Efficient Knowledge Acquisition for Extracting Temporal Relations

Son Bao Pham and Achim Hoffmann
School of Computer Science and Engineering
University of New South Wales, Australia
{sonp,achim}@cse.unsw.edu.au

Abstract

We present KAFTIE – an incremental knowledge acquisition framework which utilizes expert knowledge to build high quality knowledge base annotators. Using KAFTIE, a knowledge base was built based on a small data set that outperforms machine learning algorithms trained on a much bigger data set for the task of recognizing temporal relations. In particular, this can be incorporated to bootstrap the process of labeling data for domains where annotated data is not available. Furthermore, we demonstrate how machine learning can be utilized to reduce the knowledge acquisition effort.

1 Introduction

Recent years have seen growing interests in temporal processing for many practical NLP applications. For example, question answering tasks try to find when and how long an event occurs or what events occur after a particular event.

Several works have addressed temporal processing: identification and normalization of time expressions (Mani and Wilson, 2000), time stamping of event clauses (Filatova and Hovy, 2001), temporally ordering of events (Mani et al., 2003), recognizing time-event relations in TimeML (Boguraev and Ando, 2005). At a higher level, these temporal expressions and their relations are essential for the task of reasoning about time, for example, to find contradictory information (Fikes et al., 2003).

In this emerging domain, there is a clear lack of a large annotated corpus to build machine learning classifiers for detecting temporal relations. We pursued the idea that an incremental knowledge acquisition approach could be used to develop a knowledge base of rules that utilizes experts' knowledge to overcome the paucity of annotated data. In fact, this approach could be combined nicely with the process of annotating data. When a new piece of data is annotated differently to what an existing KB proposes, the annotator specifies a justification for the decision in the form of a rule which is added to the knowledge base. Our experience shows that the time it takes to formulate a rule explaining why the data is annotated in a certain form is not much if the users have already spent time on deciding on the annotation. This is particularly true for complex tasks e.g. annotating relations between temporal expressions where it is not obvious whether there is any relation between temporal expressions. Importantly, rule formulation time does not depend on the size of the knowledge base.

Our incremental knowledge acquisition framework is inspired by Ripple Down Rules (Compton and Jansen, 1990) which allows users to add rules to the knowledge base (KB) incrementally while automatically ensuring that the knowledge base is always consistent. A new rule added to the KB is only applicable to those cases where the current knowledge base did not perform satisfactorily according to the users. This effectively avoids the adverse interactions of multiple rules in the KB.

We show that with our framework KAFTIE (Knowledge Acquisition Framework for Text classification and Information Extraction), we can quickly develop a large KB based on a small data set that performs better than machine learning approaches trained on a much bigger data set on the task of recognizing temporal relations.

2 TimeML

TimeML is intended as a Metadata Standard for Markup of events, their temporal anchoring and how they relate to each other (Pustejovsky et al., 2003). It aims at capturing the richness of time information by formally distinguishing events and temporal expressions in text. TimeML defines four temporal elements as tags with attributes: TIMEX3, SIGNAL, EVENT and LINK. TIMEX3 is modelled on TIMEX (Setzer and Gaizauskas, 2001) and TIMEX2 (Ferro et al., 2001). It marks up explicit temporal expressions such as times, dates, durations etc. SIGNAL is used to annotate function words that indicate how temporal objects are to be related to each other e.g. temporal connectives (*when*) or

Proceedings of the Australasian Language Technology Workshop 2005, pages 87–95,
Sydney, Australia, December 2005.

temporal prepositions (*on, during*). The EVENT tag covers elements in a text describing situations that occur or happen. Syntactically, EVENTs are tensed verbs, event nominals, stative adjectives and modifiers. The LINK tag encodes various relations that exist between temporal elements of a document which is divided into three subtypes namely: TLINK, SLINK and ALINK. TLINK is a temporal link representing a relation between an event and a time or between two events. SLINK represents a subordination relation between two events or an event and a signal. ALINK represents an aspectual relationship between two events.

In this paper, we report results on recognizing TLINK between an event and a time expression. The main reason for focusing on this subtask is to enable comparison with existing works. In fact, our approach is not geared towards this task and is general enough to be applicable to recognize other link types as well.

3 Knowledge Acquisition Methodology

In this section we present the basic idea of Ripple-Down Rules (Compton and Jansen, 1990) which inspired our approach. RDR was first used to build the expert system PEIRS for interpreting chemical pathology results (Edwards et al., 1993). PEIRS appears to have been the most comprehensive medical expert system yet in routine use, but all the rules were added by pathology experts without programming skill or knowledge engineering support whilst the system was in routine use. Ripple-Down Rules and some further developments are now successfully exploited commercially by a number of companies.

Knowledge Acquisition with Ripple Down Rules: Ripple Down Rules (RDR) is an unorthodox approach to knowledge acquisition. RDR does not follow the traditional approach to knowledge based systems (KBS) where a knowledge engineer together with a domain expert perform a thorough domain analysis in order to come up with a knowledge base. Instead a KBS is built with RDR incrementally, while the system is already in use. No knowledge engineer is required as it is the domain expert who repairs the KBS as soon as an unsatisfactory system response is encountered. The expert is merely required to provide an explanation for why in the given case, the classification should be different from the system's classification.

Suppose the system's classification was produced by some rule R_A. The explanation would refer to attributes of the case, such as patient data in the medical domain or a linguistic pattern matching the case in the natural language domain. The new rule R_B will only be applied to cases for which the provided conditions in R_B are true and for which rule R_A would produce the classification, if rule R_B had not been entered. I.e. in order for R_B to be applied to a case as an exception rule to R_A, rule R_A has to be satisfied as well. A sequence of nested exception rules of any depth may occur. Whenever a new exception rule is added, a difference to the previous rule has to be identified by the expert. This is a natural activity for the expert when justifying his/her decision to colleagues or apprentices. A number of RDR-based systems store the case which triggered the addition of an exception rule along with the new rule. This case, being called the *cornerstone case* of the new rule R, is retrieved when an exception to R needs to be entered. The cornerstone case is intended to assist the expert in coming up with a justification, since a valid justification must point at differences between the cornerstone case and the case at hand for which R does not perform satisfactorily.

Single Classification Ripple Down Rules: A single classification ripple down rule (SCRDR) tree is a finite binary tree with two distinct types of edges. These edges are typically called *except* and *if not* edges. See Figure 1. Associated with each node in a tree is a *rule*. A rule has the form: *if* α *then* β where α is called the *condition* and β the *conclusion*.

Cases in SCRDR are evaluated by passing a case to the root of the tree. At any node in the tree, if the condition of a node N's rule is satisfied by the case, the case is passed on to the exception child of N. Otherwise, the case is passed on the N's if-not child. The conclusion given by this process is the conclusion from the last node in the RDR tree which *fired*. To ensure that a conclusion is always given, the root node typically contains a trivial condition which is always satisfied. This node is called the *default* node.

A new node is added to an SCRDR tree when the evaluation process returns the wrong conclusion. The new node is attached to the last node evaluated in the tree provided it is consistent with the existing rules. If the node has no exception link, the new node is attached using an exception link, otherwise an *if not* link is used. To determine the rule for the new node, the expert formulates a rule which is satisfied by the case at hand.

4 Our KAFTIE framework

We use SCRDR for building knowledge bases in the KAFTIE framework. While the process of incrementally developing knowledge bases will eventu-

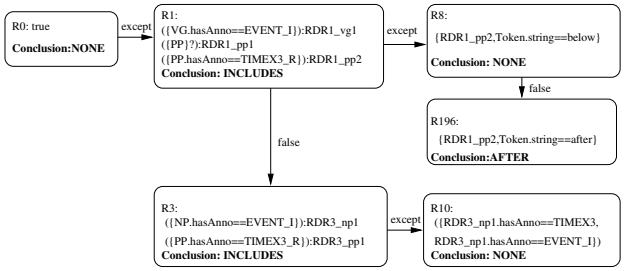

Figure 1: An extract of a KB for recognizing TLINK between an EVENT and a TIMEX3. Note that SCRDR is different from a decision tree: rules in internal nodes can be used to give conclusions to input cases.

ally lead to a reasonably accurate knowledge base, provided the domain does not drift and the experts are making the correct judgements, the time it takes to develop a good knowledge base depends heavily on the appropriateness of the used language in which conditions can be expressed by the expert.

Some levels of abstraction in the rule's condition is desirable to make the rule expressive enough in generalizing to unseen cases. To realize this, we use the idea of annotations where phrases that have similar roles (belong to the same concept) are deemed to belong to the same annotation type. Annotations contain the annotation type, the character locations of the beginning and ending position of the annotated text in the document, and a list of feature value pairs.

4.1 Rule description

A rule is composed of a condition part and a conclusion part. A condition is a regular expression pattern over annotations. It can also post new annotations over matched phrases of the pattern's subcomponents. The following is an example of a pattern which posts an annotation over the matched phrase:

({Noun}{VG.type==FVG}{Noun}):MATCH

This pattern would match phrases starting with a Noun annotation followed by a VG, which must have feature *type* equal to FVG, followed by another Noun annotation. When applying this pattern on a piece of text, MATCH annotations would be posted over phrases that match this pattern. As annotations have feature value pairs, we can impose constraints on annotations in the pattern by requiring that a feature of an annotation must have a particular value.

A piece of text is said to satisfy the rule condition if it has a substring that satisfies the condition pattern. The rule's conclusion contains the classification of the input text. In the task of recognizing temporal relations between a pair of temporal expressions (event or time), the conclusion is either the relation type or NONE.

Besides classification, our framework also offers an easy way to do information extraction. Since a rule's pattern can post annotations over components of the matched phrases, extracting those components is just a matter of selecting appropriate annotations. In this paper, the extraction feature is not used, though.

4.2 Annotations and Features

Built-in annotations: As our rules use patterns over annotations, the decision on what annotations and their corresponding features should be are important for the expressiveness of rules. Following annotations and features make patterns expressive enough to capture all rules we want to specify for various tasks.

We have Token annotations that cover every token with *string* feature holding the actual string, *category* feature holding the POS and *lemma* feature holding the token's lemma form.

As a result of the Shallow Parser module, which will be described in the next section, we have several forms of noun phrase annotations ranging from simple to complex noun phrases, e.g. NP (simple noun phrase), NPList (list of NPs) etc. All forms of noun phrase annotations are covered by a general Noun annotation.

There are also VG (verb groups) annotations with *type, voice, headVerbPos, headVerbString* etc. features and other annotations e.g. PP (prepositional phrase), SUB (subject), OBJ (object).

An important annotation that makes rules more general is Pair which annotates phrases that are bounded by commas or brackets. With this annotation, the following sentences:

> [PP On [TIMEX3 Monday TIMEX3] PP] [NP the company NP] [VG bought VG] ....
> [PP In [TIMEX3 recent months TIMEX3] PP] [NP a group of lenders NP] [Pair , led by Bank of America , Pair] [VG has extended VG] .....

could be covered by the following pattern:

{PP.hasAnno == TIMEX3}{NP}
({Pair})?{VG.type == FVG}

Every rule that has a non-empty pattern would post at least one annotation covering the entire matched phrase. Because rules in our knowledge base are stored in an exception structure, we want to be able to identify which annotations are posted by which rule. To facilitate that, we number every rule and enforce that all annotations posted by rule number x have the prefix RDRx_. Therefore, if a rule is an

exception of rule number x, it could use all annotations with the prefix RDRx_ in its condition pattern.

Apart from the requirement that an annotation's feature must have a particular value, we define extra constraints on annotations namely *hasAnno*, *hasString*, *underAnno*, *endsWithAnno*. For example, *hasAnno* requires that the text covered by the annotation must contain another specified annotation:

NP.hasAnno == TIMEX3

only matches NP annotations that has a TIMEX3 annotation covering its substring. This is used, for example, to differentiate a TIMEX3 in a noun group from a TIMEX3 in a verb group.

Custom annotations: Users could form new named lexicons during the knowledge acquisition process. The system would then post a corresponding annotation over every word in those lexicons. Doing this makes the effort of generalizing the rule quite easy and keeps the knowledge base compact.

4.3 The Knowledge Acquisition Process in KAFTIE

The knowledge acquisition process goes through a number of iterations. The user gives a text segment (e.g. a sentence) as input to the KB. The conclusion (e.g. classification) is suggested by the KB together with the *fired* rule R that gives the conclusion. If it is not the default rule, annotations posted by the rule R are also shown (see section 6.3) to help the user decide whether the conclusion is satisfactory.

If the user does not agree with the KB's performance, there are two options of addressing it: adding an exception rule to rule R or modifying rule R if possible. In either case, user's decision will be checked for consistency with the current KB before it gets committed to the KB.

To create a new exception rule, the user only has to consider why the current case should be given a different conclusion from rule R. This effort does not depend on the knowledge base size.

Modification of existing rules in the KB is not normally done with RDR as it is viewed that every rule entered to the KB has its reason for being there. However, we find that in many natural language applications it is desirable to modify previously entered rules to cover new cases.

To inspect results of a new exception rule or a modified rule R, the user can inspect the annotations posted by the rule of interest through a user interface shown in figure 2.[1] The user can quickly check the impact of the rule on a document or even on the whole training corpus. If the corpus is annotated, statistical performance of the rule will also be collected. It is found that a good GUI is necessary to productively create and test rules.

[1]The interface design is inspired by Boguraev's work.

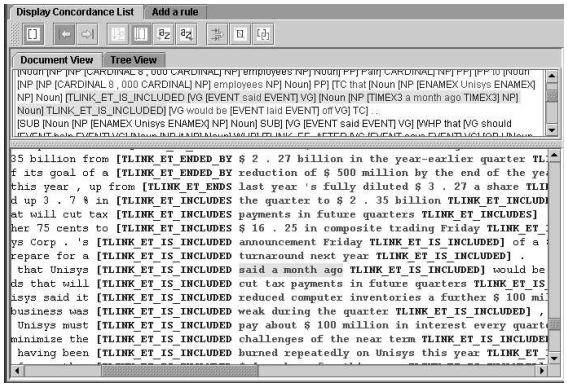

Figure 2: Display of Concordance list to quickly inspect results of rules.

5 Implementation

We built our framework KAFTIE using GATE (Cunningham et al., 2002). A set of reusable modules known as ANNIE is provided with GATE. These are able to perform basic language processing tasks such as POS tagging and semantic tagging. We use *Tokenizer, Sentence Splitter, Part-of-Speech Tagger and Semantic Tagger* processing resources from ANNIE. *Semantic Tagger* is a JAPE finite state transducer that annotate text based on JAPE grammars. Our rule's annotation pattern is implemented as a JAPE grammar with extensions to enable extra annotation feature constraints. We also developed additional processing resources for our tasks:

Shallow Parser: a processing resource using JAPE finite state transducer. The shallow parser module consists of cascaded JAPE grammars recognizing noun groups, verb groups, propositional phrases, different types of clauses, subjects and objects. These constituents are displayed hierarchically in a tree structure to help experts formulate patterns, see e.g. Figure 3. The Shallow Parser module could be refined as needed by modifying its grammars.

All these processing resources are run on the input text in a pipeline fashion. This is a preprocessing step which produces all necessary annotations before the knowledge base is applied on the text.

6 Experiments

We build a knowledge base using KAFTIE to recognize TLINK relations between an EVENT and a TIMEX3 using the TimeBank corpus.

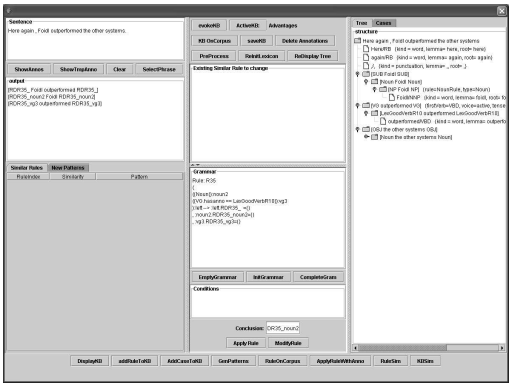

Figure 3: The interface to enter a new rule where the rule is automatically checked for consistency with the existing KB before it gets committed. Annotations including those created by the shallow parser module are shown in the tree in the *structure* box.

6.1 The TimeBank corpus

The TimeBank corpus is marked up for temporal expressions, events and basic temporal relations based on the specification of TimeML. Currently, the TimeBank corpus has 186 documents.

Excluding TIMEX3 in document's meta-data (doc creation time), the majority of TLINKs is between EVENTs and TIMEX3s within the same sentence. Hence, in all of our experiments, we focus on recognizing intra-sentential temporal relations.

The TimeBank annotation guidelines suggest distinctions among TLINK types between two EVENTs but do not explicitly specify how those types are different when it comes to relations between an EVENT and a TIMEX3. In fact, some of the TLINKs types between an EVENT and a TIMEX3 are hard to distinguish and a number of cases inconsistency is observed in the corpus. In this paper, we group similar types together: BEFORE and IBEFORE are merged, AFTER and IAFTER are merged and the rest is grouped into INCLUDES.

6.2 KAFTIE for extracting Temporal Relations

For the task of extracting EVENT-TIMEX3 temporal relations, we consider all pairs between an EVENT and a TIMEX3 in the same sentence and build a knowledge base to recognize their relations. The sentence containing the pair is used as the input to the knowledge base. As there could be more than one EVENT or TIMEX3 in the same sentence, we change the EVENT and the TIMEX3 annotations in focus to EVENT_I (instance) and TIMEX3_R (related_to) annotations respectively. This enables our rule's pattern to uniquely refer to the pair's arguments. For each pair, the KB's conclusion is the type of its temporal relation if there exists a relation between the pair's arguments or NONE otherwise.

6.3 How to build a Knowledge Base

The following examples are taken from the actual knowledge base (KB) discussed in section 6.4 and shown in figure 1. Suppose we start with an empty KB for recognizing temporal relations between an EVENT and a TIMEX3 within the same sentence. I.e. we would start with only a default rule that always produces a *NONE* conclusion. When the following sentence is encountered:

> Imports of the types of watches
> [VG [EVENT_I totaled EVENT_I] VG]
> [PP about $37.3 million PP]
> [PP in [NP [TIMEX3_R 1988 TIMEX3_R] NP]
> PP], ...

our empty KB would use the default rule to suggest the relation between EVENT_I and TIMEX3_R is NONE i.e. no relation exists. This can be corrected by adding the following rule to the KB:

> **Rule 1:**
> (({VG.hasanno==EVENT_I}):RDR1_vg1
> ({PP }?):RDR1_pp1
> ({PP.hasAnno == TIMEX3_R}):RDR1_pp2
>):RDR1_
> **Conclusion:** INCLUDES

Each of the component in the rule's pattern is automatically assigned a tag which will effectively post a new annotation over the matched token strings of the component if the pattern matches a text. New tags of rule i always start with $RDRi\_$. This rule would match phrases starting with a VG annotation which covers the EVENT_I annotation, followed by an optional PP annotation followed by a PP annotation covering the TIMEX3_R annotation. When the sentence containing the pair EVENT_I-TIMEX3_R is matched by this rule, the pair is deemed to be of INCLUDES relation type. Once matched, new annotations RDR1_vg1, RDR1_pp1 and RDR1_pp2 will be posted over the first VG, the first PP and the last PP in the pattern respectively. This is to enable exception rules to refer to the results of previously matched rules. Notice here that the first PP component is specified optional. It could be that the expert already *anticipates* future cases and make the current rule more general. Alternatively, experts always have a choice of modifying existing rule to cover new cases.[2] When we encounter this pair:

> The company's shares
> [**RDR1_vg1** [VG are [EVENT_I wallowing

[2] Automatic recommendation on which existing rules and how to modify to cover new cases is reported in (Pham and Hoffmann, 2005).

EVENT_I] far VG] **RDR1_vg1]**
[**RDR1_pp2** [PP below [NP their [TIMEX3_R 52-week TIMEX3_R] NP] PP] **RDR1_pp2]** high....

Rule **R1** *fires* and suggests that the pair has INCLUDES relation with the newly posted annotation highlighted in bold face. This is incorrect as there is no relation between the pair. The following exception rule is added to fix the misclassification:

Rule 8:[3]
({RDR1_pp2,Token.string==below}):RDR8_
Conclusion: NONE

This rule says that if the second PP matched by Rule 1 (RDR1_pp2) starts with a token string *below* then there is no relation between the pair. Notice that the sentence could have different PP annotations. As each rule posts unique annotations over the matched phrases, we can unambiguously refer to relevant annotations.

However, when we encounter the following case

It [**RDR1_vg1** [VG is deeply [EVENT_I discouraging EVENT_I] VG] **RDR1_vg1]**
[**RDR1_pp1** [PP for [NP the family NP] PP] **RDR1_pp1]**
[**RDR1_pp2** [PP after [NP [TIMEX3_R 22 months TIMEX3_R] NP] PP] **RDR1_pp2]** but ....

This case will be classified as INCLUDES type by Rule 1 while it should belong to AFTER type. We can add an exception to Rule1 catering for this case:

Rule 196:
({RDR1_pp2,Token.string==after}):RDR196_
Conclusion: AFTER

6.4 Experimental results

Out of 186 documents in the TimeBank corpus, we randomly took half of that as training data and keep the remaining half for testing. Using our KAFTIE framework, we built a knowledge base of 229 rules to recognize relations between an EVENT and a TIMEX3 in the same sentence using the training data.[4] The knowledge base uses NONE as its default conclusion. In other words, by default and initially when the KB is empty, all EVENT-TIMEX3 pairs are deemed not to have any TLINK relations. The overall results are shown in table 1 for two settings: *3 types* (BEFORE, INCLUDES and AFTER see section 6.1) and *without typing* - collapsing all TLINK types into one type, effectively detecting if the pair has a TLINK relation regardless of the type. While the accuracy reflects performance on the test data across all types including NONE, the F-measure is based on only TLINKs types, i.e. excluding NONE.[5] On the *without typing* setting, the built knowledge base achieved an F-measure of more than 75% and an accuracy of 86.7%.

Methods	3 types		w/o typing	
	F-m	Acc.	F-m	Acc.
KAFTIE	**71.3%**	**86.1%**	**75.4%**	**86.7%**
J48 (5-folds)	62.3%	78.7%	66.4%	81.2%
SMO (5-folds)	61.4%	77.4%	63.1%	78.7%

Table 1: Results on recognizing TLINKs for w/o typing and 3 types settings.

Comparison with machine learning: For comparison with standard machine learning approaches, we use Weka's J48 and SMO (Witten and Frank, 2000) as implementations for C4.5 and support vector machine algorithms respectively. To adapt machine learning algorithms to the task of extracting relations, we define the following feature representation which is capable of capturing the relation arguments (EVENTs and TIMEX3s) and the surrounding context. We break up the sentence containing the EVENT-TIMEX3 pair into five segments namely: spans of the two arguments (EVENT and TIMEX3), span between the two arguments and spans to the left/right of the left/right arguments. From each segment, we use token strings, token lemmas, parts-of-speech, bigrams of parts-of-speeches and all annotations from the Shallow Parser as features.

J48 and SMO are run using 5-fold cross validation. As the result could vary depending on the seed used for the cross validation, we report results averaged over 100 runs with different random seeds.As can be seen in table 1, the knowledge base built using our framework significantly outperforms standard J48 and SMO. In fact, our knowledge base with the initial 60 rules (as the result of seeing roughly 60 TLINK pairs) already outperforms J48 and SMO (see figure 4).

7 Reducing Knowledge Acquisition

In this section, we investigate how machine learning could be used to reduce the knowledge acquisition

[3]We only select some rules to show as examples, hence indices of rules are not consecutive

[4]To evaluate the TLINK recognition task alone, we use the EVENT and TIMEX3 annotations in the TimeBank corpus. That would also enable us to make a fair comparison with (Boguraev and Ando, 2005) as they also used *perfect* EVENT and TIMEX3 annotations from TimeBank.

[5]The NONE type occurs approximately 2.5 times more often than the TLINK types.

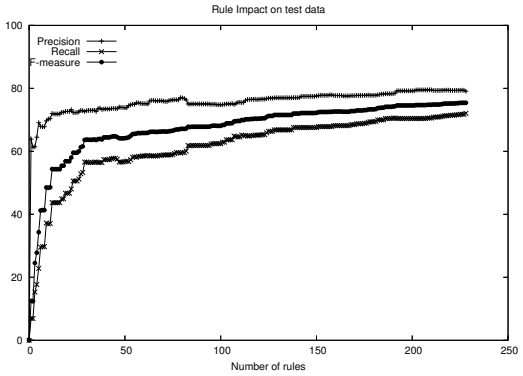

Figure 4: Impact of incrementally adding rules to the KB.

effort. A knowledge acquisition (KA) session corresponds to the creation of a new rule as a result of the KB performing incorrectly on a particular case. Notice that the KB's default rule is quite simple. It always returns the default conclusion which is *NONE* for the task of recognizing temporal relations. We conjecture that if we start with a better default rule then the number of knowledge acquisition sessions can be reduced. One way of doing it is to take some training data to train a classifier as the default rule. We carry out the following experiment:

We focus on the task of recognizing temporal relation in the *without typing* setting and use the exact training and test data from the previous section to build and test a KB respectively. The difference is that we now split the training data into two parts i.e. *ML data* and *KB data*. The *ML data* is used to train a classifier as a default rule in a KB while the *KB data* is used to add exception rules to the KB.

Instead of having real experts involved in the process of creating exception rules, we simulate it by consulting the KB built from the previous section, called *oracleKB*. For each example in the *KB data* that is misclassified by the current KB, we use the *fired* rule for the example from the *oracleKB* i.e. the rule that the *oracleKB* would use on the example.

Table 2 shows the results of using J48 and SMO which are trained on varying portions of the training data. Specifically, it shows the f-measure of the classifier alone, the KB with the classifier as the default rule on the test data as well as the number of rules in the KB. The number of rules in the KB reflects the number of KA sessions required for building the KB. All figures are averaged over 20 different runs using different seeds on splitting the training data into *ML data* and *KB data*.

As we increase the percentage of *ML data*, the number of KA sessions required gets smaller. Ideally, we want to minimize the number of KA sessions while maximize the f-measure. At one extreme end when the *ML data* is empty (0% of the training data), we effectively rebuild a KB in the same fashion as in the previous section with different orders of examples. The f-measure of 75.1% suggests that the performance of our approach are independent of the order of examples presented to the experts.

As it can be seen from table 2, the f-measures of the KBs with a classifier as the default rule follow the same trend. It first degrades as we start giving some data to the classifier and improves as more data is used to train the classifier. After certain thresholds, the f-measures start degrading again as we get less data for experts to add exception rules while the classifiers do not improve their performance.

Depending on the task and the classifier used, we can choose an appropriate amount of data to train a classifier as the default rule in order to substantially reduce the number of KA sessions involved while still achieve a reasonable performance. For example with J48 the best performance is at 72.5% when we use 60% of the training data for training J48 and the rest to add exceptions rules. Compared to a fully manual approach (using 0% data for the classifier), we achieve a 60% reduction in the number KA sessions.

As the *oracleKB* is built with the assumption that the default rule always returns *NONE*, all of the exception rules at level 1 (exception rules of the default rule) are rules covering *INCLUDES* instances. Even though rules at level 2 cover *NONE* instances, they have limited scopes because they were created as exceptions to level 1 rules. When the default rule gives an incorrect *INCLUDES* conclusion, it is likely that we would not be able to consult the *oracleKB* for an exception rule to fix this error.[6] It therefore suggests that if we use real experts, we could achieve a better result.

8 Discussions and Conclusions

Among the pioneering works on linking time and event expressions (Mani et al., 2003; Boguraev and Ando, 2005), only (Boguraev and Ando, 2005) reported results on publicly available data (TimeBank) which allows us to carry out performance comparison.[7] They use a robust risk minimization classifier utilizing a complex set of features including syntactic constructions derived from finite state

[6] A better simulation is to build another *oracleKB* with the default rule always returning *INCLUDES* conclusion.

[7] To the best of our knowledge, this is the only work done on TimeML compliant analyser using TimeBank corpus to date.

%	j48	j48 +kb	#KA	smo	smo +kb	#KA
0%	0	**75.1**	173	0	**75.1**	173
0.5%	28.4	**66.4**	146	27	**68.6**	151
1%	33.5	**65.2**	143	27.3	**71.6**	158
5%	45.3	**67.2**	138	54.3	**71.9**	142
10%	53.3	**69**	131	61.7	**72.4**	133
20%	62.9	**72.3**	117	64.3	**72.7**	118
30%	64.2	**71.9**	103	65.6	**72.7**	108
40%	66.6	**71.9**	91	66.1	**72**	93
50%	67.6	**72.2**	78	66.6	**71.8**	84
60%	68.5	**72.5**	65	67	**71.7**	70
70%	68.9	**71.7**	51	66.7	**71**	59
80%	69	**71.3**	38	66.3	**70.2**	44
90%	68	**70.3**	22	66	**68.5**	25

Table 2: Results of using j48/smo as the default rule for KBs averaged over 20 different random seeds. The first column is the percentage of the training data used to train a classifier (j48/smo). *j48* column contains the f-measures of the classifier alone on the test data. *j48+kb* column contains the f-measures of the KB with j48 as the default rule on the test data with the number of rules in the KB shown in column *#KA*. The last three columns are similar for the smo classifier.

analysis. We would assume that their features are geared towards the task, and presumably took substantial time to develop. Our rules created by the experts use annotations generated by a shallow parser. Importantly, our shallow parser is of a general purpose nature and does not generate extensive clause structures like in (Boguraev and Ando, 2005). In fact, we reuse the shallow parser developed for a different task in the technical papers domain (Pham and Hoffmann, 2004) with minor modifications. It indicates that our approach is not domain and task dependent as rules are crafted based on annotations generated by a general purpose shallow parser. It can be seen from table 3 that the KB built using our framework results in a better F-measure on all 3 settings of limiting the token distance between the EVENT and TIMEX3.

It should be noted that we used only half of the data for building the KB, while (Boguraev and Ando, 2005) used 80% of the data for training. Figure 4 shows the performance of our knowledge base on the test data as rules are incrementally added. Given the upwards trend of the graph as more rules are added, it is plausible that our KB would get to even higher performance had we used 80% of the available data for building the knowledge base.

We have shown that with our unconventional KA approach, namely RDR, we could quickly build a knowledge base that performs better than existing machine learning approaches while requiring much less data. As demonstrated in section 7, the process of building a KB can be boostraped by using machine learning algorithms. Looking at this from a different view, machine learning algorithms' performance can be improved by augmenting the KB built in KAFTIE.

Distance	Method	w/o typing
any	KAFTIE	**75.4**%
	Boguraev&Ando	74.8%
distance $\leq$ 16 tokens	KAFTIE	**77.2**%
	Boguraev&Ando	76.5%
distance $\leq$ 4 tokens	KAFTIE	**82.8**%
	Boguraev&Ando	81.8%

Table 3: Comparison with (Boguraev and Ando, 2005), who used 5-fold cross validation, i.e. 80% of the data for training while we only used 50% of the data to build the KB.

Independent of the knowledge base size, it took 7 minutes on average to create one rule. This includes the time needed to read the sentence to understand why there is a certain relation between the pair of an EVENT and a TIMEX3 as well as the time required to test the new rule before committed to the KB. If the users have to classify the pair's relation from scratch, when we do not have annotated data, then the actual time spent on creating a rule would be much less, as understanding the sentence takes most of the time. Importantly, we do not need to spend time engineering the features representation/selection for the task at hand which is usually done in machine learning approaches.

Thus our approach is particularly suitable for new tasks, when annotated data is not available or limited. Annotators could use KAFTIE to build an annotated corpus as well as a classifier at the same time. By requiring users to justify, in the form of rules, their decisions every time they annotate a case, it helps to annotate the corpus consistently. Furthermore, it also bootstraps the whole process as shown in section 6.4: after looking at half of the data to build a KB, the KB's accuracy on the other unseen half of the data is 86.7% in the '*without typing*' setting.

References

Branimir Boguraev and Rie Kubota Ando. 2005. TimeML-compliant text analysis for temporal reasoning. In *Proceedings of IJCAI*, UK.

Paul Compton and R. Jansen. 1990. A philosoph-

ical basis for knowledge acquisition. *Knowledge Acquisition*, 2:241–257.

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. 2002. Gate: An architecture for development of robust hlt applications. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics(ACL)*, Philadelphia, PA.

G. Edwards, P. Compton, R. Malor, A. Srinivasan, and L. Lazarus. 1993. PEIRS: a pathologist maintained expert system for the interpretation of chemical pathology reports. *Pathology*, 25:27–34.

Lisa Ferro, Inderjeet Mani, Beth Sundheim, and George Wilson. 2001. TIDES temporal annotation guidelines, MTR 01W0000041 MITRE technical report.

Richard Fikes, Jessica Jenkins, and Gleb Frank. 2003. JTP: A system architecture and component library for hydbrid reasoning. In *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando Florida, USA.

Elena Filatova and Eduard Hovy. 2001. Assigning time-stamps to event-clauses. In *Proceedings of the 10th Conference of the EACL*, Toulouse, France.

Inderjeet Mani and George Wilson. 2000. Robust temporal processing of news. In *Proceedings of the 38th annual meetings of the ACL*, Hong Kong.

Inderjeet Mani, Barry Schiffmann, and Jianping Zhang. 2003. Inferring temporal ordering of events in news. In *Proceedings of the NAACL*, Edmonton, Canada.

Son Bao Pham and Achim Hoffmann. 2004. Extracting positive attributions from scientific papers. In *7th International Conference on Discovery Science*, Italy.

Son Bao Pham and Achim Hoffmann. 2005. Intelligent support for building knowledge bases for natural language processing. In *Workshop on Perspective of Intelligent System Assistance*, Palmerston North, New Zealand.

J. Pustejovsky, J. Castano, R. Ingria, R. Sauri, R. Gaizauskas, A. Setzer, G. Katz, and D. Radev. 2003. TimeML: Robust specification of event and temporal expressions in text. In *AAAI Spring Symposium on New Directions in Question Answering.*, Standford, CA.

A. Setzer and R. Gaizauskas. 2001. A pilot study on annotating temporal relations in text. In *Workshop on temporal and spatial information processing, ACL*, Toulouse.

Ian H. Witten and Eibe Frank. 2000. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann.