necessary for them except reading the manual. This fact meets the first requirement of easy adaptability mentioned in the introduction.

Applications were developed in three languages: Italian, English and Dutch. The

[6] `http://tcc.itc.it/research/textec/tools-resources/learningpinocchio.html`

porting to Dutch was carried out by Bolesian almost without support by us. The experimental results show that the system is effective on real world tasks. Bolesian's analysis also shows that the system outperforms other state-of-the-art commercial tools and reaches accuracy comparable to systems with handcrafted rules.

As shown in the scientific experiments above, the approach is able to cope with different types of texts without requiring modification of system resources. The types of texts analysed include free texts (résumés and short news), semi-structured texts (ads) and web pages. In this respect the system meets the second requirement (copying with different text types) mentioned in the introduction.

The experience also showed some of the limitations of the technology. The main limitation concerns the type of IE performed. The system performs IE as tagging, this means that it is able to spot the information in the text, but it is not able to relate it to the rest of the extracted information, i.e. it is not able to extract templates. A default strategy allows to build templates in a very naïve way (at most one event per text and one slot filler is supposed to exist). For this reason in case of multiple addresses in the résumés application the system was unable to assign the correct pairing. This is a limitation in certain types of applications. The results reported in the paper show however that the tagging-based approach is suitable for a number of commercial applications.

The second limitation was noted by Bolesian in their analysis: the learning algorithm works as a black box and it is difficult to understand how it works and it is impossible to manually modify the rules. This means that the system was basically designed with naïve users in mind and the needs of IE experts were not taken into considerations. IE experts would like to use their expertise to modify the rules so to reach some additional accuracy, but it was not possible.

As for tuning to specific application needs (e.g. tuning the balance between precision and recall), the system proved to be quite effective in letting the user understand how well it performed on a manually tagged corpus. In order to modify the system behaviour, users are required to change some numerical thresholds. It is not difficult to understand how to modify the thresholds: a bit of trial and error is enough for understanding how to modify such thresholds. Unfortunately threshold modification increases the sense of opacity of the system, because there is no clear relationship between the threshold values and the actual system behaviour if the user does not know how the learning algorithm works. In this respect the system is more dedicated to an expert. Naïve users were quite puzzled by thresholds. We think that some major changes are needed in this area, so to be able to address successfully the third requirement mentioned in the introduction.

7 Conclusion

In this paper we have presented LearningPinocchio, described some of the applications that have been developed with it and discussed its suitability to real world applications. In our opinion the system was successful both from a scientific point of view and from an applicative point of view. LearningPinocchio and the IE technology behind the scenes were designed with three main requirements in mind, namely

portability with limited effort, portability across text types and possibility of tuning system results. These requirements were derived from our experience in studying, implementing and delivering traditional IE systems (Ciravegna *et al.*, 2000). We believe that LearningPinocchio successfully addresses at least the first two of them, providing limited solution for the third. The result was a system highly usable also by non-IE experts. Experiences of application building at external companies and without our direct support prove this. The main limitation is the approach of IE as flat document tagging that makes difficult to relate information and therefore building templates. A second shortcoming was the opacity of the system with respect to the user: it was difficult to understand the logic of learning and therefore the system behaviour was difficult to modify. Future work includes new application building and the development of a new methodology for system result presentation in line with (Ciravegna *et al.*, 2002). We intend also to introduce the possibility of editing the induced rule sets. Finally we are addressing the extension to template-based IE learning.

Acknowledgments

The work presented in this paper was carried out while the first author was at ITC-irst. Fabio Ciravegna is currently supported by EPSRC AKT, grant GR/N15764/01, `http://www.aktors.org`. More details on LearningPinocchio can be found at `http://tcc.itc.it/research/textec/tools-resources/learningpinocchio.html`

The authors would like to thank the anonymous reviewers for the detailed work in revising the paper and in clarifying all the details of the approach.

References

Bontcheva, K., Brewster, C., Ciravegna, F., Cunningham, H., Guthrie, L., Gaizauskas, R., & Wilks, Y. 2001. Using HLT for Acquiring, Retrieving and Publishing Knowledge in AKT: Position Paper. *In: Proceedings of the ACL2001 Workshop on Human Language Technology and Knowledge Management.* held in conjunction with the meeting of the Association for Computational Linguistics.

Califf, Mary Elaine. 1998. *Relational Learning Techniques for Natural Language Information Extraction.* Ph.D. thesis, Univ. of Texas at Austin, http://www.cs.utexas.edu/users/mecaliff.

Cardie, C. 1997. Empirical Methods in Information Extraction. *AI Journal*, **18**(4), 65–79.

Ciravegna, F., Dingli, Alexiei, Petrelli, Daniela, & Wilks, Yorick. 2002. User-System Cooperation in Document Annotation based on Information Extraction. *In: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management, EKAW02.* Springer Verlag.

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

Ciravegna, Fabio. 2001b. Challenges in Information Extraction from Text for Knowledge Management. *IEEE Intelligent Systems and Their Applications*, **27**, 97–111. November.

Ciravegna, Fabio. 2001c. $(LP)^2$, an Adaptive Algorithm for Information Extraction from Web-related Texts. *In: Proceedings of the IJCAI-2001 Workshop on Adaptive Text Extraction and Mining held in conjunction with the 17th International Joint Conference on Artificial Intelligence.* Seattle, http://www.smi.ucd.ie/ATEM2001/.

Ciravegna, Fabio, Lavelli, Alberto, & Satta, Giorgio. 2000. Bringing information extraction out of the labs: the Pinocchio Environment. *In: Proceedings of the 14th European Conference on Artificial Intelligence*. IOS Press. Berlin.

Cunningham, H., Maynard, D., Tablan, V., Ursu, C., & Bontcheva, K. 2002. *"Developing Language Processing Components with GATE"*. http://www.gate.ac.uk.

Douthat, A. 1998. The Message Understanding Conference scoring software user's manual. *In: Proceedings of the 7th Message Understanding Conference*. http://www.itl.nist.gov/iaui/894.02/related_projects/muc/.

Freitag, D. 1998. Information Extraction From HTML: Application of a General Learning Approach. *Proceedings of the Fifteenth Conference on Artificial Intelligence AAAI-98*, 517–523.

Freitag, D., & Kushmerick, N. 2000. Boosted wrapper induction. *In:* Basili, R., Ciravegna, F., & Gaizauskas, R. (eds), *ECAI2000 Workshop on Machine Learning for Information Extraction*. http://www.dcs.shef.ac.uk/ fabio/ecai-workshop.html.

Freitag, D., & McCallum, A. 1999. Information Extraction with HMMs and Shrinkage. *In: AAAI-99 Workshop on Machine Learning for Information Extraction*.

Handschuh, S., Staab, S., & Ciravegna, F. 2002. S-CREAM - Semi-automatic CREAtion of Metadata. *In: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management, EKAW02*. Springer Verlag.

Joosen, M., Jongejan, P., Kersten, G.J., & Zopfi, E. 2001. Towards complexity measures: Guidelines for the feasibility of Information Extraction Projects. *In: Proceedings of the Dutch Conference on Artificial Intelligence*.

Kushmerick, N., Weld, D., & Doorenbos, R. 1997. Wrapper induction for information extraction. *In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1997*.

Maybury, Mark. 2001. Human Language Technologies for Knowledge Management: Challenges and Opportunities. *In: Proceedings of the ACL2001 Workshop on Human Language Technology and Knowledge Management*. held in conjunction with the meeting of the Association for Computational Linguistics.

MUC7. 1998. *Proceedings of the 7th Message Understanding Conference (MUC7)*. Nist. http://www.itl.nist.gov/iaui/894.02/related_projects/muc/.

Muslea, I., Minton, S., & Knoblock, C. 1998. Wrapper induction for semistructured web-based information sources. *In: Proceedings of the Conference on Automated Learning and Discovery (CONALD), 1998*.

Soderland, S. 1999. Learning Information Extraction Rules for Semi-structured and Free Text. *Machine Learning*, **34**(1), 233–272.

Yangarber, Roman, Grishman, Ralph, Tapanainen, Pasi, & Huttunen, Silja. 2000. Automatic Acquisition of Domain Knowledge for Information Extraction. *In: Proceedings of the 18th International Conference on Computational Linguistics*.