TRIPPER: Rule learning using taxonomies

Flavian Vasile, Adrian Silvescu, Dae-Ki Kang, Vasant Honavar

Artificial Intelligence Research Laboratory
Department of Computer Science, Iowa State University
Ames, IA 50011 USA
flavian, silvescu, dkkang, honavar@cs.iastate.edu

Abstract

In many application domains, there is a need for learning algorithms that generate accurate as well as comprehensible classifiers. In this paper, we present TRIPPER - a rule induction algorithm that extends RIPPER, a widely used rule-learning algorithm. TRIPPER exploits background knowledge in the form of taxonomies over values of features used to describe data. We compare the performance of TRIPPER with that of RIPPER on a text classification problem (using the Reuters 21578 dataset). Experiments were performed using WordNet (a human-generated taxonomy), as well as a taxonomy generated by WTL (Word Taxonomy Learning) algorithm. Our experiments show that the rules generated by TRIPPER are generally more accurate and more concise (and hence more comprehensible) than those generated by RIPPER.

1. Introduction

Knowledge discovery aims at deriving patterns from data that are both accurate and comprehensible. Most existing methods exhibit trade-offs between these competing goals. One class of algorithms that output fairly interpretable results is the class of rule learning algorithms. The resulting sets of rules tend however to be quite large and too specific. In this context, we present a rule induction method that is able to output rules at higher levels of abstraction. This is accomplished by using knowledge available in the form of taxonomies.

The taxonomies are either based on preexisting knowledge in the particular domain under study or can be generated in automated fashion.

This new rule induction method named TRIPPER (Taxonomical RIPPER) will allow us to obtain more concise rulesets that lead to improvements in generalization accuracy in most the experiments.

The rest of the paper proceeds as follows: In the next section we start by presenting the RIPPER algorithm followed by its taxonomical augmentation, TRIPPER. Then, in section 3 we present a set of experiments on ten text classification tasks based on the Reuters 21578 dataset. We then summarize the results and conclude with a discussion and further work.

2. Methods

Ripper stands for *Repeated Incremental Pruning to Produce Error Reduction*, and was proposed by William W. Cohen as an optimized version of IREP in (Cohen 95).

As most of the Inductive Logic Programming methods, the form of hypothesis that RIPPER attempts to find is a disjunction of conjunctions that accurately classifies the training data as in the following example:

((money = true) and (bank = true) and (billion = true)
or
((rate=true) and (dollars=true)) → *is_interest*

The disjunction can also be seen as a set of rules that determine the class of an instance.

((money = true) and (bank = true) and (billion = true) → *is_interest*
((rate=true) and (dollars=true)) → *is_interest*

Example 1: Example of a small set of rules for classifying a document as part of class "interest" (text classification task from Reuters 21578)

2.1. Overview of the RIPPER-k algorithm:

As stated in (Cohen 95), the majority of rule-learning algorithms are based on the principle of "overfit-and-simplify". More exactly, they greedily cover the training data with rules and then they simplify the rules based on a part of the training set that was not used for learning the rules (the pruning set).

RIPPER (fig. 1) consists of two main stages: the first stage constructs an initial ruleset in a manner similar to IREP (*incremental reduced error pruning)* presented in (Fürnkranz and Widmer 94); the second stage further optimizes the ruleset initially obtained. The stages are repeated for k times.

RIPPER_k(Data)
begin
RuleSet := IREP(Data)//Stage 1*
repeat k times:
RuleSet := Optimize(RuleSet,Data)//Stage 2
UncovData:=examples in Data not covered by rules in RuleSet
RuleSet := RuleSet+ IREP(UncovData)*
endrepeat
end

Fig. 1. Ripper-k pseudocode

Stage 1 consists in running the IREP* procedure on the instances not covered previously. IREP*(fig.2) is an improved version of IREP and was presented in (Cohen 95). Both IREP and IREP* are covering algorithms, meaning that they construct one rule at a time and then remove all the examples that the rule covers from the dataset.

Stage 2 consists in the ruleset optimization that tries to improve the rules obtained in stage 1 by constructing two alternatives for each of the initial rules and by selecting the best one in terms of description length.

The alternatives of a rule are:

The replacement for rule i: A rule i' is grown and pruned such that the error of the entire set of rules is minimized on the pruning data.

The revision of rule i: A rule i" is constructed by adding conditions to i and then pruning it such that the error of the entire set of rules is minimized on the pruning data.

After stage 2, the ruleset can cover fewer positive examples so IREP* can be called again. Usually a second call is enough but this step can be also repeated for as many times as needed (Cohen 96).

Since TRIPPER concentrates on improving the first stage of the RIPPER algorithm, a more in-depth description of IREP * is necessary.

IREP* is called inside RIPPER-k for k times and for each iteration, the current dataset is randomly partitioned in two subsets, a growing set, that usually consists of 2/3 of the examples and a pruning set, consisting in the remaining 1/3. IREP* algorithm has two main phases that make use of the two training subsets we mentioned: the growing set is used for the initial rule construction (the rule growth phase) and the pruning set is used for the second phase (the rule pruning phase) (fig. 2., lines 5 and 6).

The solution used in IREP* for a stopping criterion is to use the MDL score, similar to the one used by Quinlan in Foil and C4.5 and described in (Quinlan, 95). After a rule is added, the algorithm computes the description length of the ruleset and of the examples and stops adding rules once the new description length is larger than the old one by more than d bits.

IREP*(Data)
begin
1. Data0 :=copy(Data)
2. RuleSet := an empty ruleset
3. while Exist pos examples in Data0 do
4. split Data0 in GrowData, RuneData
*5. Rule:=**GrowRule**(GrowData)*
*6. Rule:=**PruneRule**(Rule,PruneData)*
7. add Rule to RuleSet
8. remove examples covered by Rule from Data0
9. if DL(RuleSet)>DL(Ruleset)+d where RuleSet* has lowest DL*
10. then RuleSet:=Compress(RuleSet,Data0)
11. return RuleSet
12. endif
13. endwhile
14. RuleSet:=Compress(RuleSet,Data0)
15. return RuleSet;
end

Fig. 2. IREP* Pseudocode – from (Cohen, 95)

The rule growth phase: The initial form of a rule is just a head (the class value) and an empty antecedent. At each step, a condition is added to the antecedent. This is the process of specializing the rule, starting from the most general rule, that always predicts the class value, and continuing with more specific rules that predict the class value if and only if the conditions in the antecedent are simultaneously satisfied. That is why a longer rule is more specific.

The condition added at each step is selected from all the possible conditions based on its information gain. The stopping criterion for adding conditions is either obtaining an empty set of positive instances that are not covered or a negative information gain score.

The rule pruning phase: Pruning is an attempt to prevent the rules of being too specific. Pruning is done accordingly to a scoring metric denoted by *v**.

As we can see in fig. 3, IREP* chooses the candidate literals for pruning by computing the *v** score for each of the prefixes of the rule on the pruning data. The score is the defined as:

$$v^*(rule, prunepos, prunenef) = \frac{p-n}{p+n}$$

where the maximal value of v* is associated with the set of final conditions in the rule that will be pruned and p/n denote the total number of positive/negative instances covered by the rule

PruneRule(Rule,PruneData)
Begin
Size=Size(Rule)
*Score=V*_Score(Rule,PruneData)*
For i:=0 to size
TempRule:=Prefix(Rule,i)
*If(V*_Score(TempRule,PruneData)>Score)*

```
    Score=V*_Score(TempRule,PruneData)
    Prune_pos=i
  Endif
Endfor
Return Prefix(Rule,prune_pos)
End
```

Fig. 3. PruneRule pseudocode

2.2. TRIPPER – Taxonomical Ripper

In this subsection we describe two possible methods of incorporating taxonomical knowledge in the IREP* algorithm.

In order to clearly define these methods, it is helpful to introduce a formal definition of taxonomy.

Taxonomy:
Let S = {v1, v2, ... vn} be a set of feature values. Let T be a directed tree where children(i) denotes the set of nodes that have incoming arrows to the node i. A node i is called leaf if it has no children.
A taxonomy Tax(T,S) is a mapping which assigns to a node i of the tree T a subset S' of S with the following properties:

$$Tax(T,S)(i) = \bigcup_{j \in children(i)} Tax(T,S)(j)$$

$$Leaves(T) = S$$

The resulting tree is "labeled" at leaf level with distinct elements from S, covering the entire set S, and at upper levels, with the union of the labels of its children.

Note: Root(T) has the meaning of "any"

Given the formal definition we present first a method to incorporate taxonomies at the rule-growth phase, followed by a method to incorporate taxonomies at the rule-pruning phase.

Rule Growth Phase - TRIPPER(G)

Introducing the taxonomical knowledge at the rule-growth phase is a straight-forward process we call **feature space augmentation.**

The augmentation process takes all the interior nodes of the taxonomical tree and adds them to the set of candidate literals for the growth phase (assuming without loss of generality that the basic features are the leaves of the taxonomical tree)

In the resulting set of candidate literals, the abstract features from all the levels of abstraction, compete against the basic features and against each other, based on their information gain measure, computed with respect to the current uncovered instance set.

The results of this approach are presented in Tables 3-1, 3-2 and 3-3.

Rule Pruning Phase - TRIPPER(G+P)

TRIPPER(G+P) is a more involved method of incorporating taxonomical knowledge which occurs at the pruning phase.

The rule resulted from the growth phase can be an overfit. From the point of view of IREP*, this means that the rule may contain spurious features which in principle can be eliminated by a feature selection phase, namely pruning.

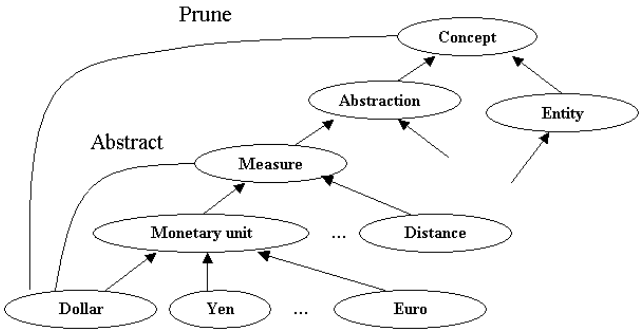

Fig. 4. Taxonomy over a set of nouns. Pruning and abstraction on a taxonomy over the set of features

A more general version of feature selection is abstraction. In the case of abstraction, instead of casting the problem as a matter of preserving or discarding a certain feature, we are able to choose from a whole range of levels of specificity for the feature under consideration.

More exactly, for the case of the taxonomy in fig. 4., the decision to use or not the value *dollar* in the rule is replaced with the decision of which value from the path between *dollar* and the root to use. Selecting the value *dollar* for specializing the rule, versus discarding it, amounts to choosing one of the extremes of the path from "dollar" to the root. In other words, choosing *dollar* amounts to preserving the feature and choosing the root amounts to discarding any information that can be provided by the corresponding feature, which is effectively equivalent with discarding the feature. The effect on the resulting rule can be observed in the following example:

a) (rate = t) and (bank = t) and ***(dollar = t)*** *=> is_interest (original rule)*
b) (rate = t) and (bank =t) and ***(any_concept = t)*** *=> is_interest (pruned rule)*
*c) (rate = t) and (bank = t) and (****monetary_unit= t)*** *=> is_interest (abstracted rule)*

Example 2: Variants of a classification rule for the class "interest"

In this setting, the classical pruning operation of removing a literal from the rule will be replaced with the action of setting the abstraction level of the literal to a node on its abstraction path, i.e. the path from the literal to the root.

The intuition behind the interest to establish a more general form of pruning (simplification) is that the objective of the second IREP* step is not to shorten the rule at all costs but to generalize it without losing predictive power on the training set. That is why pruning can be, from the point of view of loss of specificity, too "harsh" on the rule: the candidate literals for pruning were initially selected because of their information gain; once pruned, all that information will be lost and all the instances they covered will be left unsupported. To prevent this loss, we might try to replace the literals that proved to be over-specific with more general literals.

The algorithm **Prune_by_abstraction** (fig.5.) uses exactly this idea to incrementally search for useful abstractions for the literals in the suffix to be pruned according to the *v** score of the rule prefixes.

Prune_By_Abstraction (Rule, PruneData)

```
begin
1.  Size=Size(Rule)
2.  Score=V*_Score(Rule,PruneData)
3.  For i:=0 to size
4.     TempRule:=Prefix(Rule,i)
5.     If(V*_Score(TempRule,PruneData)>Score)
6.        Score=V*_Score(TempRule,PruneData)
7.        Prune_pos=i
8.    Endif
9.  Endfor
10.   Improvement=true
11.   Level=0
12.   While(improvement)
13.        Improvement=false
14.        Level++
15.       For j:=Prune_pos to size
16.          Literal=Rule(j)
17.          AbstrRule:=Abstract(Rule, Literal, level)
18.          If(V*_Score(AbstrRule, PruneData)>Score)
19.             Score=V*_Score(AbstrRule, PruneData)
20.             TmpRule=AbstrRule
21.             Improvement=true
22.          Endif
23.       Endfor
24.       If(improvement)
25.          Rule=TmpRule
26.  Endwhile
27. Return Rule
end
```

Fig. 5. Prune by Abstraction pseudocode

The algorithm gradually generalizes the rule by abstracting and testing each one of the literals that are candidates for pruning: if one of the abstractions increases the *v** score of the rule, the literal is replaced with its parent in the taxonomic tree. The iteration of the abstraction step on the suffix stops when none of the current literals has a useful abstraction.

2.3. Sources for taxonomies

The assumption of having a taxonomy for the target domain may seem at first to be somewhat restrictive. However, there are numerous current initiatives in a variety of fields to create domain-specific taxonomies. Examples range from: word taxonomies such as WordNet (Fellbaum, 98), molecular biology taxonomies for describing many aspects of macromolecular sequence, structure, and function such as those developed by the Gene Ontology Consortium (Ashburner 2000), area of interest taxonomies such as Yahoo Directory, Open Directory Project and other field-related taxonomies incorporated as parts of the Semantic Web Schemas developed by W3C (Berners-Lee et al. 2001).

The examples listed so far are part of the class of human-designed taxonomies. Their main characteristics are that they are usually general, easily interpretable, and accepted by their specific community.

Apart from human-designed taxonomies, methods for automatic extraction of taxonomies from the training data are already developed. One such method is presented in (Kang et al., 2004). The algorithm, called *Word Taxonomy Learner* (WTL) automatically generates a word-taxonomy from data by clustering words based on their class conditional distribution. The intuition behind the algorithm (fig. 6) is that, with respect to the classification task, the features with similar class conditional probabilities have the same behavior and can be grouped in one abstract feature, which becomes the parent of the original features in the taxonomic tree.

WTL:

```
begin
1. Input : data set D
2. For each word w in S :
3.    For each class c in C :
4.    Estimate the probability distribution p (c,w)
5.    Let P(C|w) = {p(c|w) , ... , p(c|w)} be the class
distribution associated with the word w.
6.    S' =S;
7.    Initialize Tax(T,S) with nodes in S'
8.    Iterate until |S'|=1
9.       In S', find (x, y) = argmin {DM (P (C|x) ||P (C|y))}
   (DM stands for Jensen Shanon Divergence
Measure)
10.      Merge x and y (x ≠ y) to create a new value z.
11.      Calculate probability distribution P (C| z).
12.      S" :=S' U {z} \ {x,y}
13.      Update Tax(T,S) by adding node z as a parent of x
         and y.
14.      S':=S"
15. Output : Tax(T,S)
end
```

Fig. 6. Word Taxonomy Learner pseudocode

The characteristics of the taxonomies that are obtained with methods similar with WTL are that they are dataset specific, consistent with regard to the extraction method

assumptions. In some of the cases, they may not be as intuitive as the human counterparts but are often more insightful for that reason.

3. Experiments

In this section we present experimental results that support the claim that the proposed approach of generalization by abstraction can lead to better results than rule pruning, from the point of view of the resulting rules' compactness and accuracy on unseen data.

3.1. TRipper vs Ripper on Reuters 21587 Text Categorization Test Collection

Reuters 21578 dataset:
For our experiments we used Reuters 21578 dataset with the generally used ModApte split of the dataset in training and testing datasets. Following the experimental setup from (Dumais et al., 98), (McCallum and Nigam, 98) , we run our experiments only on the ten biggest classes in the dataset (*in which 9603 stories are used for training and 3299 for testing)*, without removing stopwords, stemming or using any other form of text-related preprocessing method. Instead we use the mutual information score MI(x,c) of each word x with the class c for feature selection:

$$MI(x,c) = \sum_{x}\left\{\sum_{c}\left\{P(x,c)\log\frac{P(x,c)}{P(x)P(c)}\right\}\right\}$$

As a result of this step we keep for each classifier the top 300 features.

The goal of the first set of experiments was to empirically establish that the use of taxonomies can improve the results of rule-induction algorithms. In order to support this claim we ran various forms of TRIPPER and RIPPER on the Reuters dataset.

WordNet – human taxonomy for lexical knowledge:
The taxonomies used for our first set of experiments come from WordNet, a state-of-the-art lexical database developed by the Cognitive Science Laboratory at Princeton University at the initiative of George A. Miller. WordNet is currently used in many applications from the fields of Computational Linguistics, Natural Language Processing, Information Extraction and Text Classification. (Fellbaum, 98). It comprises the majority of the English nouns, verbs, adjectives and adverbs, organized under synsets, each representing an underlying lexical concept. These concepts are linked by multiple types of lexical links such as hyponymy, hypernimy, antonimy, etcetera. Since we were interested in obtaining a taxonomy we used only the hypernimy relation that stands for "kind-of" or "isa" relation between concepts.

The **variants of TRIPPER** used in the experiments are:

- TRIPPER(G) – Ripper improved with taxonomies only at growth stage;
- TRIPPER(G+P) – Ripper improved with taxonomies at growth and pruning stage;
- TRIPPER(down) – Tripper(G+P) with top-down improvement of literals (starting with the most general form of the literal (the root) and specializing until no improvement in v* of the rule is found)
- TRIPPER(up) - Tripper(G+P) with bottom-up improvement of literals (the standard setting)

The classification results report the break-even point which is a standard measure of the quality of a text classifier and returns the average of precision and recall when the difference between the two is minimum.

$$\text{Precision} = \frac{TruePositives}{TruePositives + FalseNegatives}$$

$$\text{Recall} = \frac{TruePositives}{TruePositives + FalsePositives}$$

where:
TP = the # of documents correctly accepted as in class C
FN = the # of documents incorrectly rejected
FP = the # of documents incorrectly accepted
TN = the # of documents correctly rejected

The comparative results obtained on the Reuters dataset (Tables 3-1, 3-2, 3-3) are promising: Tripper(G+P) has better or equal accuracy compared with Ripper in 7 out of 10 classes, better or equal break-even point in 8 out of 10 classes and smaller rules in 7 out of 10 classes. Both of the phases (growth and pruning) generated improvements, giving empirical support for the idea that both of the extensions are useful.

Table 3-1. Comparison of Break-even points (%) between Tripper and Ripper

Class	Break-even point Tripper (G+P,up)	Break-even point Tripper (G,up)	Break-even point Ripper
interest	**71.0**	65.6	58.7
Wheat	**84.5**	**84.5**	83.0
Earn	**95.1**	94.2	94.0
Grain	87.9	**90.6**	**90.6**
Crude	**82.5**	81.4	79.3
Money-fx	**70.4**	64.8	65.3
Ship	**80.9**	75.2	73.0
Trade	58.9	67.5	**68.3**
Acq	**86.3**	85.3	85.3
Corn	**85.7**	**85.7**	83.9

Table 3-2. Comparison of accuracy(%) results between Tripper and Ripper

Class	Accuracy Tripper (G+P)	Accuracy Tripper (G)	Accuracy Ripper
Interest	**97.58**	97.01	96.98
Wheat	**99.50**	**99.50**	99.47
Earn	**96.42**	96.02	96.02
Grain	99.13	99.13	**99.25**
Crude	97.38	97.38	**97.64**
Money-Fx	**96.38**	95.72	95.69
Ship	**98.77**	98.70	98.44
Trade	96.85	96.78	**97.58**
Acq	**94**	93.47	93.80
Corn	**99.63**	**99.63**	**99.63**

Table 3-3. Comparison of ruleset sizes between Tripper and Ripper

Class	Rule size Tripper (G+P)	Rule size Tripper (G)	Rule size Ripper
Interest	14	**13**	16
Wheat	**3**	**3**	5
Earn	**20**	24	28
Grain	**10**	**10**	12
Crude	**14**	19	17
Money-Fx	21	24	**20**
Ship	**9**	13	13
Trade	**12**	13	14
Acq	27	27	**21**
Corn	4	8	**3**

Abstraction versus Specialization.

The experiments also showed that the direction of the search in the taxonomy matters. With TRIPPER(down) we explored an alternative way to abstraction that starts with literals at the root level of abstraction and gradually specializes them in a top-down manner similar to the method for learning decision trees using attribute-value taxonomies presented in (Zhang 2003) (Rule c from example 3.). However, this approach did not outperform our strategy (bottom-up) in terms of ruleset size or accuracy. These findings, along with the short number of abstractions over the literals obtained experimentally come to support our assumption that the literals that are normally pruned contain useful specific information that is located closer to the leaf level. These findings may also suggest that some of the literals that are still pruned because lack of improvement could have a useful abstraction somewhere higher in the tree.

a) Rule found after growing:

(rate = true) and (discount = true) and (reserve = false) and **(dividend = false)** => is_interest=true

b) Rule found after one successful abstraction step:

(rate = true) and (discount = true) and (reserve = false) and **(financial_gain = false)** => is_interest=true

c) Rule found after one successful specialization step:

(rate = true) and (discount = true) and (reserve = false) and **(abstraction = false)**=> is_interest=true

Example 3. Comparison of rules obtain by Abstraction step versus specialization step in the taxonomical tree provided by WordNet

3.2. Human-designed taxonomies versus automatic extracted taxonomies

For our second set of experiments we used the Word Taxonomy Learner (WTL) algorithm to generate taxonomies on the initial set of 300 of features participating in each of the classification tasks.

WTL generates a binary taxonomical implying that all the resulting abstract concepts are unions of two other more specific concepts.

The comparative results between the use of WordNet taxonomies and the use of WTL-generated taxonomies are shown in Table 3-4.

It can be seen that the WTL-taxonomy performs generally better than the human-designed one, by producing smaller or equivalent rulesets in 7/10 of the cases, better accuracy in 7/10 of the cases and better break-even points in 6/10 of the cases.

The results show that the algorithm is competitive on artificial taxonomies and therefore can be applied to a variety of domains without the need of prior hand-crafted taxonomical knowledge.

Table 3-4. Comparison of TRIPPER results using different types of taxonomies

Class	Size WN	Size WT	Acc. WN	Acc. WTL	BE WN	BE WT
interest	14	**12**	**97.58**	97.21	**71.0**	67.1
wheat	**3**	6	99.50	**99.53**	84.5	**90.1**
earn	20	**15**	96.42	**97.05**	95.1	**95.7**
grain	10	10	**99.13**	99	87.9	**88.6**
crude	14	**11**	97.38	**99.12**	**82.5**	78.8
money -fx	21	**18**	**96.38**	96.25	**70.4**	68.1
ship	**9**	10	98.77	**98.80**	80.9	79.7
trade	12	**11**	96.85	**97.38**	58.9	**67**
acq	27	**12**	94	**95.72**	86.3	**89.2**
corn	**4**	5	99.63	**99.60**	85.7	**87.5**

BE = break-even point (%)
Acc. = accuracy (%)
WN = WordNet
WT=Word Taxonomy Learner

3.3 The comprehensibility of the resulting rules

The generalization step in TRIPPER makes the algorithm more appealing to the non-tehnical user than the normal rule-induction algorithm, since it results in rules that encompass the overall regularities of the category instead of a simple listing of specific features.

In the case of artificially-generated taxonomies, there is still a question of the degree of the comprehensibility of the results. However, we found that the resulting rules from TRIPPER using a WTL-taxonomy had a very intuitive form, as seen in the following example:

(wheat = true) and (((weather+(cover+(tonnes+ecus)))+(ranged+(bids+(china+(farmers+(growers+tonne)))))) = true) and (corn = false) => is_wheat=true

(wheat = true) and (((debt+dollar)+((rates+after)+((31+dlr)+(reserves+(billion+(exchange+rate)))))) = false) => is_wheat=true

(wheat = true) and ((cover+(tonnes+ecus)) = true) => is_wheat=true

(wheat = true) and ((92+competitive) = true) => is_wheat=true

((glickman+((cereal+(subproducts+deficiency))+((subsidized+barley)+((laydays+semolina)+(shultz+enhancement))))) = true) and (((lanka+farmer)+(soviets+((henan+sown)+(winterkill+oats)))) =true) => is_wheat=true

Example 4. Classification rules for class "wheat" using WTL-generated-taxonomy.

4. Conclusions

4.1. Summary.

TRIPPER is an extension of the previous rule-induction algorithm RIPPER, that replaces the method of avoiding overfiting by pruning with the more general method of abstraction guided by a taxonomy over the features. The experiments showed that TRIPPER has improved results compared with RIPPER on the Reuters dataset.

A comparison between results using human-designed taxonomies and results using automatic extracted taxonomies has also been made, suggesting that automatic extracted taxonomies are actually more useful for the classification task than their human counterpart.

We believe that the intuitive conceptual foundation of the algorithm together with the reduction in ruleset length and good accuracy performance makes it an excellent candidate for real-world classification tasks, where the results and the process of obtaining the results can be understood by non-technical users.

4.2. Related work.

The use of taxonomies has been tried in various contexts. Among the closely related work, Zhang and Honavar (Zhang et al., 2003, 2004, 2005) developed a decision tree learner and a Naive Bayes learner regularized over feature value taxonomies mainly for multivariate datasets.

In rule learning, Taylor et al. (1997) described the use of taxonomy in rule learning , Han and Fu (1996) developed a method for using hierarchical background knowledge for association rules learning. Another approach belongs also to Sasaki and Haruno (1995) that propose a relational learner with hierarchical background knowledge.

In more recent work, Michalski (2001, 2004), coined the term of natural induction and established a general framework of attributional calculus, that can be seen as an alternative way of representing rules containing abstractions. Among other important studies in the area of comprehensibility of machine learning algorithms in terms of expressing the results in the forms of rules are the results of Pazzani (1997), Darbari (2000) and Santos et al. (2000).

4.3. Future work.

There are two natural directions of improvement for TRIPPER: the first is to extend the use of taxonomies to first-order representations and the second to research the effect of more involved abstraction search strategies in the taxonomical trees such as the use of a combination of abstraction and specialization steps.

For the text classification task we also plan to compare TRIPPER with state-of-the-art methods such as Naïve Bayes, Decision Trees and SVM. This approach usually demands the use of the entire set of features as in (Cohen, 96).

References

Ashburner M, et al. 2000. Gene ontology: tool for the unification of biology. *The Gene Ontology Consortium. Nature Genetics 25, pp 25-29.*

Berners-Lee T, Hendler J, and Lassila O, 2001, The semantic web. *Scientific American, pp 35-43.*

Cohen, W. W. 1995a. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning* (Lake Tahoe, CA).

Cohen, W. W. 1996. Learning with set-valued features. In *Proceedings of the 13th National Conference on Artificial Intelligence* (Portland, OR).

D.-K. Kang, A. Silvescu, J. Zhang, and V. Honavar, 2004. Generation of Attribute Value Taxonomies from Data for Data-Driven Construction of Accurate and Compact

Classifiers, *Proceedings of the Fourth IEEE International Conference on Data Mining* (ICDM '04), Brighton, UK,.

Darbari A. 2000 : Rule Extraction from Trained ANN:A survey, Technical report *Institut of Artificial intelligence, Dep. of Comp. Science, TU Dresden* (2000)

Dumais, S., Platt, J., Heckerman, D., Sahami, M. 1998 Inductive learning algorithms and representations for text categorization. In: *CIKM '98: Proceedings of the seventh international conference on Information and knowledge management*, ACM Press (1998) 148-155

Fellbaum, C. 1998. WordNet, An Electronic Lexical Database. *The MIT Press* (with a preface by George Miller)

Fürnkranz, J. and Widmer, G. 1994. Incremental reduced error pruning. In *Proceedings of the 11th Annual Conference on Machine Learning* (New Brunswick, NJ). Morgan Kaufmann Publishers Inc., San Francisco, CA.

Han, J., Fu, Y. 1996 : Exploration of the power of feature-oriented induction in data mining. In *Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., eds.: Advances in Knowledge Discovery and Data Mining*. AIII Press/MIT Press

Quinlan, J. R. 1995. MDL and categorical theories (continued). In *Proceedings of the 12th International Conference on Machine Learning* (Lake Tahoe, CA).

Yutaka Sasaki and Masahiko Haruno, A Relational Learner with Hierarchical Background Knowledge, ECML95 Workshop on Knowledge

McCallum, A., Nigam, K. 1998: A comparison of event models for naive bayes text classification. In: *AAAI-98 Workshop on Learning for Text Categorization*.

Michalski, R.S., 2001 "Natural Induction: A Theory and Methodology, *Reports of the Machine Learning and Inference Laboratory, MLI 01-1*, George Mason University, Fairfax, VA.

Michalski, R. S., 2004, Attributional Calculus: A Logic and Representation Language for Natural Induction, *Reports of the Machine Learning and Inference Laboratory*, MLI 04-2, George Mason University, Fairfax, VA, April, 2004.

Pazzani M, Mani S, Shankle W 1997 Beyond concise and colorful: Learning Intelligible Rules. Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, pp 235-238, AAAI Press.

Santos R., Nievola J., Freitas A.: Extracting Comprehensible Rules from Neural Networks via Genetic Algorithm, Proc.2000 IEEE Symp. On Combination of Evolutionary Algorithm and Neural Network (2000) pp. 130-139, S. Antonio, RX, USA

Taylor, M.G., Stoffel, K., Hendler, J.A. 1997 : Ontology-based induction of high level classification rules. In: *DMKD*.

Zhang, J., Honavar, V. 2003: Learning decision tree classifiers from attribute value taxonomies and partially specified data. In: *the Twentieth International Conference on Machine Learning* (ICML 2003), Washington, DC

Zhang, J., Honavar, V.: AVT-NBL 2004: An algorithm for learning compact and accurate naive bayes classifiers from feature value taxonomies and data. In: *International Conference on Data Mining (ICDM 2004).*

Zhang, J., Kang, D-K., Silvescu, A. and Honavar, V. 2005. Learning Compact and Accurate Naive Bayes Classifiers from Attribute Value Taxonomies and Data. In: *Knowledge and Information Systems*.