# An Analysis of Bootstrapping for the Recognition of Temporal Expressions

**Jordi Poveda**
TALP Research Center
Technical University of Catalonia (UPC)
Barcelona, Spain
jpoveda@lsi.upc.edu

**Mihai Surdeanu**
NLP Group
Stanford University
Stanford, CA
mihais@stanford.edu

**Jordi Turmo**
TALP Research Center
Technical University of Catalonia (UPC)
Barcelona, Spain
turmo@lsi.upc.edu

## Abstract

We present a semi-supervised (bootstrapping) approach to the extraction of time expression mentions in large unlabelled corpora. Because the only supervision is in the form of seed examples, it becomes necessary to resort to heuristics to rank and filter out spurious patterns and candidate time expressions. The application of bootstrapping to time expression recognition is, to the best of our knowledge, novel. In this paper, we describe one such architecture for bootstrapping Information Extraction (IE) patterns —suited to the extraction of entities, as opposed to events or relations— and summarize our experimental findings. These point out to the fact that a pattern set with a good increase in recall with respect to the seeds is achievable within our framework while, on the other side, the decrease in precision in successive iterations is succesfully controlled through the use of ranking and selection heuristics. Experiments are still underway to achieve the best use of these heuristics and other parameters of the bootstrapping algorithm.

## 1 Introduction

The problem of time expression recognition refers to the identification in free-format natural language text of the occurrences of expressions that denote time. Time-denoting expressions appear in a great diversity of forms, beyond the most obvious absolute time or date references (e.g. *11pm*, *February 14th, 2005*): time references that anchor on another time (*three hours after midnight*, *two weeks before Christmas*), expressions denoting durations (*a few months*), expressions denoting recurring times (*every third month*, *twice in the hour*), context-dependent times (*today*, *last year*), vague references (*somewhere in the middle of June*, *the near future*) or times that are indicated by an event (*the day G. Bush was reelected*). This problem is a subpart of a task called TERN (Temporal Expression Recognition and Normalization), where temporal expressions are first identified in text and then its intended temporal meaning is represented in a canonical format. TERN was first proposed as an independent task in the 2004 edition of the ACE conferences[1]. The most widely used standard for the annotation of temporal expressions is TIMEX (Ferro et al., 2005).

The most common approach to temporal expression recognition in the past has been the use of hand-made grammars to capture the expressions (see (Wiebe et al., 1998; Filatova and Hovy, 2001; Saquete et al., 2004) for examples), which can then be easily expanded with additional attributes for the normalization task, based on computing distance and direction (past or future) with respect to a reference time. This approach achieves an $F_1$-measure of approximately 85% for recognition and normalization. The use of machine learning techniques —mainly statistical— for this task is a more recent development, either alongside the traditional hand-grammar approach to learn to distinguish specific difficult cases (Mani and Wilson, 2000), or on its own (Hacioglu et al., 2005). The latter apply SVMs to the recognition task alone, using the output of several human-made taggers as additional features for the classifier, and report an $F_1$-measure of 87.8%.

[1] http://www.nist.gov/speech/tests/ace/



*Proceedings of the NAACL HLT Workshop on Semi-supervised Learning for Natural Language Processing*, pages 49–57,
Boulder, Colorado, June 2009. 

Bootstrapping techniques have been used for such diverse NLP problems as: word sense disambiguation (Yarowsky, 1995), named entity classification (Collins and Singer, 1999), IE pattern acquisition (Riloff, 1996; Yangarber et al., 2000; Yangarber, 2003; Stevenson and Greenwood, 2005), document classification (Surdeanu et al., 2006), fact extraction from the web (Paşca et al., 2006) and hyponymy relation extraction (Kozareva et al., 2008).

(Yarowsky, 1995) used bootstrapping to train decision list classifiers to disambiguate between two senses of a word, achieving impressive classification accuracy. (Collins and Singer, 1999) applied bootstrapping to extract rules for named entity (NE) classification, seeding the sytem with a few handcrafted rules. Their main innovation was to split training in two alternate stages: during one step, only contextual rules are sought; during the second step, the new contextual rules are used to tag further NEs and these are used to produce new spelling rules.

Bootstrapping approaches are employed in (Riloff, 1996), (Yangarber et al., 2000), (Yangarber, 2003), and (Stevenson and Greenwood, 2005) in order to find IE patterns for domain-specific event extraction. (Paşca et al., 2006) employ a bootstrapping process to extract general facts from the Web, viewed as two-term relationships (e.g [Donald Knuth, 1938] could be an instance of a "born in year" relationship). (Surdeanu et al., 2006) used bootstrapping co-trained with an EM classifier in order to perform topic classification of documents based on the presence of certain learned syntactic-semantic patterns. In (Kozareva et al., 2008), bootstrapping is applied to finding new members of certain class of objects (i.e. an "is-a" relationship), by providing a member of the required class as seed and using a "such as" type of textual pattern to locate new instances.

The recognition of temporal expressions is crucial for many applications in NLP, among them: IE, Question Answering (QA) and Automatic Summarization (for the temporal ordering of events). Work on slightly supervised approaches such as bootstrapping is justified by the large availability of unlabelled corpora, as opposed to tagged ones, from which to learn models for recognition.

## 2 Architecture

Figure 1 illustrates the building blocks of the algorithm and their interactions, along with input and output data.

The inputs to the bootstrapping algorithm are the unlabelled training corpus and a file of seed examples. The unlabelled corpus is a large collection of documents which has been tokenized, POS tagged, lemmatized, and syntactically analyzed for basic syntactic constituents (shallow parsing) and headwords. The second input is a set of *seed examples*, consisting of a series of token sequences which we assume to be correct time expressions. The seeds are supplied without additional features, and without context information.

Our bootstrapping algorithm works with two alternative views of the same target data (time expressions), that is: *patterns* and *examples* (i.e. an instance of a pattern in the corpus). A *pattern* is a generalized representation that can match any sequence of tokens meeting the conditions expressed in the pattern (these can be morphological, semantic, syntactic and contextual). An *example* is an actual candidate occurrence of a time expression. Patterns are generated from examples found in the corpus and, in its turn, new examples are found by searching for matches of new patterns. Both patterns and examples may carry contextual information, that is, a window of tokens left and right of the candidate time expression.

*Output examples* and *output patterns* are the outputs of the bootstrapping process. Both the set of *output examples* and the set of *output patterns* are increased with each new iteration, by adding the new candidate examples (respectively, patterns) that have been "accepted" during the last iteration (i.e. those that have passed the ranking and selection step).

Initially, a single pass through the corpus is performed in order to find occurrences of the seeds in the text. Thus, we bootstrap an initial set of *examples*. From then on, the bootstrapping process consists of a succession of iterations with the following steps:

1. Ranking and selection of examples: Each example produced during any of the previous iterations, 0 to $i-1$, is assigned a score (ranking). The top $n$ examples are selected to grow the set of output examples (selection) and will

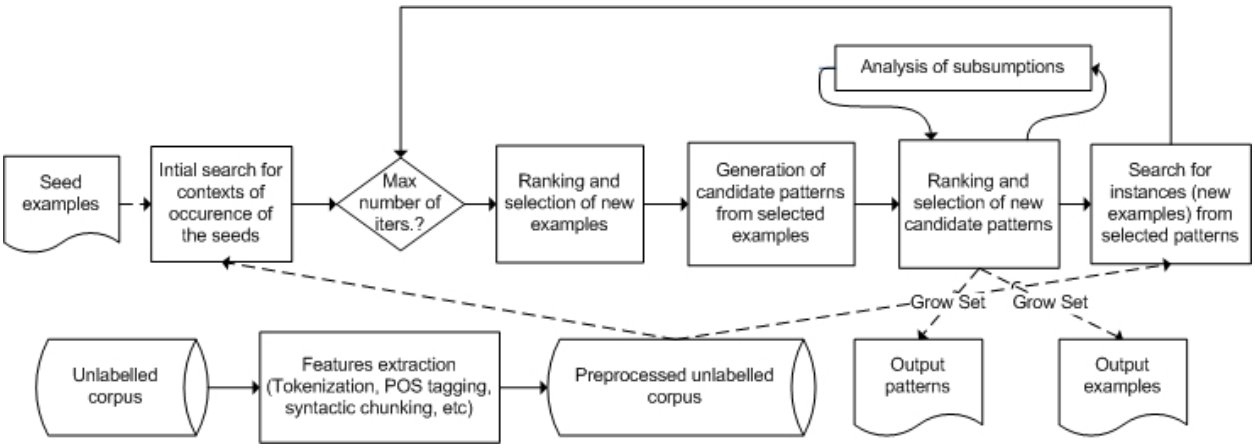


Figure 1: Block diagram of bootstrapping algorithm

be used for the next step. The details are given in Section 4.2.

2. Generation of candidate patterns: Candidate patterns for the current iteration are generated from the selected examples of the previous step (discussed in Section 3).
3. Ranking and selection of candidate patterns: Each pattern from the current iteration is assigned a score and the top $m$ patterns are selected to grow the set of output patterns and to be used in the next step (discussed in Section 4.1). This step also involves a process of analysis of subsumptions, performed simultaneously with selection, in which the set of selected patterns is examined and those that are subsumed by other patterns are discarded.
4. Search for instances of the selected patterns: The training corpus is traversed, in order to search for instances (matches) of the selected patterns, which, together with the accepted examples from all previous iterations, will form the set of candidate examples for iteration $i+1$.

Also, in order to relax the matching of patterns to corpus tokens and of token forms among themselves, the matching of token forms is case-insensitive, and all the digits in a token are generalized to a generic digit marker (for instance, "12-23-2006" is internally rewritten as "@@-@@-@@@@").

Even though our architecture is built on a traditional boostrapping approach, there are several elements that are novel, at least in the context of temporal expression recognition: a) our pattern representation incorporates full syntax and distributional semantics in a unified model (see Section 3); b) our pattern ranking/selection approach includes a subsumption model to limit redundancy; c) the formulae in our example ranking/selection approach are designed to work with variable-length expressions that incorporate a context.

## 3 Pattern representation

Patterns capture both the sequence of tokens that integrate a potential time expression (i.e. a time expression mention), and information from the left and right context where it occurs (up to a bounded length). Let us call *prefix* the part of the pattern that represents the left context, *infix* the part that represents a potential time expression mention and *postfix* the part that represents the right context.

The EBNF grammar that encodes our pattern representation is given in Figure 2. Patterns are composed of multiple *pattern elements* (PEs). A pattern element is the minimal unit that is matched against the tokens in the text, and a single pattern element can match to one or several tokens, depending on the pattern element type. A pattern is considered to match a sequence of tokens in the text when: first, all the PEs from the infix are matched (this gives the potential time expression mention) and, second, all the PEs from the prefix and the postfix are matched (this gives the left and right context information for the new candidate example, respectively). Therefore, patterns with a larger context window are more restrictive, because all of the PEs in the prefix and the postfix have to be matched (on top of the infix) for the pattern to yield a match.

We distinguish among token-level generalizations

```
pattern ::= prefix SEP infix SEP postfix SEP
            (modifiers)*
prefix ::= (pattern-elem)*
infix ::= (pattern-elem)+
postfix ::= (pattern-elem)*
pattern-elem ::= FORM "(" token-form ")" |
  SEMCLASS "(" token-form ")" |
  POS "(" pos-tag ")" | LEMMA "(" lemma-form ")" |
  SYN "(" syn-type "," head ")"  |
  SYN-SEM "(" syn-type "," head ")"
 modifiers ::= COMPLETE-PHRASE
```

Figure 2: The EBNF Grammar for Patterns

(i.e. PEs) and chunk-level generalizations. The former have been generated from the features of a single token and will match to a single token in the text. The latter have been generated from and match to a sequence of tokens in the text (e.g. a basic syntactic chunk). Patterns are built from the following types of PEs (which can be seen in the grammar from Figure 2):

1. Token form PEs: The more restrictive, only match a given token form.
2. Semantic class PEs: Match tokens (sometimes multiwords) that belong to a given semantic similarity class. This concept is defined below.
3. POS tag PEs: Match tokens with a given POS.
4. Lemma PEs: Match tokens with a given lemma.
5. Syntactic chunk PEs: Match a sequence of tokens that is a syntactic chunk of a given type (e.g. NP) and whose headword has the same lemma as indicated.
6. Generalized syntactic PEs: Same as the previous, but the lemma of the headword may be any in a given semantic similarity class.

The *semantic similarity class* of a word is defined as the word itself plus a group of other semantically similar words. For computing these, we employ Lin's corpus of pairwise distributional similarities among words (nouns, verbs and adjectives) (Lin, 1998), filtered to include only those words whose similarity value is above both an absolute (highest $n$) and relative (to the highest similarity value in the class) threshold. Even after filtering, Lin's similarities can be "noisy", since the corpus has been constructed relying on purely statistical means. Therefore, we are employing in addition a set of manually defined semantic classes (hardcoded lists) sensitive to our domain of temporal expressions, such that these lists "override" the Lin's similarity corpus whenever the semantic class of a word present in them is involved. The manually defined semantic classes include: the written form of cardinals; ordinals; days of the week (plus *today*, *tomorrow* and *yesterday*); months of the year; date trigger words (e.g. *day*, *week*); time trigger words (e.g. *hour*, *second*); frequency adverbs (e.g. *hourly*, *monthly*); date adjectives (e.g. *two- day*, *@@-week-long*); and time adjectives (e.g. *three-hour*, *@@-minute-long*).

We use a *dynamic window* for the amount of context that is encoded into a pattern, that is, we generate all the possible patterns with the same infix, and anything between 0 and the specified length of the context window PEs in the prefix and the postfix, and let the selection step decide which variations get accepted into the next iteration.

The modifiers field in the pattern representation has been devised as an extension mechanism. Currently the only implemented modifier is COMPLETE-PHRASE, which when attached to a pattern, "rounds" the instance (i.e. candidate time expression) captured by its infix to include the closest complete basic syntactic chunk (e.g. "LEMMA(end) LEMMA(of) SEMCLASS(January)" would match "the end of December 2009" instead of only "end of December" against the text "...By the end of December 2009, ..."). This modifier was implemented in view of the fact that most temporal expressions correspond with whole noun phrases or adverbial phrases.

From the above types of PEs, we have built the following types of patterns:

1. All-lemma patterns (including the prefix and postfix).
2. All-semantic class patterns.
3. Combinations of token form with sem. class.
4. Combinations of lemma with sem. class.
5. All-POS tag patterns.
6. Combinations of token form with POS tag.
7. Combinations of lemma with POS tag.
8. All-syntactic chunk patterns.
9. All-generalized syntactic patterns.

## 4 Ranking and selection of patterns and learning examples

### 4.1 Patterns

For the purposes of this section, let us define the *control set* $\mathcal{C}$ as being formed by the seed examples plus all the selected examples over the previous iterations (only the infix considered, not the context).

Note that, except for the seed examples, this is only *assumed correct*, but cannot be guaranteed to be correct (unsupervised). In addition, let us define the *instance set* $\mathcal{I}_p$ of a candidate pattern $p$ as the set of all the instances of the pattern found in a fraction of the unlabelled corpus (only infix of the instance considered). Each candidate pattern $\mathrm{pat}$ is assigned two partial scores:

1. A frequency-based score $\mathrm{freq\_sc(p)}$ that measures the coverage of the pattern in (a section of) the unsupervised corpus:
   $\mathrm{freq\_sc(p)} = \mathrm{Card}(\mathcal{I}_p \cap \mathcal{C})$
2. A precision score $\mathrm{prec\_sc(p)}$ that evaluates the precision of the pattern in (a section of) the unsupervised corpus, measured against the control set:
   $\mathrm{prec\_sc(p)} = \frac{\mathrm{Card}(\mathcal{I}_p \cap \mathcal{C})}{\mathrm{Card}(\mathcal{I}_p)}$

These two scores are computed only against a fraction of the unlabelled corpus for time efficiency. There remains an issue with whether *multisets* (counting each repeated instance several times) or *normal sets* (counting them only once) should be used for the instance sets $\mathcal{I}_p$. Our experiments indicate that the best results are obtained by employing multisets for the frequency-based score and normal sets for the precision score.

Given the two partial scores above, we have tried three different strategies for combining them:

- Multiplicative combination: $\lambda_1 \log(\epsilon_1 + \mathrm{freq\_sc(p)}) + \lambda_2 \log(\epsilon_2 + \mathrm{prec\_sc(p)})$
- The strategy suggested in (Collins and Singer, 1999): Patterns are first filtered by imposing a threshold on their precision score. Only for those patterns that pass this first filter, their final score is considered to be their frequency-based score.
- The strategy suggested in (Riloff, 1996):
  $\begin{cases} \mathrm{prec\_sc(p)} \cdot \log(\mathrm{freq\_sc(p)}) & \text{if } \mathrm{prec\_sc(p)} \geq \mathrm{thr} \\ 0 & \text{otherwise} \end{cases}$

#### 4.1.1 Analysis of subsumptions

Intertwined with the selection step, an analysis of subsumptions is performed among the selected patterns, and the patterns found to be subsumed by others in the set are discarded. This is repeated until either a maximum of $m$ patterns with no subsumptions among them are selected, or the list of candidate patterns is exhausted, whichever happens first. The purpose of this analysis of subsumptions is twofold: on the one hand, it results in a cleaner output pattern set by getting rid of redundant patterns; on the other hand, it improves temporal efficiency by reducing the number of patterns being handled in the last step of the algorithm (i.e. searching for new candidate examples).

In our scenario, a pattern $\mathrm{p_1}$ with instance set $\mathcal{I}_{p_1}$ is subsumed by a pattern $\mathrm{p_2}$ with instance set $\mathcal{I}_{p_2}$ if $\mathcal{I}_{p_1} \subset \mathcal{I}_{p_2}$. We make a distinction among "theoretical" and "empirical" subsumptions. Theoretical subsumptions are those that can be justified based on theoretical grounds alone, from observing the form of the patterns. Empirical subsumptions are those cases where in fact one pattern subsumes another according to the former definition, but this could only be detected by having calculated their respective instance sets a priori, which beats one of the purposes of the analysis of subsumptions —namely, temporal efficiency—. We are only dealing with theoretical subsumptions here. A pattern *theoretically subsumes* another pattern when either of these conditions occur:

- The first pattern is identical to the second, except that the first has fewer contextual PEs in the prefix and/or the postfix.
- Part or all of the PEs of the first pattern are identical to the corresponding PEs in the second pattern, except for the fact that they are of a *more general type* (element-wise); the remaining PEs are identical. To this end, we have defined a partial order of generality in the PE types (see section 3), as follows:
  FORM $\prec$ LEMMA $\prec$ SEMCLASS; FORM $\prec$ POS;
  SYN $\prec$ SYN-SEMC
- Both the above conditions (fewer contextual PEs and of a more general type) happen at the same time.

### 4.2 Learning Examples

An *example* is composed of the tokens which have been identified as a potential time expression (which we shall call the *infix*) plus a certain amount of left and right context (from now on, the *context*) encoded alongside the infix. For ranking and selecting

examples, we first assign a score and select a number $n$ of distinct *infixes* and, in a second stage, we assign a score to each context of appearance of an infix and select (at most) $m$ contexts per infix. Our scoring system for the infixes is adapted from (Paşca et al., 2006). Each distinct infix receives three partial scores and the final score for the infix is a linear combination of these, with the $\lambda_i$ being parameters: $\lambda_1 \mathrm{sim\_sc(ex)} + \lambda_2 \mathrm{pc\_sc(ex)} + \lambda_3 \mathrm{ctxt\_sc(ex)}$

1. A similarity-based score ($\mathrm{sim\_sc(ex)}$), which measures the semantic similarity (as per the Lin's similarity corpus (Lin, 1998)) of the infix with respect to set of "accepted" output examples from all previous iterations plus the initial seeds. If $w_1, \ldots, w_n$ are the tokens in the infix (excluding stopwords); $e_{j,1}, \ldots, e_{j,m_j}$ are the tokens in the $j$-th example of the set $E$ of seed plus output examples; and $\mathrm{sv}(x, y)$ represents a similarity value, the similarity $\mathrm{Sim}(w_i)$ of the $i$-th word of the infix wrt the seeds and output is given by $\mathrm{Sim}(w_i) = \sum_{j=1}^{|E|} \max(\mathrm{sv}(w_i, e_{j,1}), \ldots, \mathrm{sv}(w_i, e_{j,m_j}))$, and the similarity-based score of an infix containing $n$ words is given by $\frac{\sum_{i=1}^{n} \log(1+\mathrm{Sim}(w_i))}{n}$.
2. A phrase-completeness score ($\mathrm{pc\_sc(ex)}$), which measures the likelihood that the infix is a complete time expression and not merely a part of one, over the entire set of candidate example: $\frac{\mathrm{count(INFIX)}}{\mathrm{count(*INFIX*)}}$
3. A context-based score ($\mathrm{ctxt\_sc(ex)}$), intended as a measure of the infix's relevance. For each context (up to a length) where this infix appears in the corpus, the frequency of the word with maximum relative frequency (over the words in all the infix's contexts) is taken. The sum is then scaled by the relative frequency of this particular infix.

Apart from the score associated with the infix, each example (i.e. infix plus a context) receives two additional frequency scores for the left and right context part of the example respectively. Each of these is given by the relative frequency of the token with maximum frequency of that context, computed over all the tokens that appear in all the contexts of all the candidate examples. For each selected infix, the $m$ contexts with best score are selected.

# 5 Experiments

## 5.1 Experimental setup

As unsupervised data for our experiments, we use the NW (newswire) category of LDC's ACE 2005 Unsupervised Data Pool, containing 456 Mbytes of data in 204K documents for a total of over 82 million tokens. Simultaneously, we use a much smaller labelled corpus (where the correct time expressions are tagged) to measure the precision, recall and $\mathrm{F_1}$- measure of the pattern set learned by the bootstrapping process. This is the ACE 2005 corpus, containing 550 documents with 257K tokens and approx. 4650 time expression mentions. The labelled corpus is split in two halves: one half is used to obtain the initial *seed examples* from among the time expressions found therein; the other half is used for evaluation. We are requiring that a pattern captures the target time expression mention *exactly* (no misalignment allowed at the boundaries), in order to count it as a precision or recall hit.

We will also be interested in measuring the *gain in recall*, that is, the difference between the recall in the best iteration and the initial recall given by the seeds. Also important is the number of iterations after which the bootstrapping process converges. In the case where the same $\mathrm{F_1}$- measure mark is achieved in two experimental settings, earlier convergence of the algorithm will be prefered. Otherwise, better $\mathrm{F_1}$ and gain in recall are the primary goals.

In order to start with a set of seeds with high precision, we select them automatically, imposing that a seed time expression must have precision above a certain value (understood as the percentage, of all the appearances of the sequence of tokens in the supervised corpus, those in which it is tagged as a correct time expression). In the experiments presented below, this threshold for precision of the seeds is 90% —in the half of the supervised corpus reserved for extraction of seeds—. From those that pass this filter, the ones that appear with greater frequency are selected. For time expressions that have an identical digit pattern (e.g. two dates "@@ December" or two years "@@@@", where @ stands for any digit), only one seed is taken. This approach simulates the human domain expert, which typically is the first step in bootstrapping IE models

Unless specifically stated otherwise, all the experiments presented below share the following default settings:

- Only the first 2.36 Mbytes of the unsupervised corpus are used (10 Mbytes after tokenization and feature extraction), that is 0.5% of the available data. This is to keep the execution time of experiments low, where multiple experiments need to be run to optimize a certain parameter.
- We use the Collins and Singer strategy (see section 4.1) with a precision threshold of 0.50 for sub-score combination in pattern selection. This strategy favours patterns with slightly higher precision.
- The maximum length of prefix and postfix is 1 and 0 elements, respectively. This was determined experimentally.
- 100 seed examples are used (out of a maximum of 605 available).
- In the ranking of examples, the $\lambda_i$ weights for the three sub- scores for infixes are 0.5 for the "similarity-based score", 0.25 for "phrase-completeness" and 0.25 for "context-based score".
- In the selection of examples, the maximum number of new infixes accepted per iteration is 200, with a maximum of 50 different contexts per infix. In the selection of patterns, the maximum number of new accepted patterns per iteration is 5000 (although this number is never reached due to the analysis of subsumptions).
- In the selection of patterns, multisets are used for computing the instance set of a pattern for the frequency-based score and normal sets for the precision score (determined experimentally).
- The POS tag type of generalization (pattern element) has been deactivated, that is, neither all-POS patterns, nor patterns that are combinations of POS PEs with another are generated. After an analysis of errors, it was observed that POS generalizations (because of the fact that they are not lexicalized like, for instance, the syntactic PEs with a given headword) give rise to a considerable number of precision errors.
- All patterns are generated with COMPLETE-PHRASE modifier automatically attached. It was determined experimentally that it was best to use this heuristic in all cases (see section 3).

### 5.2 Variation of the number of seeds

We have performed experiments using 1, 5, 10, 20, 50, 100, 200 and 500 seeds. The general trends observed were as follows. The final precision (when the bootstrapping converges) decreases more or less monotonically as the number of seeds increases, although there are slight fluctuations; besides, the difference in this respect between using few seeds (20 to 50) or more (100 to 200) is of only around 3%. However, a big leap can be observed in moving from 200 to 500 seeds, where both the initial precision (of the seeds) and final precision (at point of convergence) drop by 10% wrt to using 200 seeds. The final recall increases monotonically as the number of seeds increases —since more supervised information is provided—. The final $F_1$-measure first increases and then decreases with an increasing number of seeds, with an optimum value being reached somewhere between the 50 and 100 seeds.

The largest gain in recall (difference between recall of the seeds and recall at the point of convergence) is achieved with 20 seeds, for a gain of 16.38% (initial recall is 20.08% and final is 36.46%). The best mark in $F_1$-measure is achieved with 100 seeds, after 6 iterations: 60.43% (the final precision is 69.29% and the final recall is 53.58%; the drop in precision is 6.5% and the gain in recall is 14.28%). Figure 3 shows a line plot of precision vs recall for these experiments. This experiment suggests that the problem of temporal expression recognition can be captured with minimal supervised information (100 seeds) and larger amounts of unsupervised information.

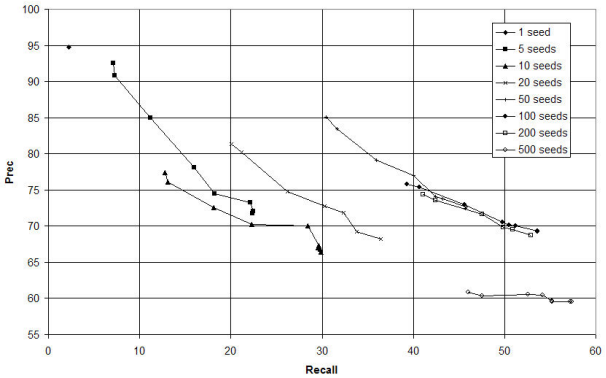


Figure 3: Effect of varying the number of seeds

### 5.3 Variation of the type of generalizations used in patterns

In these experiments, we have defined four differents sets of generalizations (i.e. types of pattern elements among those specified in section 3) to evaluate how semantic and syntactic generalizations contribute to performance of the algorithm. These four experiments are labelled as follows: NONE includes only PEs of the LEMMA type; SYN includes PEs of the lemma type and of the not-generalized syntactic chunk (SYN) type; SEM includes PEs of the lemma type and of the semantic class (SEMCLASS) type, as well as combinations of lemma with SEMCLASS PEs; and lastly, SYN+SEM includes everything that both SYN and SEM experiments include, plus PEs of the generalized syntactic chunk (SYNSEMC) type.

One can observe than neither type of generalization, syntactic or semantic, is specially "effective" when used in isolation (only a 3.5% gain in recall in both cases). It is only the combination of both types that gives a good gain in recall (14.28% in the case of this experiment). Figure 4 shows a line plot of this experiment. The figure indicates that the problem of temporal expression recognition, even though apparently simple, requires both syntactic and semantic information for efficient modeling.

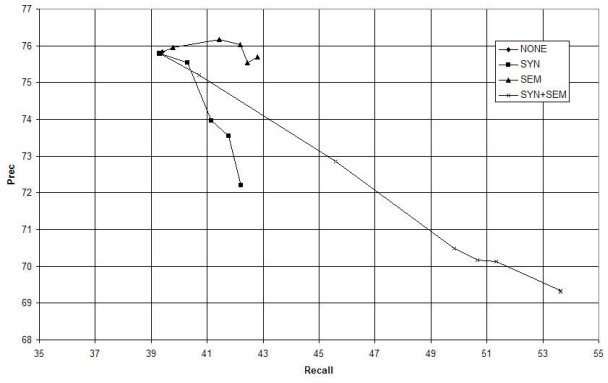


Figure 4: Effect of using syntactic and/or semantic generalizations

### 5.4 Variation of the size of unsupervised data used

We performed experiments using increasing amounts of unsupervised data for training in the bootstrapping: 1, 5, 10, 50 and 100 Mbytes of preprocessed corpus (tokenized and with feature extraction). The amounts of plain text data are roughly a fifth part, respectively. The objective of these experiments is to determine whether performance improves as the amount of training data is increased. The number of seeds passed to the bootstrapping is 68. The maximum number of new infixes (the part of an example that contains a candidate time expression) accepted per iteration has been increased from 200 to 1000, because it was observed that larger amounts of unsupervised training data need a greater number of selection "slots" in order to render an improvement (that is, a more "reckless" bootstrapping), otherwise they will fill up all the allowed selection slots.

The observed effect is that both the drop in precision (from the initial iteration to the point of convergence) and the gain in recall improve more or less consistently as a larger amount of training data is taken, or otherwise the same recall point is achieved in an earlier iteration. These improvements are nevertheless slight, in the order of between 0.5% and 2%. The biggest improvement is observed in the 100 Mbytes experiment, where recall after 5 iterations is 6% better than in the 50 Mbytes experiment after 7 iterations. The drop in precision in the 100 Mbytes experiment is 13.05%, for a gain in recall of 21.36% (final precision is 71.02%, final recall 52.84% and final $F_1$ 60.59%). Figure 5 shows a line plot of this experiment. This experiment indicates that increasing amounts of unsupervised data can be used to improve the performance of our model, but the task is not trivial.

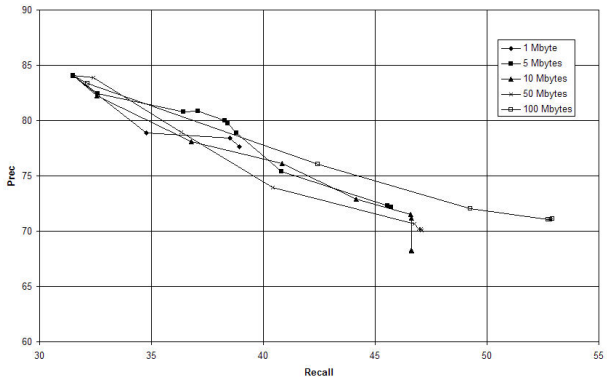


Figure 5: Effect of varying the amount of unsupervised training data

## 6 Conclusions and future research

We have presented a slightly supervised algorithm for the extraction of IE patterns for the recognition

of time expressions, based on bootstrapping, which introduces a novel representation of patterns suited to this task. Our experiments show that with a relatively small amount of supervision (50 to 100 initial correct examples or seeds) and using a combination of syntactic and semantic generalizations, it is possible to obtain an improvement of around 15%-20% in recall (with regard to the seeds) and $F_1$-measure over 60% learning exclusively from unlabelled data. Furthermore, using increasing amounts of unlabelled training data (of which there is plenty available) is a workable way to obtain small improvements in performance, at the expense of training time. Our current focus is on addressing specific problems that appear on inspection of the precision errors in test, which can improve both precision and recall to a degree. Future planned lines of research include using WordNet for improving the semantic aspects of the algorithm (semantic classes and similarity), and studying forms of combining the patterns obtained in this semi-supervised approach with supervised learning.

## References


M. Collins and Y. Singer. 1999. Unsupervised models for named entity classification. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 100–110, College Park, MD. ACL.

L. Ferro, L. Gerber, I. Mani, B. Sundheim, and G. Wilson. 2005. Tides 2005 standard for the annotation of temporal expressions. Technical report, MITRE Corporation.

E. Filatova and E. Hovy. 2001. Assigning time-stamps to event-clauses. In *Proceedings of the 2001 ACL Workshop on Temporal and Spatial Information Processing*, pages 88–95.

K. Hacioglu, Y. Chen, and B. Douglas. 2005. Automatic time expression labelling for english and chinese text. In *Proc. of the 6th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*, pages 548–559. Springer.

Z. Kozareva, E. Riloff, and E. Hovy. 2008. Semantic class learning from the web with hyponym pattern linkage graphs. In *Proc. of the Association for Computational Linguistics 2008 (ACL-2008:HLT)*, pages 1048–1056.

D. Lin. 1998. Automatic retrieval and clustering of similar words. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL-98)*, pages 768–774, Montreal, Quebec. ACL.

I. Mani and G. Wilson. 2000. Robust temporal processing of news. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 69–76, Morristown, NJ, USA. ACL.

M. Paşca, D. Lin, J. Bigham, A. Lifchits, and A. Jain. 2006. Names and similarities on the web: Fact extraction in the fast lane. In *Proceedings of the 21th International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, pages 809–816. ACL.

E. Riloff. 1996. Automatically generating extraction patterns from untagged text. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1044–1049. AAAI/MIT Press.

E. Saquete, R. Muñoz, and P. Martínez-Barco. 2004. Event ordering using terseo system. In *Proc. of the 9th International Conference on Application of Natural Language to Information Systems (NLDB)*, pages 39–50. Springer.

M. Stevenson and M. Greenwood. 2005. A semantic approach to IE pattern induction. In *Proceedings of the 43rd Meeting of the Association for Computational Linguistics*, pages 379–386. ACL.

M. Surdeanu, J. Turmo, and A. Ageno. 2006. A hybrid approach for the acquisition of information extraction patterns. In *Proceedings of the EACL 2006 Workshop on Adaptive Text Extraction and Mining (ATEM 2006)*. ACL.

J. M. Wiebe, T. P. O'Hara, T. Ohrstrom-Sandgren, and K. J. McKeever. 1998. An empirical approach to temporal reference resolution. *Journal of Artificial Intelligence Research*, 9:247–293.

R. Yangarber, R. Grishman, P. Tapanainen, and S. Hutunen. 2000. Automatic acquisition of domain knowledge for information extraction. In *Proceedings of the 18th International Conference of Computational Linguistics*, pages 940–946.

R. Yangarber. 2003. Counter-training in discovery of semantic patterns. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. ACL.

D. Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196, Cambridge, MA. ACL.