RDRCE: Combining Machine Learning and Knowledge Acquisition

Han Xu and Achim Hoffmann

School of Computer Science and Engineering
University of New South Wales, Sydney, Australia

Abstract. We present a new interactive workbench RDRCE (RDR Case Explorer) to facilitate the combination of Machine Learning and manual Knowledge Acquisition for Natural Language Processing problems. We show how to use Brill's well regarded transformational learning approach and convert its results into an RDR tree. RDRCE then strongly guides the systematic inspection of the generated RDR tree in order to further refine and improve it by manually adding more rules. Furthermore, RDRCE also helps in quickly recognising potential noise in the training data and allows to deal with noise effectively. Finally, we present a first study using RDRCE to build a high-quality Part-of-Speech tagger for English. After some 60 hours of manual knowledge acquisition, we already exceed slightly the state-of-the art performance on unseen benchmark test data and the fruits of some 15 years of further research in learning methods for Part-of-Speech taggers.

Keywords: Knowledge Acquisition, Ripple Down Rules, Machine Learning, TBL, Part-of-Speech tagger.

1 Introduction

For long there has been a division of opinions regarding the best way of building a knowledge base. Manual approaches to build knowledge bases tend to be tedious while automatic approaches require usually large amounts of training data and the results may still be inferior to those achieved by manually building a knowledge base.

In this paper we present our new workbench Ripple Down Rules Case Explorer (RDRCE) which has been designed to facilitate both, the automatic construction of an initial knowledge base which is subsequently further improved in a manual fashion. This seems to be of particular value for applications in Natural Language Processing (NLP). Reason being that often some training data has been developed for machine learning, but the results of the applied learners are not really satisfactory as the NLP domains are too complex.

Ripple Down Rules [4] have proven to be a very effective way of manually building knowledge bases. The approach has been widely applied and adapted, including to various NLP applications, e.g. [8,11]. RDRCE not only allows to

B.-H. Kang and D. Richards (Eds.): PKAW 2010, LNAI 6232, pp. 165–179, 2010.
© Springer-Verlag Berlin Heidelberg 2010

automatically generate an initial RDR tree using Transformation-Based Learning(TBL), which is a well-entrenched learning technique for certain NLP problems. It also interactively supports the inspection and refinement of the initial RDR tree to quickly improve upon the performance achieved by the machine learner. A common issue turned out to be that the training data may contain considerable degrees of noise. We designed RDRCE to support a user to quickly identify noisy training data and to correct it on the spot or mark it as dubious and to be set aside for later examination.

Finally, we applied RDRCE to the problem of Part-of-Speech tagging, which has attracted considerable interest by NLP researchers in the past. Since large scale training data has been available (i.e. more than 1,000,000 words with manually generated Part-of-Speech tags) virtually all approaches to build automatic Part-of-Speech taggers have been some sort of learning or statistical approach.

In our first case study we managed already to exceed slightly the best performance of any known Part-of-Speech tagger that has been used on the given training and test corpus with a total of about 60 hours of knowledge acquisition sessions. This is despite the fact that our training algorithm produces inferior results compared to the state-of-the-art. Furthermore, while developing that tagger, we also discovered that the widely used benchmark training and test data contains a considerable degree of noise.

The paper is organised as follows: The next section presents the background and related research. In section 3 we present the conceptual design of RDRCE. Section 4 contains a detailed description of RDRCE and its rule language. In section 5 we discuss our case study. Section 6 presents the conclusions and future work.

2 Background

2.1 Ripple Down Rules

Ripple Down Rules (RDR) [4] form the basis of our approach. RDR allows one to add rules to a knowledge base (KB) incrementally without the need of a knowledge engineer. A new rule is only created when the KB performs unsatisfactorily on a given case. The rule represents an explanation for why the conclusion should be different from the KB's conclusion on the case at hand. RDR was first used to build the expert system PEIRS for interpreting chemical pathology results [5].

A Single Classification Ripple Down Rules (SCRDR) tree is a binary tree where each node has between zero and two child nodes. The edges or links connecting the child nodes are typically called *except* and *if-not* (or *false*) edges. See Fig. 1. Associated with each node in a tree is a *rule*. A rule has the form: *if* COND *then* CONCLUSION where COND is called the *condition* and CONCLUSION the *conclusion*, which is most often simply a class label assigned to a case, but can be any kind of action. A *case* is an object to which the RDR knowledge base is applied, such as an attribute-value vector that should be classified. To decide on the conclusion to be applied to a case given an SCRDR tree we start

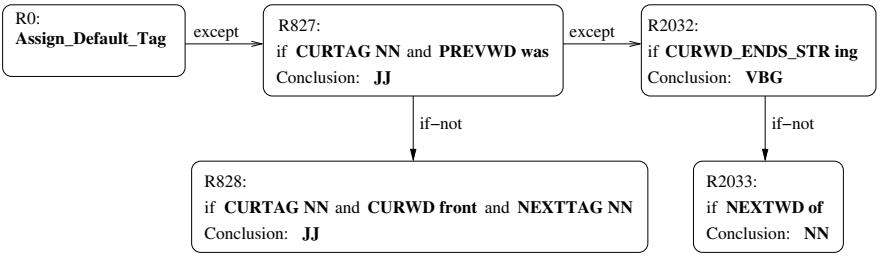

Fig. 1. Part of a Ripple Down Rules tree for Part-of-Speech tagging

with the root node. At each node that is encountered, the corresponding rule condition is evaluated for the case. If the case satisfies the condition, we say that the node *fires*. If a node fires, we follow the exception link of that node, if there is any. Otherwise, we follow the if-not link, if there is any.

The final conclusion performed by the SCRDR tree is the conclusion of the node that fired last, i.e. that is deepest in the tree (but is often not a leaf node). To ensure that a conclusion is always performed, 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 in the evaluation path of the given case. If that node fired, the new node is attached via an *except* link and otherwise via an *if-not* link.

Besides SCRDR a number of other styles of Ripple Down Rules have been developed which are useful for specific application domains, such as Multiple Classification RDR (MCRDR) [7], or [10] to just name a few. An excellent recent survey paper on the RDR methodology is [14]. In the following we will limit our consideration to SCRDR trees, even so we would just speak of RDR trees for brevity.

2.2 Part-of-Speech Tagging

Part-of-Speech (POS) Tagging refers to the process that assigns each word in a natural language sentence with a suitable tag from a given set of tags, such as Verb, Noun, etc. It has its significance in various areas of Natural Language Processing (NLP), such as Parsing, Machine Translation, Text to Speech and Information Extraction. It is usually used as a low-level pre-processor for other high-level NLP tasks, thus its accuracy has a direct influence on the final performance of the entire system. Various automatic POS tagging techniques and applications have been developed over the past few decades. Early approaches were hand-crafted rules, e.g. [9]. The Brill tagger, e.g. [1], using a machine learning approach resulted in a higher accuracy than those earlier attempts. Since then a considerable number of more sophisticated statistical and machine learning approaches have been developed to improve on the performance of the Brill tagger. Those developments elaborate on various statistical techniques, such as

Hidden Markov Models, e.g. [3], or Maximum Entropy, e.g. [13]. The most recent advancements in statistical POS tagging using a very elaborate technique involving a Perceptron [18] have pushed the state-of-the-art tagging accuracy on the Penn Treebank corpus Wall Street Journal to 97.44%.

3 Our Approach

Our aim is to build high quality knowledge bases for domains where there is a considerable amount of classified training data available. We developed our RDR case Explorer (RDRCE) which allows to automatically generate an initial RDR tree which can subsequently be further refined and improved by supporting the browsing of existing training data as well as it allows to look at new data as it becomes available. However, since training data is often prone to noise, RDRCE has also been designed to allow the efficient checking of suspicious training data and allows to record data that has been identified as noisy or that warrants more thorough checking at a later stage.

If a large volume of training data is available it appears sensible to take advantage of that and build automatically an initial RDR tree and to improve and refine it subsequently by interaction with a domain expert. Building a sophisticated knowledge base with our approach can be divided into three phases:

1. Generating the initial RDR tree. If classified training data is available, RDRCE uses Transformation-Based Learning (TBL) [1] to generate a rule list which can be automatically converted into an RDR tree.

   If no training data is available or if one opts to build the RDR tree from scratch, the RDR tree would only consist of the root node and a default conclusion.
2. Continued refinement of the initial RDR tree: RDRCE allows the user to gain quickly an in-depth understanding of the domain and the short-comings of the rules generated so far. Furthermore, as some of the training data may also be incorrectly classified, RDRCE also allows the user to correct the classification of training cases or to mark them as dubious for detailed checks at a later stage.

   A *case* in the training or testing corpus is a token (usually a word) along with its provided tag. Each case is provided with its context, i.e. words and their respective tags on both sides of the tag in question. We call the *cornerstone cases of a rule* r all cases in the training corpus for which r fires and assigns the correct tag, i.e. the same tag as provided in the training corpus. We call *bad cases* of a rule r all cases in the training corpus for which r fires but assigns an incorrect tag, i.e. different to the tag in the training corpus.
3. Adding new RDR rules: RDRCE provides a more versatile rule language than the rule templates used in Brill's tagger. It also allows to build dictionaries which hold an extensible list of words or tags. Such dictionaries can be used in rule conditions where a given word or tag in a case is tested for being a member in a given dictionary.

Quite a few learning algorithms have been developed which either directly build RDR trees from classified data, e.g. INDUCT [6], or which build similar structures, such as decision trees that can be transformed into an RDR tree. Investigations into the combination of ML and RDR had also been carried out by researchers in the past. Wada et al. [21] demonstrated a flexible knowledge-based system, where the KB is constructed by a human expert using RDR at an earlier stage of the development when there is no large amount of data available. Subsequently, when sufficient data is available, the KB is refined by using a Minimum Description Length Principle based induction technique. Suryanto and Compton [19] presented a method to minimise the KA work load for the human expert by enabling the RDR system to generalise the knowledge acquired from the expert and to automatically discover intermediate conclusions. Other studies of using RDR exception structures for Machine Learning include Catlett [2] and Scheffer [16].

For our study, we chose to build on Brill's TBL approach [1] which proved to perform exceptionally well in the domain of Part-of-Speech tagging. Brill's TBL approach has also been successfully applied to solve many other problems, including Noun Phrase Chunking [12] and Dialog Act Tagging [15]. Furthermore, it appears to provide an excellent basis for an initial RDR tree which can then be further improved in a manual fashion.

In the following we describe Brill's transformational learning approach and show how the learning result can be converted to an RDR tree.

3.1 Turning the Brill Tagger into an Initial RDR Tree

As mentioned in section 2.2, Brill developed his Transofrmation-Based Learning (TBL) [1] approach for automatically acquiring rules for a rule-based tagger from a manually tagged training corpus.

Initially, known words are tagged with their most-frequent tag and unknown words are tagged either as 'noun' or 'proper noun', depending on whether the word is capitalized. Tagging errors are identified by comparing the current tagging with the training corpus and candidate error-correction rules are considered. The candidate rules are constructed by exhaustively instantiating a set of rule templates, which make reference to a token's lexical features (morphological features such as a token's prefix or suffix) and contextual features (such as the token's preceding/following token or POS tag). All rules are of the form FROM_TAG TO_TAG if COND. For lexical rules, COND may include predicates like 'the word has suffix XY'. For contextual rules, COND may include predicates like the surrounding tags are X and Y. Only the rule candidate with the highest net improvement is added to the rule list. This newly learned rule is subsequently applied to the whole training corpus and effects changes where the rule's condition is satisfied. The same process to add the next rule is repeated until there is no rule whose net improvement reaches a user-specified threshold.

The outcome of the rule learning process is an ordered list of transformation rules. We converted the so generated rule list automatically into an RDR tree as follows:

3.2 Converting a TBL Rule List Into an RDR Tree

Following is the algorithm RLRDR (Rule List to Ripple Down Rules) which converts a TBL rule list to an RDR tree:

```
rdrnode RLRDR(corpus C, rule_list R) {
 LOOP:
    if (length(R) == 0) return NULL;
    else {
        Rule = pop(R); // remove the lowest indexed rule from R
        R_e = R - Rule;
        R_f = R - Rule; //R_e = R_f = rest of the rule list
        for (case in C) {
            if (Rule is evaluated to True) {
                apply_rule(Rule, case);
                C_e.append(case);
            } else C_f.append(case);
        }
        if (length(C_e) != 0) { //if the Rule applied at least once to corpus C
            N = new rdrnode; N→question = Rule;
            N→cornerstonecases=set_correctly_classified_cases;
            N→except = RLRDR(C_e, R_e); N→false = RLRDR(C_f, R_f);
            return N;
        } else goto LOOP;
    }
}
```

4 Ripple Down Rules Case Explorer (RDRCE)

4.1 RDRCE's Rule Language

Our rule language builds upon the rule templates used in the Brill Tagger [1], but goes significantly beyond that. For a full list of the rule templates used in the Brill tagger, see [1].

We follow the notation of the rule templates by Brill. As a consequence, each rule consists of two parts – the action part and the condition part. The action part itself consists of two tags, the first one being the current tag of a token and the second tag being the new tag assigned to the token, if the rule condition is satisfied. I.e. each rule is of the following form:

FROM_TAG TO_TAG CONDITION

The rule condition can be a single elementary condition or a combination of such using *and*, *or*, and *negation*. In our rule language, we added the following new elementary condition types to those already available in Brill's rule templates. Our extensions can be divided into the following groups:

1. Dictionary membership checks: checks if a token or tag at a specific relative position to the focus token is in a predefined group (dictionary). The following condition names are available with their intuitive meaning. Each condition name needs to be followed by the name of the respective dictionary. E.g. CURWD_IN_DIC BE_DIC would require the current token to be some form of *be*. The other conditions available are the following: PREV1OR2OR3WD_IN_DIC, PREV1OR2WD_IN_DIC, PREV2WD_IN_DIC, PREVWD_IN_DIC, CURWD_IN_DIC, NEXTWD_IN_DIC, NEXT2WD_IN_DIC, NEXT1OR2WD_IN_DIC, and NEXT1OR2OR3WD_IN_DIC. Furthermore, all conditions above can also check whether the assigned tag to the respective token is in the specified dictionary. The condition names are PREVTAG_IN_DIC etc. During the knowledge acquisition process using RDRCE, we developed dictionaries that contained the following word or tag groups: all forms of *be*, *have*, or *do* respectively. All variants of adjective, noun, adverb, and verb tags. Others include time markers and causative words, such as *help*, *let*, etc.

   The purpose of introducing those dictionaries is to raise the levels of abstraction in a rule's condition to make the rules expressive enough to generalize well to unseen cases while keeping the rules concise and easily interpretable. All of those dictionaries are constructed incrementally along the way when doing the knowledge acquisition. Our workbench allows the user to directly interact with those dictionaries at run time. If the user encounters a new word it can immediately be added to a suitable dictionary.
2. Additional elementary conditions:
   - NEARBYTAG (NEARBYWD) – checks if a user specified tag (word) is on either side of the focus token
   - SURROUNDWD – checks if the focus token is in the middle of the two user specified words
   - PREVBIGRAMWD (NEXTBIGRAMWD) – checks if the preceding (following) bigram is the user specified bigram
3. Elementary lexical conditions: they check the following lexical properties of a token:
   - ISTITLE – checks if the focus word starts with a capital letter. ISLOWER (ISUPPER) – checks if the focus word is without any (with only) upper case letter. CURWD_IS_DIGIT – checks if it is a digit
   - HAS_STR *str* – checks if the focus word contains *str* as a substring. CURWD_ENDS_STR *str* (PREVWD_ENDS_STR *str*) – checks if the focus word (previous word) ends with *str*.

4.2 Inspecting and Refining RDR Trees with RDRCE

RDRCE provides various useful statistics, such as reports on the type and size of error clusters. By *error cluster* T_1-T_2, we mean all cases that are tagged T_1 by the knowledge base but the training data tag is T_2. RDRCE also provides the current tagging accuracy for a given tag as well as for a prospective rule which the user might consider to commit to the knowledge base. Error distributions are automatically calculated and provided to the user. The user can freely navigate

through the existing RDR tree by choosing a specific node and traversing the tree or by choosing a specific error cluster.

To understand why certain errors are being made by the knowledge base the user can view the cornerstone cases and the bad cases associated with a given rule. Since the number of cases can be quite large (several hundred cases is quite normal) various sorting and filtering functions (such as filtering by nearby contexts) can be used for displaying purposes. This allows an easier comparison between cornerstone cases and bad cases in order to develop valid refinement rules. Once a new rule is formulated the user can test its performance immediately on the existing corpus to assist in deciding to commit the rule to the knowledge base. (This is different from the traditional RDR approach, where cases are presented to the user one at a time. A rule is added to fix this single case, without having a good impression on how well the rule might generalize to other forthcoming cases.) Fig. 2 shows a screenshot of RDRCE displaying the performance of a candidate rule.

Fig. 2. This screenshot shows how the performance of the candidate rule EX RB NEXT1OR2OR3WD_IN_DIC BE_VB_DIC on the training corpus can be evaluated before the rule is actually committed

5 Case Study

5.1 Building the Initial RDR Tree

We used the Penn Treebank 3 (PTB) corpus and we use the same data division of the PTB parsed section as all of [3,20,17,18] do in order to allow better comparisons. We trained the Brill tagger V1.14 to obtain our initial rule lists. This resulted in a learned lexical rule list of length 200 and a contextual rule

list of size 932. We converted the 932 contextual rules into our initial RDR tree using our automatic conversion algorithm.

This resulted in 1,318 nodes for the initial RDR tree (some TBL rules resulted in multiple nodes in the RDR tree). The tagging process then used first the lexical rules and subsequently the contextual RDR rule tree. This resulted in an accuracy of 97.31% on the training corpus. An accuracy of 97.08% was achieved on the previously unseen sections 22-24 of test data. In this study we limited ourselves to acquiring contextual rules, as they appeared to hold more potential for improvement.

5.2 Inspecting the Initial RDR Tree

There are some nodes in the initial RDR tree that have a negative effect. This is as a consequence of the TBL calculating scores for candidate rules on the entire training corpus. In contrast, a TBL rule may have multiple copies in the initial RDR tree each applicable only to a local context, i.e. a small subset of the entire training corpus. So, the performance of the various copies of a rule can vary considerably. RDRCE allows to specifically analyse and fix those rules that have a negative score in the local RDR tree context. This might quite possibly result in cancelling the rule in its entirety. Here is such an example:

In node #1011, the rule: RB JJ LBIGRAM how much yields a negative score on bad cases like the following:

1) But/CC how/WRB **much/JJ** will/MD shoppers/NNS benefit/VB
2) The/DT question/NN remains/VBZ :/: how/WRB **much/JJ** can/MD the/DT West/JJ

There is no cornerstone case for this particular node, i.e. no case where the rule produces a correct result. The rule was generated in the TBL learning process because in other contexts there were cases for which the rule produces a correct result. RDRCE allows to retrieve those supporting cases from other nodes in the initial RDR tree to provide a better understanding in what circumstances the rule makes sense. For the above rule the cornerstone cases for the nodes #378, #660, #1201 and #1217 are collected and displayed by RDRCE. Below are some of those cornerstone cases given as examples:

decided/VBN when/WRB or/CC by/IN how/WRB **much/JJ** ./.
did/VBD n't/RB say/VB by/IN how/WRB **much/JJ** ./.
,/, showing/VBG traffic/NN wardens/NNS how/WRB **much/JJ** time/NN the/DT motorist/NN

In all of the bad cases, there is a modal (MD) following the phrase "how much". It is most likely, that a verb is somewhere in the right contexts which the "much" is modifying, thus the "much" should be tagged as a RB rather than JJ. This is clearly visible for the first bad case 1) above: the "much" is modifying the verb "benefit", but the window is not wide enough for the second bad case. The user can request a full sentence inspection which shows the following for 2):

> "/" The/DT question/NN remains/VBZ :/: how/WRB **much/JJ** can/MD the/DT West/JJ German/JJ market/NN absorb/VB ?/. "/" says/VBZ one/CD senior/JJ dealer/NN ./.

Now it becomes clear that the "much" in this case is modifying the verb "absorb" 6 positions to the right of the focus token "much". Thus, a following MD is a good cue that "much" in the preceding phrase "how much" should be tagged as adverb (RB). This leads to the addition of the following exception rule: JJ RB NEXTTAG MD. I.e. change the tag from adjective (JJ) to adverb (RB), if the immediately following tag is a modal (MD).

Other rules were only learned by the TBL due to incorrect manual tagging of the training corpus. Below are some examples:

Node #286 with the rule: VBZ NNS PREVTAG IN had a negative score of -22 given its local context in the generated RDR tree. The rule says: change the tag from verb 3rd person singular present (VBZ) to noun plural (NNS) if the previous tag is preposition (IN). All the bad cases of this node are about the token *'s* as shown in the following examples:

> allowed/VBN in/IN there/RB ,/, that/IN **'s/NNS** all/DT I/PRP know/VBP But/CC he/PRP says/VBZ that/IN **'s/NNS** no/DT more/JJR a/DT requirements/NNS ,/, and/CC that/IN **'s/NNS** a/DT source/NN of/IN

Clearly, *'s* should never have the possible tag of NNS (noun plural). Inspecting the training corpus shows that the token *'s* had been erroneously tagged as NNS while it should be VBZ, as e.g. in:

> "/" I/PRP 've/VBP never/RB seen/VBN so/IN many/JJ New/NNP York/NNP City/NNP G.O./NNP **'s/NNS** up/IN for/IN sale/NN ,/, "/" said/VBD another/DT trader/NN ./.

So we added the following wild-card rule to effectively cancel the rule and to revert all of the bad cases within this node: NNS VBZ ANY ANY.

Node #286 is an exception rule to node #284. The rule in #284 is: POS VBZ NEXTTAG DT, which allows only cases with ending *'s*, due to the fact that *'s* is the only token that can be tagged as both POS (possessive ending) and VBZ. Based on that the expert can have a high degree of confidence that the cancellation of the rule in node #286 is correct.

Another interesting situation was found in node #1161, which covers 14 cases while yielding 0 improvement. The node's rule is VBD VBN WDNEXTTAG received IN. I.e. change tag from verb past tense (VBD) to verb past participle (VBN), if the focus word is *received* followed by a preposition (IN). E.g.

> The/DT funds/NNS **received/VBN** from/IN pharmaceutical/JJ firms/NNS the/DT value/NN of/IN the/DT stock/NN **received/VBN** in/IN a/DT conversion/NN

Bad cases within this node include the following examples:

> dating/VBG back/RB to/TO forms/NNS it/PRP **received/VBN** in/IN 1985/CD ./.
> dividend/NN similar/JJ to/TO one/CD he/PRP **received/VBN** before/IN selling/VBG his/PRP$

It is fairly easy to notice the difference here: for the cornerstone cases, the focus word is following an NN or NNS, which results in the phrase structure: "*something* received IN", where it is very likely that "received" here is indeed used in a passive voice. However, for the bad cases, the tag preceding "received" is a personal pronoun which resulted the following structure: "*someone* received IN", where it is more likely that the focus word is immediately following its subject, and thus, is used in active voice and should be tagged as VBD rather than VBN. Based on this observation, the following rule was added as an exception rule to this node: VBN VBD PREVTAG PRP. Fig. 3 shows a screenshot when the above rule is tested.

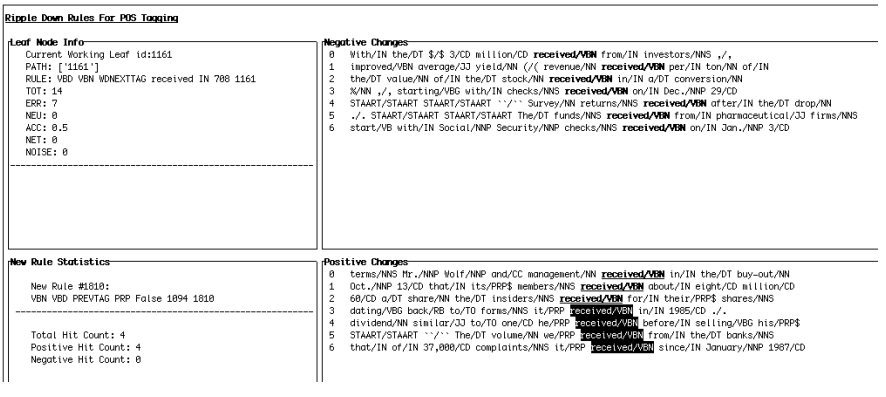

Fig. 3. Screenshot of RDRCE examining the performance of the new rule VBN VBD PREVTAG PRP

5.3 Adding Rules for the Residue

The set of training cases which are not retagged by the TBL is often called the *residue*. Each of the 912,344 training tokens were initially tagged by the start-state-tagger (part of Brill tagger, which simply assigns the most frequently occurred tag to a given token). Only 39,139 (4.29%) tokens were retagged by the rules produced by the TBL, or equivalently, by our initial RDR tree. The residue of 95.71% does not need any retagging in most cases, i.e. 94.48% are actually correctly tagged. This leaves 1.33% (or 11,237 cases) of the training corpus in need of retagging, which actually makes up some 46% of all erroneous tags. All of the *residue* is in the last node along the *if-not* link chain of the initial RDR tree.

In the residue, RDRCE found 251 error clusters. 144 out of those have a size ≤ 10. For our study, we were mainly interested in the larger error clusters with size ≥ 50, where a potentially higher performance improvement can be achieved. Adding rules to cope with tagging errors in the residue is as described in the previous section, except that the navigation is based on error cluster names rather than node numbers. Below is an example:

In the error cluster: "EX-RB" (tagged as "existential there" (EX) but should be adverb (RB)), the following rule was added: EX RB NEXT1OR2OR3WD_IN_DIC BE_VB_DIC based on a very simple intuition that the *existential there* should usually be followed closely in its right context by some form of *be*. If there is no form of *be* following, the correct tag of "there" will more likely be adverb (RB). This single rule fixed 34 (54%) cases within this error cluster. See also Fig. 2.

However, not all errors in the residue are this easily addressable. Our experience showed that the larger the error cluster the harder it is to formulate rules. This is because it is more difficult to tailor rules to accommodate a large set of cases, even if one allows for a reasonable number of exception rules. However, developing rules for the residue clusters was not only harder because of the size of the clusters but also because of the fact that the TBL did not manage to come up with applicable rules, thus indicating that the individual cases themselves are quite difficult to discern.

5.4 Discovering Noise in the Training Data

RDRCE made it relatively easy to detect noise in the training data and also allowed the user to correct the data by the press of a single button for each case. Much of the noise was also found in the residue. It seems likely that due the noise the cases ended up in the residue, as the TBL was incapable of finding effective rules.

For example, the PTB tagging guidelines on page 12 state: "Hyphenated modifiers, should always be tagged as adjectives (JJ)". However, in the error cluster NN-JJ we find e.g. that "junk-bond" in the phrase "junk-bond market" was tagged as NN 15 times and as JJ 4 times although the tagging guidelines say clearly that it should be tagged as JJ in all cases. Further, when we tried out the following rule within error cluster NN-JJ: NN JJ HASSTR '-' and NEXTTAG NN or NNS, which is expressing the above guideline in our rule language, the new rule gave us 229 positive changes but 1255 negative changes. As a next step using RDRCE we can walk through the 1255 negative cases and can verify whether all of them are noise. While the number of cases is large, it appears to take only a fraction of a second for most cases to verify that they are noise indeed.

5.5 Results

While we discovered a considerable amount of noise in the training data we did not correct it in order to allow a sensible comparison of the performance of our resulting RDR tree with the best of other POS taggers. Where available, we report the accuracies on both, the training data (WSJ section 0-18) as well as

the unseen test data (WSJ section 22-24). While the sections 19-21 were used by others for calibrating their learning approaches, we did not need that and did not touch those sections for comparison purposes. The respective results are shown in Table 1. We spent about 60 hours of knowledge acquisition to refine the initial RDR tree. During the knowledge acquisition process we added 415 rules to the initial RDR tree. This resulted in a performance exceeding the best known POS tagger to date [18], which as automatically learned, though. However, it took some 15 extra years of research to develop the algorithm used in [18] compared to Brill's TBL learner which we used as a basis for our POS tagger development. We think that the considerable effort that went into developing the today's best learning techniques, e.g. in [18], would well exceed the manual effort required to refine an initial RDR tree using RDRCE. Of course, we spent also quite an effort for building RDRCE. However, we believe that RDRCE can be easily applied to other tasks in the Natural language Processing domain, such as citation classification, sentence classification, or information extraction whenever there is a training corpus available.

Table 1. Part-of-Speech tagging results for the PTB WSJ corpus. The knowledge base built using RDRCE within some 60 hours exceeds, albeit slightly, the performance of the best automatically learned taggers to date.

Tagger	Accuracy Training WSJ00-18	Accuracy Test WSJ22-24
Brill's Tagger (turned into inital RDR tree)	97.31%	97.08%
Ratnaparkhi 1996	unknown	96.63%
Collins 2002	unknown	97.11%
Toutanova et al. 2003	unknown	97.24%
Shen et al. 2007	unknown	97.33%
Spoustova et al. 2009	unknown	97.44%
RDRCE	97.76%	**97.46%**

6 Conclusions and Future Work

We presented our new RDRCE which was designed to support the hybrid process of building an RDR knowledge base partly manually and partly automatically using available training data.

Using RDRCE we showed that it is feasible to develop a Part-of-Speech tagger that exceeds, albeit slightly, the current state-of-the-art on standard test data within a week and a half or so of knowledge acquisition sessions. The manual effort spent appears to compare quite favourably to the effort that was spent otherwise on improving the state-of-the-art of learning algorithms for part-of-speech tagging. Overall, it appears that the best possible performance for any Part-of-Speech tagger will lie well below 100%. This is partly due to noise in the underlying corpus, as we discovered, but also partly due to the fact that even humans will disagree on some tags.

We employed Brill's TBL approach to generate a transformation rule list. We introduced a new conversion algorithm in section 3.2 that turns the transformation rule list into an RDR tree and assigns the training data to the relevant nodes in the RDR tree. We believe that the TBL approach is applicable not only to the problem of Part-of-Speech tagging but also to a host other NLP problems, including citation or sentence classification, or information extraction. Our initial comparison of the performance of the TBL with alternative learning algorithms whose results is either an RDR tree or can be converted into one, namely INDUCT and C4.5, favoured strongly the TBL approach. For the full training data set, the WEKA implementations of the named algorithms did not actually manage to complete the learning as they ran out of memory. It is also not clear how to take the flexibility of the TBL into account, as it allows to adapt it easily to a new problem domain by modifying the templates it uses. This kind of domain knowledge appears to be more difficult to provide to the other learners in a feasible way (as they tend to run into computational problems with too many features).

However, future work should look more closely into the comparative performance of different suitable learning algorithms. Furthermore, we intend to explore to what degree the refinement of an initial RDR tree can be further automated.

References

1. Brill, E.: Some advances in transformation-based part of speech tagging. In: AAAI 1994: Proceedings of the Twelfth National Conference on Artificial Intelligence, vol. 1, pp. 722–727 (1994)
2. Catlett, J.: Ripple-down rules as a mediating representation in interactive induction. In: Proceedings of the Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, Kobe, Japan, pp. 155–170 (1992)
3. Collins, M.: Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In: EMNLP 2002: Proceedings of the ACL 2002 Conference on Empirical Methods in Natural Language Processing, p. 10 (July 2002)
4. Compton, P., Jansen, R.: A philosophical basis for knowledge acquisition. Knowl. Acquis. 2(3), 241–257 (1990)
5. Edwards, G., Compton, P., Malor, R., Srinivasan, A., Lazarus, L.: Peirs: a pathologist maintained expert system for the interpretation of chemical pathology reports. Pathology 25, 27–34 (1993)
6. Gaines, B.R.: An ounce of knowledge is worth a ton of data: Quantiative studies of the trade-off between expertise and data based on statistically well-founded empirical induction. In: Proceedings of the 6th International Workshop on Machine Learning, pp. 156–159 (June 1989)
7. Kang, B., Compton, P., Preston, P.: Multiple classification ripple down rules: Evaluation and possibilities. In: Proceedings of the 9th AAAI-sponsored Banff Knowledge Acquisition for Knowledge Based Systems Workshop, pp. 17.1–17.20 (1995)
8. Kim, Y.S., Kang, B.H., Choi, Y.J.: Incremental Knowledge Management of Web Community Groups on Web Portals. In: 5th International Conference on Practical Aspects of Knowledge Management, Vienna, Austria, pp. 198–207 (2004)

9. Klein, S., Simmons, R.F.: A computational approach to grammatical coding of english words. ACM 10(3), 334–347 (1963)
10. Martinez-Bejar, R., Ibanez-Cruz, F., Le-Gia, T., Cao, T.M., Compton, P.: Fmr: An incremental knowledge acquisition system for fuzzy domains. In: Fensel, D., Studer, R. (eds.) EKAW 1999. LNCS (LNAI), vol. 1621, pp. 349–354. Springer, Heidelberg (1999)
11. Pham, S.B., Hoffmann, A.: Efficient knowledge acquisition for extracting temporal relations. In: Proceedings of the European Conference on Artificial Intelligence (ECAI), Riva del Garda, Italy, pp. 521–525 (2006)
12. Ramshaw, L.A., Marcus, M.P.: Text chunking using transformation-based learning. In: Proceedings of the Third Workshop on Very Large Corpora, pp. 82–94 (1995)
13. Ratnaparkhi, A.: A maximum entropy model for part-of-speech tagging. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, vol. 1, pp. 133–142 (1996)
14. Richards, D.: Two decades of ripple down rules research. The Knowledge Engineering Review 24(2), 159–184 (2009)
15. Samuel, K., Carberry, S., Vijay-Shanker, K.: Dialogue act tagging with transformation-based learning. In: Proceedings of the 17th International Conference on Computational Linguistics (August 1998)
16. Scheffer, T.: Algebraic foundations and improved methods of induction or ripple-down rules. In: Proceedings of the 2nd Pacific Rim Knowledge Acquisition Workshop, Sydney, Australia, pp. 279–292 (1996), ISBN: 0-7334-1450-8
17. Shen, L., Satta, G., Joshi, A.K.: Guided learning for bidirectional sequence classification. In: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pp. 760–767 (June 2007)
18. Spoustová, D., Hajič, J., Raab, J., Spousta, M.: Semi-supervised training for the averaged perceptron pos tagger. In: EACL 2009: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (March 2009)
19. Suryanto, H., Compton, P.: Invented predicates to reduce knowledge acquisition. In: Motta, E., Shadbolt, N.R., Stutt, A., Gibbins, N. (eds.) EKAW 2004. LNCS (LNAI), vol. 3257, pp. 293–306. Springer, Heidelberg (2004)
20. Toutanova, K., Klein, D., Manning, C., Singer, Y.: Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology, vol. 1, pp. 173–180 (2003)
21. Wada, T., Motoda, H., Washio, T.: Knowledge acquisition from both human expert and data. In: Cheung, D., Williams, G.J., Li, Q. (eds.) PAKDD 2001. LNCS (LNAI), vol. 2035, pp. 550–561. Springer, Heidelberg (2001)