Extracting Positive Attributions from Scientific Papers

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 provide support for reading (or skimming) scientific papers. In this paper we report on the task to identify concepts or terms with positive attributions in scientific papers. This task is challenging as it requires the analysis of the relationship between a concept or term and its sentiment expression. Furthermore, the context of the expression needs to be inspected. We propose an incremental knowledge acquisition framework to tackle these challenges. With our framework we could rapidly (within 2 days of an expert's time) develop a prototype system to identify positive attributions in scientific papers. The resulting system achieves high precision (above 74%) and high recall rates (above 88%) in our initial experiments on corpora of scientific papers. It also drastically outperforms baseline machine learning algorithms trained on the same data.

1 Introduction

Knowing the advantages and disadvantages of a particular concept or algorithm is important for every researcher. It helps researchers in learning a new field or even experienced researchers in keeping up to date. Unfortunately, such information is usually scattered across many papers. Survey papers are generally written on an irregular basis, and hence up-to-date surveys may not be available. Furthermore, in new and emerging fields, often survey papers do not exist at all. Having a tool that could collect all the relevant information for a concept of interest would therefore be of tremendous value.

For example, suppose we want to check if a particular algorithm is suitable for our task at hand, such a tool could go through available papers and extract sentences together with the contexts that mention the advantages and disadvantages of the algorithm. This would make our task much simpler. We only have to look at those extracted sentences rather than going through a large number of entire papers.

Another useful scenario is the following: before reading a paper, we could quickly have a look at what the advantages and disadvantages of the things discussed in the paper are to make a decision whether the paper is relevant to our interest.

In this paper, we introduce a new framework for acquiring rules to classify text segments, such as sentences, as well as extracting information relevant to the classification from the text segments. We apply our framework to extract advantages of concepts or actions, i.e. positive attributions of the concepts/actions, in technical papers. An advantage is detected when a positive sentiment is expressed towards the concept or action. For example, given the following sentences:

E. Suzuki and S. Arikawa (Eds.): DS 2004, LNAI 3245, pp. 169–182, 2004.

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

We would like to detect that the algorithm *Randomizing* and the action *to use Knowledge Acquisition to solve the task* have been mentioned with a positive sentiment. Analysis of positive sentiments towards a concept is a challenging task that requires deep understanding of the textual context, drawing on common sense, domain knowledge and linguistic knowledge. A concept could be mentioned with a positive 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 its wider context it is not.

In this paper, we will first describe the underlying methodology of our framework in section 2- 4. Section 5 illustrates the process by giving examples on how the knowledge base evolves. In section 6, we present experimental results. Section 7 discusses related work and our conclusions are found in the last section.

2 Methodology

In this section we present the basic idea behind Ripple-Down Rules upon which our approach is based.

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.

This approach resulted in the expert system PEIRS used for interpreting chemical pathology results [4]. 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 a wrong conclusion. The new node is attached to the last node evaluated in the tree. 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. Importantly, new node is added only when its rule is consistent with the knowledge base i.e. all cases that have been correctly classified by existing rules will not be classified differently by the new rule.

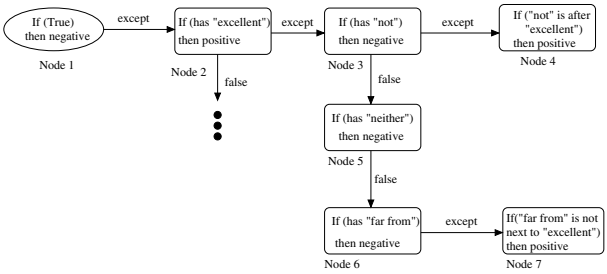

Fig. 1. An example SCRDR tree with simple rule language to classify a text into positive or negative class. Node 1 is the default node. A text that contains *excellent* is classified as *positive* by Node 2 as long as none of its exception rules fires, i.e., the text does neither contain *not, neither* 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 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 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. An annotation contains the character locations of the beginning and ending positions of the annotated text in the document along with the type of annotation and a list of feature value pairs.

The annotation pattern is a simplified 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.

The 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 as in this example:

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, such as here:

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

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

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

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 (Noun2), which must also contain a LexGoodAdj annotation for the annotation qualifier to be satisfied. When a phrase satisfies the above rule condition, a MATCH annotation would be posted over the whole phrase and Noun1, Noun2 annotations will be posted over the first and second NP in the pattern respectively. Note that Noun1 is not used in the condition part but it could be used later in the conclusion part or in the exception 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, being a purpose built lexicon containing terms indicating a positive sentiment:

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

This sentence triggers the posting of the following new annotations:

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

However, the following sentences 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 our task, 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. In this work, we extract the concept/term 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. In regards to whether a new exception rule needs to be added to the KB, a conclusion 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 our tasks.

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

As a result of the Shallow Parser module, 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 group) 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 bound 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 should 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 annotation qualifier.

Custom Annotations: In our current implementation we manually created a list of about 50 *good* adjectives and adverbs as a seed lexicon. We post LexGoodAdj annotations over words in that lexicon. In fact, users could form new named lexicons during the knowledge acquisition process. The system would then post a corresponding annotation over every word in such a lexicon. Doing this makes the effort of generalizing the rule quite easy and keeps the rules compact.

4 Implementation

We built our framework using GATE [2]. A set of reusable modules known as ANNIE is provided with GATE. These are able to perform basic language processing tasks such as part-of-speech (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 annotates text based on JAPE grammars. Our rule's annotation pattern is implemented as a JAPE grammar with some additional features.

We also developed additional processing resources for our task:

Lemmatizer: a processing resource that automatically puts a *lemma* feature into every Token annotation containing the lemma form of the token's string [11]. Lemmatizer uses information from WordNet [5] 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.

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

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:

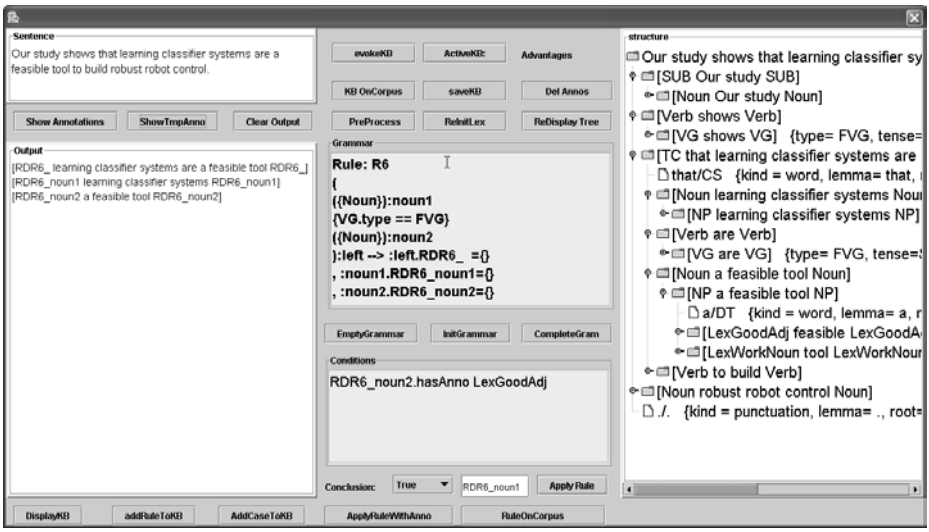

Fig. 2. The interface to enter a new rule where the rule is automatically checked for consistency with the existing KB before it gets committed. Annotations including those created by the shallow parser module are shown in the tree in the *structure* box.

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

A corpus was collected consisting of 140 machine learning conference and journal papers 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 quite diverse including most subtopics in machine learning.

We have applied our framework to build a knowledge base (KB) for recognizing sentences that contain advantages of a concept/term as well as extracting that concept/term. A sentence is considered to mention an advantage of a concept/term if the author expresses a positive sentiment towards that concept/term. Given a sentence, the fired rule from the KB would give the *true* conclusion if the sentence is considered to be of the *Advantages* class and *false* otherwise.

We randomly selected 16 documents from 16 different authors and grouped them into 2 corpora. The first corpus has 3672 sentences from 9 documents called the training corpus. The second corpus contains 4713 sentences from 7 documents called the test corpus.

Using the training corpus, we have built a KB consisting of 61 rules. Applying the knowledge base on the test corpus, it classified 178 sentences as belonging to *Advantages* class. Checking the accuracy, 132 cases are correct, resulting in a precision of 74% (132/178). A case is deemed correct, if in fact it mentions a concept/term with an advantage and that concept/term must be at least partly extracted. This flexibility allows for the imperfection of our shallow parser. The KB missed 18 cases resulting in a recall rate of 88% and an F-measure of 80.4%. Examples of sentences with the extracted concept in bold face are:

> *(3) Again,* **EM** *improves accuracy significantly.*
> *(4) In this low dimensional problem it was more computationally efficient* **to consider a random candidate** *set.*
> *(5)* **The topology of a KBANN-net** *is better suited to learning in a particular domain than a standard ANN.*

There are now 445 sentences in the corpus that contain at least one word from the seed lexicon. Our KB returned 178 cases out of which many do not contain any word from the seed lexicon. For example, sentence (3) above is returned because of the verb *improve*. This is to show that naively selecting sentences containing *good* adjectives or adverbs is not enough.

6.1 Comparison with Machine Learning Approaches

As a baseline with which to compare our approach, we used decision tree C4.5, Naive Bayes and Support Vector Machine algorithms using the bag-of-word representation on the task of classifying a sentence into the *true* or *false* class depending on whether the sentence mentions an advantage of a concept. Notice that this task is simpler than our task of extracting positive attributions as we do not ask the learner to extract the concept.

We prepared four different datasets which are variants of the training corpus (3672 sentences) described in the previous section:

- DataSet 1: It contains all full sentences from the training corpus.
- DataSet 2: It contains all sentences from the training corpus but sentences belonging to the *Advantages* class (i.e. containing an advantage of a concept/term) are duplicated 15 times. This is to give a balanced number of positive and negative examples in the training data.
- DataSet 3: Only the relevant phrase within a sentence is used in the bag-of-word representation for those sentences that are classified into the *Advantages* class. I.e. in building the KB, when an expert identifies a sentence as belonging to the *Advantages* class, s/he has to select a relevant phrase of the sentence and formulates a rule that would match the phrase. The phrase can be thought of as the most relevant part in the sentence which indicates that the sentence mentions an advantage. All other sentences are used as in the previous two datasets. This is done to give the learner more expert knowledge.
- DataSet 4: It contains all cases in DataSet 3 except that the *true* class cases are duplicated 15 times as done in Dataset 2.

Because there are 216 positive examples and 3456 negative examples in the original training corpus, Dataset 2 and 4 duplicate positive examples to give a balanced training data set.

We use J48, NaiveBayes and SMO in Weka [14] as implementations for C4.5, Naive Bayes and Support Vector Machine algorithms respectively. The 3 learners are trained on 4 datasets and are tested against the same test corpus described in the previous section. Their results are shown in table 1.

When the learners were trained on the data containing only the class label of the entire sentences, as in dataset 1 and 2, they performed very poorly giving the best F-measure of 27.6%. In dataset 3 and 4, more expert knowledge is provided in the form of relevant phrases of the sentences boosting the best F-measure to 40.5%, which is still significantly lower compared to the performance of our manually created KB of 80.4% based on the same data.

Table 1. Precision, **R**ecall and **F**-measure of the three algorithms trained on different training datasets but tested on the same test corpus.

	C4.5(J48)			Naive Bayes			SVM(SMO)		
	P	R	F	P	R	F	P	R	F
DataSet 1 *full sentence*	37	6.7	**11.3**	12.1	38	**18.3**	20	10	**13.3**
DataSet 2 *full sentence, balanced*	23.9	32.7	**27.6**	7.6	74	**13.8**	22.6	16	**18.8**
DataSet 3 *relevant phrase if available*	0	0	**0**	30.6	7.3	**11.8**	43.8	35.3	**39.1**
DataSet 4 *relevant phrase if available, balanced*	31.8	42.7	**36.5**	6.4	82.7	**11.8**	33.8	50.7	**40.5**

It is possible that the bag-of-word approach and the number of training examples given to those learners are not large enough to achieve a high precision and recall. It however supports our approach of utilizing more expert knowledge instead of requiring much more training data given that it takes only about 2 minutes for the expert to formulate and enter a rule into the KB. 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. In general, 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 approaches which would only exploit a boolean class label as expert information for each sentence. 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.

6.2 Can the KB's Performance Be Improved?

We fixed the errors made by the KB on the test corpus by adding exception rules to the KB. The resulting KB2 had 74 rules. We randomly selected 5 documents (containing 1961 sentences) from 5 different authors which are all different from the authors in the training and test corpus described in previous sections. When applied to this new test corpus, the KB2 achieved a precision of 79%(60/78) and recall of 90%(60/67). This is an indication that as we add more rules to the knowledge base, both precision and recall are improved.

6.3 Extracting Advantages

To verify the quality of our approach, we looked at an application scenario of finding advantages for a particular concept/term e.g. *decision tree*. The literal string *decision tree* appears at least 720 times in the corpus. We only considered those sentences, which were classified as *Advantages* by the KB and had the string *decision tree* in the extracted *target annotation* to be of interest. Clearly, this simple analysis would miss cases where the actual string is not in the sentence but is mentioned via an anaphora. Future work will address this point.

Our analysis results in 22 cases that have *decision tree* in the target annotation. These 22 cases were from 7 documents of 5 different authors. Some examples are:

> **Decision trees** *are a particularly useful tool in this context because they perform classification by a sequence of simpler, easy-to-understand tests whose semantics is intuitively clear to domain experts.*
> *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.*

Out of the suggested sentences, 12 are correct giving a precision of 55%.

6.4 Analysis of KB's Errors

Inspecting misclassified sentences of the above experiment 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 pdf2text program.

Overall, errors appear to come from the fact that rules tend to either be overly general or too specific and, thus, do not cover exactly what they should cover. While the particular choice of generality or specificity of an exception rule 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 entered, while a subsequent if-not rule patches a too specific rule.

6.5 Structure of Exception Rules

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 $n-1$ [1].

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

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. Our latest KB2 (from section 6.2) had 25, 43 and 5 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 is shown in section 5.

7 Related Work

Most of the related work on sentiment analysis has focused on news and review genres [9, 13, 7, 10]. Our work instead looks at technical papers as it appears more challenging. However, our framework is domain independent.

A majority of existing approaches only analyze co-occurrences of simple phrases (unigrams or bigrams) within a short distance [7, 10] or indicative adjectives and adverbs [13]. 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. [9] 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. The sentential context containing the sentiment expression is in fact analyzed in our work for sentiment classification.

Our approach takes a knowledge acquisition approach where experts create rules manually. This differs from machine learning approaches that automatically learn rule patterns [12, 6], or learn implicit rules (not expressed as patterns) to classify sentiment expressions [10, 13]. We strongly believe that our approach is superior to automatic approaches since experts need to spend their time to classify the sentences even where machine learning approaches are taken. This has been proved true on the task of classifying sentences into *Advantages* category described in Section 6.1 where our manually build KB outperformed several machine learning algorithms trained on the same data. Similar findings for medical knowledge bases have been obtained in systematic studies for Ripple Down Rules [1].

Many researchers [9, 7] 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 controlling their interactions in a systematic manner.

Our work could also be considered as a kind of information extraction. Allowing the 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. [6, 12], see [8] 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 extracted fields.
- it can accommodate both single-slot and multi-slot rules.
- it can be applied to a wide variety of documents ranging from rigidly formatted text to free text.

8 Conclusion and Future Work

In this paper, we have presented a domain independent framework to easily build a knowledge base for text classification as well as information extraction. We applied the framework to the task of identifying the mention of advantages of a concept/term in a scientific paper. The objective being to support scientists in digesting the ever increasing literature available on a subject.

Initial experiments on this task have shown that the knowledge base built using our framework achieved precision of at least 74% and recall rates of more than 88% on unseen corpora.

We compared the performance of our knowledge base on a simpler task, i.e. sentence classification only (without extracting any information), with that of classifiers constructed using three different machine learning techniques, often used for text mining. Our knowledge base proved to be far superior to those machine learning approaches, based on the same text. Furthermore, it should be noted that for both approaches, our knowledge acquisition approach as well as supervised machine learning approaches, a human is required who classifies the individual sentences. The additional time our human expert required to enter rules into the knowledge base was only a fraction of the time it took to classify the sentences in the first place. The reason being that many more sentences need to be classified than the number of rules that was entered. This suggests that our approach uses the expert's time much more economically than supervised machine learning approaches which only utilize the class label as the indication of what to do with a sentence.

It should be noted that all documents in those corpora were from different authors covering different topics. Furthermore, building the KB of the above quality was reasonably quick. This suggests that by using our framework, it is feasible to quickly, i.e. within a few days, 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. We are working towards using machine learning techniques to automatically propose candidate rules to be used in a *mixed initiative* style [3].

References

1. 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.
2. 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.
3. 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.

4. 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.
5. C. Fellbaum, editor. *WordNet - An electronic lexical database*. MIT PRESS, Cambridge, MA, 1998.
6. 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.
7. 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.
8. I. Muslea. Extraction patterns for information extraction tasks: A survey. In *The AAAI Workshop on Machine Learning for Information Extraction*, 1999.
9. 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.
10. 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.
11. M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
12. S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.
13. 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.
14. I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, 2000.