Incremental Knowledge Acquisition for building sophisticated Information Extraction Systems with KAFTIE

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. The aim of our work is to develop a flexible and powerful Knowledge Acquisition framework that allows users to rapidly develop Natural Language Processing systems, including information extraction systems. In this paper we present our knowledge acquisition framework, KAFTIE, which strongly supports the rapid development of complex knowledge bases for information extraction. We specifically target scientific papers which involve rather complex sentence structures from which different types of information are automatically extracted. Tasks on which we experimented with our framework are to identify concepts/terms of which positive or negative aspects are mentioned in scientific papers. These tasks are challenging as they require the analysis of the relationship between the concept/term and its sentiment expression. Furthermore, the context of the expression needs to be inspected. The results so far are very promising as we managed to build systems with relative ease that achieve F-measures of around 84% on a corpus of scientific papers in the area of artificial intelligence.
Keywords: Incremental Knowledge Acquisition, Knowledge-based systems, Natural language processing.

1 Introduction

The development of sophisticated Natural Language Understanding systems, including information extraction, is usually a rather time-consuming and expensive process. For a new application, usually large parts of the system have to be redesigned in order to take relevant domain-dependent wordings into account. The particular task to be accomplished, e.g. the particular information to be extracted will also require an often substantial tailoring of the system.

In this paper, we present KAFTIE (Knowledge Acquisition Framework for Text classification and Information Extraction), an incremental knowledge acquisition framework that strongly supports the rapid prototyping of new NLP systems, that require classification of text segments and/or information extraction tasks. Our framework is inspired by the idea behind Ripple Down Rules [2] and allows for the incremental construction of large knowledge bases by providing one rule at a time. An expert just needs to monitor the system's performance on text and intervenes whenever the system does not perform as desired. The intervention will be based on a concrete text segment which the expert uses to specify rule conditions which are met by that text segment in order to formulate an exception rule to the rule that produced the undesirable system performances. Alternatively, the experts could modify an existing rule to cover the new case at hand provided the KB is still consistent.

Another strength of KAFTIE which makes it powerful and easy to use on NLP domain is its flexible annotation-based rule language and a customizable Shallow Parser.

We apply KAFTIE to the two tasks of extracting advantages and disadvantages of concepts or actions in technical papers. An advantage/disadvantage is detected when a positive/negative sentiment is expressed towards the concept or action. For example, given the following sentences:

There is some evidence that Bagging performs worse in low noise settings.
It is more efficient to use Knowledge Acquisition to solve the task.

The tasks are to detect that the algorithm *Bagging* and the action *to use Knowledge Acquisition to solve the task* have been mentioned with negative and positive sentiments respectively. These tasks are challenging and will be described in more details in the experiments section.

In this paper, we will first describe the underlying methodology of our framework and how to realize it on natural language domains. We will then illustrate the process by giving examples on how the knowledge base evolves. We will present experimental results and conclude with future works.

2 Methodology

In this section we present the basic idea of Ripple-Down Rules which inspired our approach.

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.

Say, 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.

This approach resulted in the expert system PEIRS used for interpreting chemical pathology results [6]. PEIRS appears to have been the most comprehensive medical expert system yet in routine use, but all the rules were added by a pathology expert without programming or knowledge engineering support or skill whilst the system was in routine use. Ripple-Down Rules and some further developments are now successfully exploited commercially by a number of companies.

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.

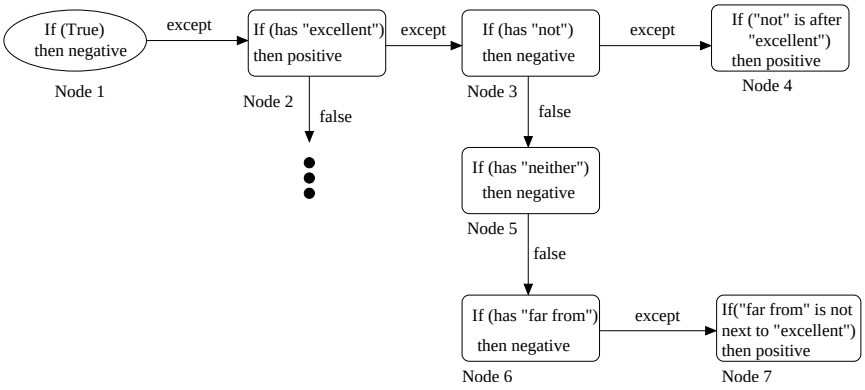

Fig. 1. An example SCRDR tree with simple rule language to classify a text into positive or negative class. Node 1 is the default node. A text that contains *excellent* is classified as *positive* by Node 2 as long as none of its exception rules fires, i.e., the text does not contain *not, neither* nor *far from* so Node 3,5,6 would not fire. A text that has *not excellent* is classified as *negative* by Node 3 while it is classified as *positive* by Node 4, if it contains *excellent but not*. If it contains *far from excellent* then it is classified as *negative* by Node 6.

3 Our approach

While the process of incrementally developing knowledge bases will eventually lead to a reasonably accurate knowledge base, provided the domain does not drift and the experts are making the correct judgments, 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 annotation where phrases that have similar roles are deemed to belong to the same annotation type.

3.1 Rule description

A rule is composed of a condition part and a conclusion part. A condition has an annotation pattern and an annotation qualifier. An annotation is an abstraction over string tokens. Conceptually, string tokens covered by annotations of the same type are considered to represent the same concept. Annotations contain the character locations of the beginning and ending position of the annotated text in the document along with the type of annotations and a list of feature value pairs.

The pattern is a regular expression over annotations. It can also post new annotations over matched phrases of the pattern's sub-components . The following is an example of a pattern which posts an annotation over the matched phrase:

({Noun} {VG} {Noun}):MATCH

This pattern would match phrases starting with a Noun annotation followed by a VG 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.

An annotation qualifier is a conjunction of constraints over annotations, including newly posted ones. An annotation constraint may require that a feature of that annotation must have a particular value:

VG.voice == active
Token.string == increase

A constraint can also require that the text covered by an annotation must contain (or not contain) another annotation or a string of text:

NP.hasAnno == LexGoodAdj
VG.has == outperform
VG.hasnot == not

A rule condition is satisfied by a phrase if the phrase matches the pattern and also the annotations qualifier is satisfied. For example we have the following rule condition:

Pattern: (({NP}):Noun1 {VG.voice == active} ({NP}):Noun2):MATCH

Annotations Qualifier: Noun2.hasAnno == LexGoodAdj

This pattern would match phrases starting with a NP annotation followed by a VG annotation (with feature *voice* having value *active*) followed by another NP annotation.

When a phrase satisfies the above rule condition, it must match the pattern which means that a MATCH annotation would be posted over the whole phrase and Noun1, Noun2 will be posted over the first and second NP in the pattern respectively. The second NP(Noun2) must also contain a LexGoodAdj annotation for the annotation qualifier to be satisfied. Notice that Noun1 is not used in the condition part but it could be used later in the conclusion part or in the exceptions of the current rule.

A piece of text is said to satisfy the rule condition if it has a substring that satisfies the condition. The following sentence matches the above rule condition as *useful* is annotated by the LexGoodAdj annotation:

[NP Parallelism NP][VG is VG][NP a useful way NP] to speed up computation.

with following new annotations posted:

[MATCH Parallelism is a useful way MATCH]
[Noun1 Parallelism Noun1]
[Noun2 a useful way Noun2]

but the following do not match:

(1) [NP Parallelism NP] [VG is VG] [NP a method NP] used in our approach.
(2) [NP Parallelism NP] [VG has been shown VG] [VG to be VG] very useful.

Sentence (1) matches the pattern, but it does not satisfy the annotation constraints. Sentence (2) does not match the pattern.

The rule's conclusion contains the classification of the input text. In the task of extracting positive attributions, it is *true* if the text mentions an advantage or a positive aspect of a concept/term and *false* otherwise.

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 phrase, extracting those components is just a matter of selecting appropriate annotations. For example, we can extract the concept/terms of interest whenever the case is classified as containing a positive aspect by specifying the target annotation. A conclusion of the rule with the condition shown above could be:

Conclusion: true

Concept Annotation: Noun1

The rule's conclusion contains a classification and an annotation to be extracted. It is deemed to be incorrect if either part of the conclusion is incorrect.

3.2 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. We experimentally tested the expressiveness of rules on technical papers and found that the 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 is also a VG (verb groups) annotation with *type, voice* features, several annotations for clauses 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:

The EM algorithm (Dempster , Laird , & Rubin , 1977) is effective.....
...the algorithm, in noisy domains, outperformed .....

could be covered by the following patterns respectively:

{Noun}({Pair})?{Token.lemma == be}{LexGoodAdj}
{Noun}({Pair})?{Token.lemma == outperform}

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 or annotations qualifier.

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.

3.3 KAFTIE

Our Knowledge Acquisition Framework for Text classification and Information Extraction (KAFTIE) allows users to easily and quickly develop knowledge bases to classify a text segment into a number of categories as well as extracting relevant information for those categories. For each category *C*, a SCRDR tree will be acquired which determines whether a text segment is classified as class *C* and which phrases in the text segment are extracted for the class *C*. Each SCRDR tree is composed of a number of nodes each containing a rule as described in section 3.1.

As our approach has the objective of ensuring the incremental improvement of a system's capabilities, all text segments which KAFTIE had already given conclusions (i.e. classification and information extracted) to the satisfaction of the user need to be treated in the same way after any modification to KAFTIE's knowledge base. To ensure this, KAFTIE checks automatically for any potential inconsistencies with previous conclusions of text segments for each modification that a user makes.

4 Implementation

We built our framework KAFTIE using GATE [4]. 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.

We also developed additional processing resources for our tasks:

Lemmatizer: a processing resource that puts a *lemma* feature into every Token annotation containing the lemma form of the token's string. Lemmatizer uses information from WordNet [7] and the result from the POS Tagger module.

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 2. 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.

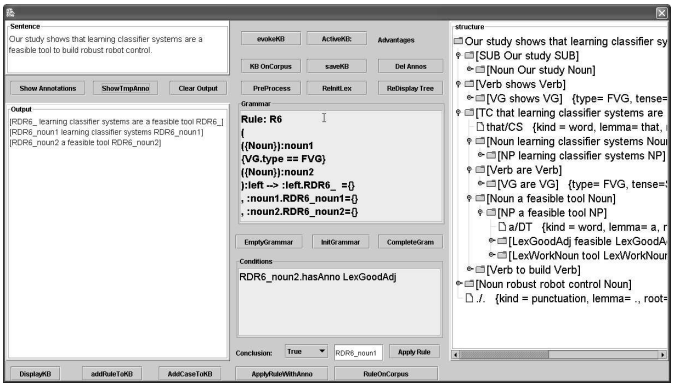

Fig. 2. The interface to enter a new rule where the rule is automatically checked for consistency with the existing KB before it gets committed. Annotations including those created by the shallow parser module are shown in the tree in the *structure* box.

5 Examples of how to build a Knowledge Base

The following examples are taken from the actual KB as discussed in section 6. Suppose we start with an empty knowledge base (KB) for recognizing advantages. I.e. the KB would only contain a default rule which always produces a *'false'* conclusion. When the following sentence is encountered:

Our study shows that learning classifier systems are a feasible tool to build robust robot control.

our empty KB would initially use the default rule to suggest it does not belong to the *Advantages* class. This can be corrected by adding the following rule to the KB:

Rule:R6
(({Noun}):RDR6_noun1 {VG.type==FVG} ({Noun.hasAnno == LexGoodAdj}):RDR6_noun2):RDR6_
Conclusion: true
Target Concept: RDR6_noun1

This rule would match phrases starting with a Noun annotation, followed by a VG annotation (with feature *type* equal to *FVG*) followed by a Noun annotation. Furthermore, the second Noun annotation must contain a LexGoodAdj annotation covering its substring. As there is a LexGoodAdj annotation covering the token *feasible*, the phrase *learning classifier systems are a feasible tool* is matched by **Rule6** and *learning classifier systems* is extracted as the concept of interest. When we encounter this sentence:

Given a data set, it is often not clear beforehand which **algorithm will yield the best performance**.

Rule **R6** suggests that the sentence mentions *algorithm* with a positive sentiment (the matched phrase is highlighted in boldface) which is not correct. The following exception rule is added to fix that:

Rule:R32 ({Token.lemma==which}{RDR6_}):RDR32_
Conclusion: false

This rule says that if the phrase matched by **Rule6** follows a *which* token , then the sentence containing it does not belong to *Advantages* class. However, when we encounter the following sentence

The latter approach searches for the subset of attributes over **which naive Bayes has the best performance**.

Rule **R6** suggests that *naive Bayes* has been mentioned with a positive sentiment but its exception rule, **R32**, overrules the decision because the phrase that matches **R6** (annotated by RDR6_) follows a token *which*. Obviously, *naive Bayes* should be the correct answer since the token *which* is used differently here than in the context in which **R32** was created. We can add an exception to **R32** catering for this case:

Rule:R56 ({Token.string==over} {RDR32_}):RDR56_
Conclusion: true
Target Concept: RDR6_noun2

6 Experimental Results

We have applied our framework KAFTIE to tackle two different tasks of recognizing sentences that contain positive and negative attributions of a concept/term as well as extracting the concept/term.[1] These tasks are challenging as the analysis of positive and negative sentiments towards a concept requires deep understanding of the textual context, drawing on common sense, domain knowledge and linguistic knowledge. A concept could be mentioned with a positive or negative sentiment in a local context but not in a wider context. For example,

We do not think that X is very efficient.

If we just look at the phrase *X is very efficient*, we could say that *X* is of positive sentiment, but considering a wider context it is not.

The task of extracting negative attributions may appear similar to the task of extracting positive attributions, but it is more difficult in this domain. One reason is because when authors mention a disadvantage of a concept, they sometimes talk about its positive aspects first. For example:

These innovations *proved very useful in increasing the CS efficiency , but* **inadequate for our needs**

The negative sentiment towards *These innovations* is not next to the term it attributes to. Therefore, recognizing and extracting *These innovations* with its negative sentiment is more difficult compared to its positive sentiment.

A corpus was collected consisting of 140 machine learning papers and journals downloaded from citeseer, and converted from PDF into text. Even though these papers are from the same domain, we have to stress that the topics they cover are rather diverse and include most areas within machine learning. We randomly selected 16 documents of different authors and grouped them into 2 corpora. The first corpus has 3672 sentences from 9 documents and the second corpus contains 4713 sentences from 7 documents.

For each task, a knowledge base is built using the first corpus based on the following procedure:

for *each document:*
  repeat
    apply the KB on all sentences of the document;
    add a rule to correct error for every sentence misclassified (for precision)
    add a rule to cover every missed sentence (for recall)
  until *no rule has been added*

The two KBs are built independently, at different times. Quality of the built KBs will be evaluated against the second corpus.

[1] The phrase *positive/negative sentiment or attribution* and *advantage/disadvantage* of a term are used interchangeably.

6.1 Experience with the Knowledge Acquisition Process

Building KBs using our framework is easy and quick. Most of the time spent is on going through the corpus, sentence by sentence, to determine if a sentence belongs to the class of interest but is not picked up by the KB. After the expert identifies the sentence of interest and the phrase to be extracted, it takes about 2 minutes to come up with a new rule or change an existing rule. This process involves first creating a pattern and then modifying it, if needed, until the pattern covers the sentence at hand.

While the particular choice of generality or specificity of a rule's pattern affects to some degree the convergence speed of the knowledge base towards complete accuracy, it is not that crucial in our approach. This is because suboptimal early choices are naturally patched up by further rules as the knowledge acquisition process progresses. Where a too general rule was entered a subsequent exception rule will be created, while a subsequent if-not rule patches a too specific rule. However, to help speed up the convergence of the KB, we do allow experts to change rules (usually to make it more general) while automatically ensuring the KB is consistent.

In fact, for the two tasks follow, we tried to formulate more general rules for the first task (Advantages) and more specific rules for the second task (Disadvantages). The performance of the two KBs on an unseen test corpus reveals a similar F-measure of above 80%. Looking at it more closely, we achieved a lower precision and a higher recall for the first task but a higher precision and a lower recall for the second task on the same corpus. This is quite intuitive as if we make rules very specific to the case, we would have a good precision but unlikely to cover other cases, hence a low recall.

6.2 Performance of Knowledge Bases

For each of the two tasks of recognizing positive and negative attributions of concepts/terms, we built a KB based on the first corpus and will test it on the second corpus. A sentence is deemed correctly suggested by the KB if the KB classifies the sentence to the right class and the concept/term of interest is also at least partly extracted. We do not require the full phrase containing the concept/term to be extracted to accommodate for the imperfection of our shallow parser as well as other pre-processing modules. For example:

*Randomized C 4.***5** *performed badly in this setting.*

Our KB correctly classifies this sentence into the *Disadvantage* class and extracts phrase **5** as the term of interest rather than **Randomized C 4.5**. We still consider this correct. Ultimately, users would inspect phrases extracted by the KB and can easily pick up the full phrase if it is not fully extracted.

Recognizing Positive Attributions: For this task, we build a knowledge base to determine if a sentence belongs to the *Advantages* class and also to extract the concept/term of interest. Given a sentence, the fired rule from the KB would give a *true* conclusion if the sentence is considered to be of *Advantages* class and *false* otherwise.

Using the first corpus we have built a KB consisting of 61 rules. Applying the knowledge base to the second corpus (4713 sentences), it classifies 178 sentences as belonging to the *Advantages* class. Checking the accuracy, 132 cases are correct. That gives a 74% (132/178) precision. It misses 18 cases giving a recall of 88% and an F-measure of 80.4%. Examples of sentences returned by the KB with the extracted concepts in bold face are:

> *Again,* **EM** *improves accuracy significantly.*
> *In this low dimensional problem it was more computationally efficient* **to consider a random candidate** *set.*

Recognizing Negative Attributions: This task is similar to the task of *recognizing positive attributions* except we would like to extract phrases that have been mentioned with a disadvantage or a negative sentiment.

Using the first corpus, we build a KB of 65 rules for this task. When applying to the second corpus (4713 sentences), it classifies 132 sentences as belonging to the *Disadvantages* class. Out

of them 114 sentences are correctly classified resulting in a precision of 86.4%. In the corpus, there are 25 cases that should be classified into the *Disadvantages* class but are missed by the KB. That gives a 82% recall and an F-measure of 84.1%.

Some examples of sentences in the *Disadvantages* class returned by the KB with extracted terms in bold face are :

> **The mechanism in ICET for handling conditional test costs** *has some limitations.*
> *The development of Foil was motivated by a failure we observed when* **applying existing ILP methods to a particular problem** *, that of learning the past tense of English verbs.*

Notice that the second example above is quite tricky to extract.

Query-directed Advantages and Disadvantages Extraction: To verify the quality of our approach, we look at an application scenario of finding advantages and disadvantages for a particular concept/term e.g. *decision tree*. The literal string *decision tree* appears at least 720 times in the corpus of 140 journals and papers. We only consider sentences that are classified into *Advantages* or *Disadvantages* class by the KBs and have the string *decision tree* in the extracted *target annotation*. Clearly, this simple analysis would miss cases where the actual string is not in the sentence but is mentioned via an anaphora instead. Future work will address this point.

The two KBs described above suggested 22 sentences for the *Advantages* class and 9 sentences for the *Disadvantages* class that have *decision tree* in the target annotation. These sentences were from 7 documents of 5 different authors. Some examples are:

> **Advantages:**
> *Results clearly indicate that* **decision trees** *can be used to improve the performance of CBL systems and do so without reliance on potentially expensive expert knowledge*
> **Disadvantages:**
> *Rule sets are relatively easy for people to understand, and rule learning systems outperform* **decision tree learners on many problems** .

Among the suggested sentences, 12 (out of 22) sentences of the *Advantages* class are correct and 7 (out of 9) sentences of the *Disadvantages* class are correct. That gives 55% and 78% accuracy for the *Advantages* and *Disadvantages* classes respectively. We will analyze why the accuracy is not very good for the *Advantages* class in the next section.

6.3 Analysis of KB errors:

Inspecting misclassified sentences of the above experiment for the *Advantages* class reveals that 6 of them are from the same document (accounting for 60% of the error) and are of the same type:

> *...what is the probability that [*RDR1_ *[*RDR1_noun2 *the smaller decision tree* RDR1_noun2*] is more accurate* RDR1_ *].*

which does not say that the decision tree is more accurate if we consider the sentential context. This misclassification is due to the fact that we have not seen this type of pattern during training which could easily be overcome by adding a single exception rule.

A number of misclassifications is due to errors in modules we used, such as the POS tagger or the Shallow Parser, that generate annotations and features used in the rule's condition. For example, in

> *Comparing classificational accuracy alone , assistant performed better than cn 2 in these particular domains.*

performed is tagged as VBN(past participle), rather than VBD(past tense), causing our Shallow Parser not to recognize *performed* as a proper VG. Consequently, our rule base did not classify this case correctly. We also noticed that several errors came from noise in the text we get from the pdf2text program.

Overall, errors appear to come from the inaccuracy of pre-processing modules or the fact that rules' patterns are either overly general or too specific. This is not crucial in our approach and in fact highlights a strength of KAFTIE - it allows experts to enter exception rules for the refinement of existing rules.

6.4 Exception Rules Structure

Apart from the default rule, every rule is an exception rule which is created to correct an error of another rule. An SCRDR KB could be viewed as consisting of layers of exception rules where every rule in layer n is an exception of a rule in layer $n-1$ [2]. The default rule is the only rule in layer 0. A conventional method that stores rules in a flat list is equivalent to layer 1 in our SCRDR KB. The KB for the *Disadvantages* class has 32, 33 and 1 rules in layers 1, 2 and 3 respectively. The KB for the *Advantages* class has 25, 31 and 4 rules in layers 1, 2 and 3 respectively. Having more than one level of exceptions indicates the necessity of the exception structure in storing rules. An example of 3 levels of exception rules has been shown in section 5.

7 Related Work

7.1 Ripple Down Rules (RDR)

Unlike domains where traditional RDR has been applied, the number of attributes is arbitrary large in natural language domains. It is therefore impossible to just compare the cornerstone case against the current misclassified case for a list of attributes that are different for the creation of new exception rules. In our framework, KAFTIE, the rule that misclassified the case, in addition to its cornerstone case, is presented to the experts to form a new exception rule. Furthermore, in order to reduce the number of duplicated or *similar* rules and to speed up the KB's convergence, existing rules could be modified as long as the KB remains consistent. For example, the sentence:

PBIL is very powerful

could be covered by a pattern:

(a) {Noun}{VG}{Token.string == powerful}

When we encounter a new sentence:

PBIL , in this experiment, is very powerful

we could add a new exception rule containing a pattern which covers the new case:

(b) {Noun}{Pair}{VG}{Token.string == powerful}

where an annotation Pair covers the phrase *" , in this experiment,"*. However, these two patterns are quite similar and could be combined into:

(c) {Noun}({Pair})?{VG}{Token.string == powerful}

Rather then entering a new rule containing the second pattern (b), experts are allowed to go back and modify the first pattern (a) , making it more general to cover the new case as well. The system would automatically check for consistency of the newly modified rule before committing the change. In future work, we plan to develop a recommendation mechanism where experts would be notified of rules that *nearly* match a case at hand so they could be made more general if needed.

Ripple Down Rules have also been applied to sentence and relation classification tasks [14, 8]. There are several shortcomings with these approaches. Firstly, patterns used in these works are simple utilizing only words, word groups and POS. Moreover, a pattern in an exception rule cannot refer to the exact matched phrases of the parent rule. That limits the capability of expressing exception rules. In this work we addressed these shortcomings by defining patterns over annotations. Annotations subsume all features used in previous work, as well as enable the reference to the matched phrase of the parent rules.

7.2 Knowledge Acquisition versus Machine Learning

Our approach takes a knowledge acquisition approach where experts create rules manually. This differs from machine learning approaches that automatically learn rule patterns [16, 9, 1], or learn implicit rules (not expressed as patterns) to classify sentiment expressions [13, 17] or for the task of information extraction [16]. We strongly believe that our approach is superior to automatic

[2] Rules in Figure 1 are presented in layers structure

approaches since an expert needs to spend their time to classify the sentences even where machine learning approaches are taken. This is also corroborated by some very recent studies comparing Machine Learning with Knowledge Acquisition using the same corpus of text and showing vast differences in accuracy in the range around 40% for Machine Learning depending on the used method and 80%+ for Knowledge Acquisition [15].

If the expert has to classify the sentences anyway, it does not take much more time to also provide some explanations on some of the sentences for why they should be classified differently to how the current knowledge base classified the sentence. I.e. the knowledge acquisition process comes almost for free compared to the effort of manual sentence classification required for both, our knowledge acquisition as well as for machine learning approaches.

For the two tasks presented in section 6, it took one expert about 10 hours to build one KB based on a corpus of 3672 sentences. However, most of the time was spent in looking at sentences to determine if they belong to the class of interest and which phrase in the sentence needs to be extracted. This amount of time is also needed for the tagging of training data for other techniques that attempt to learn rules automatically from examples. The actual time required to build a knowledge base consisting of 65 rules by one expert is only about 2 hours in total.

We believe that our approach leads to a significantly higher accuracy based on a given set of sentences than could be achieved by machine learning which would only exploit a boolean class label as expert information for each sentence [15]. As a consequence, the total number of sentences an expert needs to read and classify in order to achieve a certain classification accuracy can be expected to be much less with our approach than a machine learning approach. Similar findings for medical knowledge bases have been obtained in systematic studies for Ripple Down Rules [3]. Many other researchers [12, 10] share the same view by manually creating patterns and lexicons in their approaches. We take one step further by helping experts to incrementally acquire rules and lexicons as well as control their interactions in a systematic manner.

7.3 Information Extraction

Allowing a pattern to be a regular expression over annotations, the expressiveness of our rule language is at least as good as existing IE systems (e.g. [9, 16], see [11] for a survey). Our framework combines the following valuable features of those IE systems: it allows syntactic and semantic constraints on all components of the patterns including the fields to be extracted. Further, it can accommodate both single-slot and multi-slot rules. Finally, it can be applied to a wide variety of documents ranging from rigidly formatted text to free text.

7.4 Sentiment Analysis

Most of the related work on sentiment analysis has focused on news and review genres [12, 17, 10, 13]. We demonstrated KAFTIE's effectiveness on a corpus of technical papers as it appears more challenging but the framework is domain independent. A majority of existing approaches only analyze co-occurrences of simple phrases (unigrams or bigrams) within a short distance [10, 13] or indicative adjectives and adverbs [17]. In contrast to that, we look at more complex patterns in relation to the subject matter to determine its polarity. One reason for this difference might be that those applications tend to classify the polarity of the whole article assuming that all sentiment expressions in the article contribute towards the article's subject classification. In this work, we identify sentiment expressions to extract the subject of the expression as well as its sentiment classification. [12] has a similar goal as ours but they do not consider the surrounding context of the sentiment expression which could affect the subject's polarity.

8 Conclusion and Future Work

In this paper, we presented KAFTIE, an incremental knowledge acquisition framework that allows users to rapidly develop new text classification and information extraction systems. We

demonstrated the effectiveness of KAFTIE using two sample tasks of extracting information from scientific papers regarding advantages of concepts/terms as well as disadvantages.

Our experiments so far were very encouraging and showed that KAFTIE can be a very valuable support tool for the rapid development of advanced information extraction systems operating on complex texts as present in scientific papers.

The performance of the resulting information extraction system, after 10 hours of knowledge acquisition (expert time), have shown that the knowledge bases built using the framework achieved precisions of at least 74% and recalls up to 88% on an unseen corpus. It should be noted that all documents in the corpus are from different authors covering different topics. This suggests that it would be feasible to quickly build new knowledge bases for different tasks in new domains. Although it appears easy for experts to create rules, it would be desirable if the experts are presented with possible candidate rules to choose from. Even when the suggested rules are not correct to be used as-is, using them as a starting point to create the final rules should be helpful. Future work will address the problem of using machine learning techniques to automatically propose candidate rules to be used in a *mixed initiative* style [5]. Furthermore, we plan to investigate a method to suggest to the experts existing rules in the KB that *nearly* match the case at hand for possible modification of those rules.

References

1. F. Ciravegna. Adaptive information extraction from text by rule induction and generalization. In *17th International Joint Conference on Artifi cial Intelligence*, Seattle, 2001.
2. P. Compton and R. Jansen. A philosophical basis for knowledge acquisition. *Knowledge Acquisition*, 2:241–257, 1990.
3. P. Compton, P. Preston, and B. Kang. The use of simulated experts in evaluating knowledge acquisition. In *Proceedings of the Banff KA workshop on Knowledge Acquisition for Knowledge-Based Systems*. 1995.
4. H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 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, 2002.
5. D. Day, J. Aberdeen, L. Hirschman, R. Kozierok, P. Robinson, and M. Vilain. Mixed-initiative development of language processing systems. In *Fifth ACL Conference on Applied Natural Language Processing*, Washington, DC, 1997.
6. G. Edwards, P. Compton, R. Malor, A. Srinivasan, and L. Lazarus. PEIRS: a pathologist maintained expert system for the interpretation of chemical pathology reports. *Pathology*, 25:27–34, 1993.
7. C. Fellbaum, editor. *WordNet - An electronic lexical database*. MIT PRESS, Cambridge, MA, 1998.
8. A. Hoffmann and S. B. Pham. Towards topic-based summarization for interactive document viewing. In *Proceedings of the 2nd International Conference on Knowledge Capture(K-Cap)*, Florida, 2003.
9. J. Kim and D. Moldovan. Acquisition of linguistic patterns for knowledge-based information extraction. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):713–724, 1995.
10. S. Morinaga, K. Yamanishi, K. Tateishi, and T. Fukushima. Mining product reputations on the web. In *Proceedings of the Eighth ACM International Conference on Knowledge Discovery and Data Mining(KDD)*, pages 341–349, 2002.
11. I. Muslea. Extraction patterns for information extraction tasks: A survey. In *The AAAI Workshop on Machine Learning for Information Extraction*, 1999.
12. T. Nasukawa and J. Yi. Sentiment analysis: Capturing favorability using natural language processing. In *Proceedings of the 2nd International Conference on Knowledge Capture(K-Cap)*, Florida, 2003.
13. B. Pang and L. Lee. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing(EMNLP)*, pages 79–86, 2002.
14. S. B. Pham and A. Hoffmann. A new approach for scientific citation classification using cue phrases. In *Proceedings of Australian Joint Conference in Artifi cial Intelligence*, Perth, Australia, 2003.
15. S. B. Pham and A. Hoffmann. Extracting positive attributions from scientific papers. In *Submitted to Discovery Science*, Italy, 2004.
16. S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.
17. P. Turney. Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics(ACL)*, pages 417–424, 2002.