Learning Decision Lists with Known Rules for Text Mining

Abstract

Many real-world systems for handling unstructured text data are rule-based. Examples of such systems are named entity annotators, information extraction systems, and text classifiers. In each of these applications, ordering rules into a decision list is an important issue. In this paper, we assume that a set of rules is given and study the problem (MaxDL) of ordering them into an optimal decision list with respect to a given training set. We formalize this problem and show that it is NP-Hard and cannot even be approximated within any reasonable factors. We then propose some heuristic algorithms. We conduct exhaustive experiments to evaluate the performance of the proposed algorithms in practice. In our experiments we also observe performance improvement over an existing decision list learning algorithm, by merely re-ordering the rules output by it. As an aside, we also prove bounds on the worst case performance of the algorithms for variants of the MaxDL problem.

1 Introduction

Rule-based systems have been extensively used for several problems in text mining. Some problems in text mining where rule-based systems have been successfully used are part of speech tagging (Brill, 1992), named entity annotation (Grishman, 1997; Appelt et al., 1995), information extraction (Maynard et al., 2001), question answering (Riloff and Thelen, 2000) and classification (Han et al., 2003; Li and Yamanishi, 1999; Sasaki and Kita, 1998). Several studies have been conducted that compare the performance of rule-based systems and other machine learning techniques with mixed results. While there is no clear winner between the two approaches in terms of performance, the rule-based approach is clearly preferred in operational settings (Borthwick, 1999; Varadarajan et al., 2002; Cunningham et al., 2002). Rule-based systems are human comprehensible and can be improved over time. Therefore, it is imperative to develop methods that assist in building rule-based systems.

A rule-based system consists of a set of rules. These rules can either be manually designed or could be learnt from a training set using rule-induction techniques (J. and G, 1994; Cohen, 1995). Each rule consists of an *antecedent* or *pattern* and a *consequent* or *predicted annotation*. In this paper, we will restrict our attention to a broad class of rules in which the antecedent describes a series of conditions on the input item and the consequent specifies the label that applies to instances *covered* by the antecedent. The conditions could also be expressed as patterns in regular or more powerful grammars.

In general, rules could be ambiguous, *i.e.*, multiple rules could *cover* an instance. A common approach for resolving this ambiguity is to define an ordering on the rules (Maynard et al., 2001; Borthwick, 1999). A decision list is one such mechanism (Rivest, 1987). A set of rules that are intended to be interpreted in a sequence is called a *decision list*. In other words, a decision list is an ordering of the given set of rules. Given an instance t, the rules are applied in the specified order until a pattern of a rule R covers t. The instance t is assigned the predicted annotation associated with R.

In this paper, we study the problem of arranging a given set of rules into the "best" decision list. Learning decision lists using training data has been studied in the past (Rivest, 1987; J. and G, 1994; Cohen, 1995; Li and Yamanishi, 1999). These methods attempt to simultaneously learn rules and their ordering. Typically they use *separate and conquer* (Witten and Frank, 2005) strategy and order generated

rules as they are discovered. The generation and ordering of rules are not considered as two separate tasks. In contrast, we assume that the rules are given to us and study the problem of arranging them into an optimal decision list, where optimality is determined over a training data set. Our approach is motivated by the observation that in many operational settings, it is easier and preferred to get a set of rules designed by domain experts (Lewis et al., 2003). Alternatively, the set of rules can be determined using existing techniques for rule learning (J. and G, 1994; Cohen, 1995; Califf and Mooney, 1998). The separation of rule ordering from rule generation allows us to analyze the problem of ordering in detail and to develop *effective* methods for rule ordering. We demonstrate the usefulness of the proposed methods for ordering manually designed rules in the task of named entity annotation and machine learnt rules in the task of classification. We also observe performance improvement over an existing decision list learning algorithm, by merely reordering the rules output by that algorithm.

We determine the ordering of the given set of rules based on a training set. A training set consists of a set of pairs (t_i, a_i) where t_i is an instance and a_i is its actual annotation. Given a set of rules and a training data set, we define the problem as follows: *Arrange the rules into a decision list such that maximum number of instances are assigned the correct annotation.* We refer to this problem as the MAXDL problem. We are not aware of any prior work that studies the MAXDL problem. In this paper, we formalize the MAXDL problem. We show that the problem is NP hard and cannot approximated within a factor of $n^{1-\epsilon}$, for any $\epsilon > 0$. We then propose some heuristics and present an experimental study of these heuristics. Our experimental results show performance improvement over an existing decision list learning algorithm, by merely reordering the rules output by that algorithm. We also illustrate the performance improvements obtained by applying our algorithms for ordering named entity annotation rules and classification rules.

Since the general MAXDL cannot be approximated, in the appendix, we study special cases of the problem that are obtained by considering how easy it is to annotate an instance correctly.

In the rest of the paper we formalize the MAXDL problem (§2), show it is NP-hard and can't be approximated withinreasonable factors (§3), and propose heuristics in a greedy framework (§4). We present a experiments in Section§5 and conclude with Section§6.

2 MAXDL Problem Definition and Notations

MAXDL Problem:

The input consists of a set of *instances* $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$, a set of annotations $\mathcal{A}$ and a set of rules $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$. Each rule $R_i = (p, a)$ is a pair, where p is called the *pattern* and $a \in \mathcal{A}$ is called the *predicted annotation*. The patten p will be given as a set $p \subseteq \mathcal{I}$; we say that the instances in p are *covered* by R. The input also includes a mapping $A : \mathcal{T} \mapsto \mathcal{A}$, that provides for each instance t an annotation $A(t)$, called the *actual annotation* of t. The pair $(\mathcal{T}, A)$ is the *training data*.

Given the above input, a *decision list* L is an ordering (i.e. permutation) of the input rules. The list L assigns an annotation to each instance t as defined below. We consider each rule according to the ordering given by L until we find a rule $R_i = (p, a)$ that covers t and assign the annotation a to t. We denote by $L(t)$ the annotation assigned by L to t. Thus, L defines a function $L : \mathcal{I} \mapsto \mathcal{A}$. We say that the list L *correctly annotates* an instance t, if the annotation assigned by L matches the actual annotation of t, i.e., $L(t) = A(t)$.

Given the above input, the MAXDL problem is to to construct a decision list L such that the number of instances correctly annotated by L, is maximized i.e., we want to maximize $|\{t|A(t) = L(t)\}|$.

Notations:

Let $R = (p, a)$ be a rule and t be an instance covered by R. We say that a rule R *correctly covers* t, if $a = A(t)$. Similarly, R said to *incorrectly cover* t, if $a \neq A(t)$.

Let L be a decision list. We say that an instance t is *happy* under L, if L correctly annotates t, i.e., $L(t) = A(t)$. Let $\mathrm{Happy}(L)$ denote the set of instances that are happy under L. Notice that the MAXDL problem asks for a decision list L such that $|\mathrm{Happy}(L)|$ is maximized.

3 NP-Hardness and Inapproximability

In this section, we prove that the MAXDL problem is NP-Hard and also show that the problem cannot even be approximated with any constant factor.

Theorem 1 *The* MAXDL *problem is* NP*-Hard.*

Proof: We give a reduction from the maximum independent set problem (MAXIS), a well-known NP-Hard problem (Garey and Johnson, 1979). Recall that an independent set in a graph refers to any subset of vertices such that no two vertices from the set share an edge. The MAXIS problem is to find the largest independent set in a given undirected graph.

Let $G = (V, E)$ be the input graph having vertex set $V = \{v_1, v_2, \ldots, v_n\}$. We create an instance of the MAXDL problem as follows. For each vertex v_i, we add an annotation a_i to $\mathcal{A}$, an instance t_i to $\mathcal{T}$ and a rule R_i to $\mathcal{R}$. We declare a_i to be the actual annotation of t_i. The predicted annotation of R_i is set to a_i. We define R_i to cover only the instance t_i and the instances corresponding to the neighbors of v_i. Meaning, R_i covers the instances in the set $\{t_i\} \cup \{t_j | (v_i, v_j) \in E\}$. This completes the reduction. We claim that given a decision list L having k happy instances, we can construct an independent set of size k and vice versa. The NP-Hardness of MAXDL follows from the claim. We now proceed to prove the claim.

Consider a decision list L. Notice that for any instance t_i, R_i is the only rule that correctly covers t_i. Take any two different instances t_i and t_j that are happy under L. Without loss of generality, assume that R_i appears before R_j in L. Now, if R_i covers t_j, t_j would be unhappy under L. So, R_i does not cover t_j, which implies that v_j is not a neighbor of v_i (i.e., $(v_i, v_j) \notin E$). Hence, the set $I = \{v_i | t_i \in \mathrm{Happy}(L)\}$ is an independent set of G. We note that $|I| = |\mathrm{Happy}(L)|$.

Conversely, consider an independent set I of G. Let $R(I) = \{R_i | v_i \in I\}$. Form a decision list L by first arranging the rules from $R(I)$ in any arbitrary order followed by arranging the rest of rules in any arbitray order. Notice that for any vertex $v_i \in I$, R_i correctly covers t_i and no other rule appearing before R_i covers t_i. Thus, t_i is happy under L. It follows that $|\mathrm{Happy}(L)| \geq |I|$.

We have proved that the MAXDL problem is NP-Hard. □

We next show that the MAXDL problem cannot be approximated within any reasonable factors. In our NP-Hardness reduction, we had shown that given a decision list L, we can construct an independent set I such that $|\mathrm{Happy}(L)| = |I|$, and vice versa. This means that any approximation algorithm for the MAXDL problem can be translated (by combining it with our NP-Hardness reduction) into an equally good approximation algorithm for the MAXIS problem. Thus the known inapproximability result for the MAXIS problem (Zuckerman, 2006) translates to the following result for the MAXDL problem.

Corollary 1 *If* NP $\neq$ P *then for any* $\epsilon > 0$*, the* MAXDL *problem cannot approximated within a factor of* $n^{1-\epsilon}$*. In particular, the problem is not approximable within any constant factor, unless* NP = P.

4 Heuristic Algorithms for the MAXDL Problem

As the MAXDL problem is hard to approximate, we turn to heuristic approaches. All our heuristics fall into a natural greedy framework, described below.

4.1 A Greedy Framework

Our greedy framework for finding a decision list is as follows. In each iteration we greedily choose a rule and output it. For this purpose, we use some scoring function for assigning scores to the rules and choose the rule having the maximum score. Then the chosen rule is deleted. The process is continued until all the rules are output. The above procedure gives us a decision list. We present this general framework in the Figure 1. The only unspecified part in the above framework is the scoring function. Intuitively, the scoring function tries to measure the goodness of a rule.

Given rule set $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$, instance set $\mathcal{T}$ and the actual annotations $A(\cdot)$
while $\mathcal{R} \neq null$ **do**
  (re)compute scores for each rule in R, based on the scoring function
  select the rule R that has the maximum score
  remove R from the set $\mathcal{R}$
  remove from $\mathcal{T}$ all the instances covered by R
end while

Figure 1: A Greedy Framework for MAXDL problem

The following additional notations will be useful

in defining the scoring functions proposed in this paper.

Notations: Consider a MAXDL input $\mathcal{I}$. For each rule R and instance t, we define the following sets.

$\texttt{Inst}_R = \{t | R \text{ covers } t\}$ $\quad$ $\texttt{Rules}_t = \{R | t \text{ is covered by } R\}$
$\texttt{Inst}_R^+ = \{t | R \text{ correctly covers } t\}$ $\quad$ $\texttt{Rules}_t^+ = \{R | t \text{ is correctly covered by } R\}$
$\texttt{Inst}_R^- = \{t | R \text{ incorrectly covers } t\}$ $\quad$ $\texttt{Rules}_t^- = \{R | t \text{ is incorrectly covered by } R\}$

4.2 Simple Precision Scoring

We now present our first candidate scoring function, which we call *simple precision scoring*. A natural score for a rule R is its *precision*: the fraction of instances covered correctly by R among the instances covered by it.

$$\text{Score}_{\text{SP}}(R) = \frac{|\texttt{Inst}_R^+|}{|\texttt{Inst}_R|} = \frac{|\texttt{Inst}_R^+|}{|\texttt{Inst}_R^+| + |\texttt{Inst}_R^-|}$$

4.3 Weighted Precision Scoring

Under Score_{SP}, the score of a rule R is determined only by the number of instances covered correctly ($|\texttt{Inst}_R^+|$) and incorrectly ($|\texttt{Inst}_R^-|$). The nature of instances are not taken into account. The variants of Score_{SP} proposed here assign weights to instances, based on which the scores are computed. We can construct examples where the Score_{SP} performs badly, while the weight-based methods discussed below produce the optimal list.

We assign weights to the instances based on how easy it is to make them happy. For an instance t, define the *happiness quotient* $h(t)$ to be the fraction of rules that correctly cover t among all the rules that cover t:

$$h(t) = \frac{|\texttt{Rules}_t^+|}{|\texttt{Rules}_t|}.$$

The value $h(t)$ is a measure of how easy it is to make t happy; the larger the value of $h(t)$, it is easier to make t happy. For instance, if $h(t) \approx 1$, then $|\texttt{Rules}_t^+| \approx |\texttt{Rules}_t|$, meaning that almost any rule that covers t will annotate it correctly. Thus, it is easy to make t happy. On the other extreme, if $h(t) \approx 0$, then only a small fraction of the rules that cover t annotate it correctly. Thus it is harder to make t happy.

When we schedule a rule R, the instances in $\texttt{Inst}_R^+$ become happy and those in $\texttt{Inst}_R^-$ become unhappy. Our new scoring functions give credit to R for each instance in $\texttt{Inst}_R^+$ and award a penalty R for each instance in $\texttt{Inst}_R^-$. The credit and the penalty depend on the happiness quotient of the instance. Informally, we want to give more credit R for making hard instances happy; similarly, we want to penalize R for making easy instances unhappy. A natural way of accomplishing the above is to award a credit of $(1 - h(t))$ for each instance $t \in \texttt{Inst}_R^+$ and a penalty of $h(t)$ for each instance $t \in \texttt{Inst}_R^-$. Below, we formally define the above quantities as *gain* and *loss* associated with R. For each rule R, define

$$\begin{aligned} \text{Gain}(R) &= \sum_{t \in \texttt{Inst}_R^+} (1 - h(t)) \\ \text{Loss}(R) &= \sum_{t \in \texttt{Inst}_R^-} h(t) \end{aligned}$$

Based on the above quantities, we define a natural scoring function, called *Weighted Precision*:

$$\text{Score}_{\text{WP}}(R) = \frac{\text{Gain}(R)}{\text{Gain}(R) + \text{Loss}(R)}$$

4.4 Refined Weighted Precision Scoring

Our third scoring function is a refinement of the weighted precision scoring. In Score_{WP}, we compute the happiness quotient of a token by taking in account the number of rules that cover the token and among those the ones that cover it correctly. The refinement is obtained by also considering the nature of these rules.

We define

$$h_{\text{RP}}(t) = \frac{\sum_{R \in \texttt{Rules}_t^+} \text{precision}(R)}{\sum_{R \in \texttt{Rules}_t} \text{precision}(R)}.$$

(Recall that precision of a rule is defined as $\text{precision}(R) = |\texttt{Inst}_R^+|/|\texttt{Inst}_R|$.)

Gain, loss and the scoring function are defined similar to that of Score_{WP}:

$$\begin{aligned} \text{Gain}_{\text{RP}}(R) &= \sum_{t \in \texttt{Inst}_R^+} (1 - h_{\text{RP}}(t)) \\ \text{Loss}_{\text{RP}}(R) &= \sum_{t \in \texttt{Inst}_R^-} h_{\text{RP}}(t) \end{aligned}$$

The scoring function Score_{RP} is defined as

$$\text{Score}_{\text{RP}}(R) = \frac{\text{Gain}_{\text{RP}}(R)}{\text{Gain}_{\text{RP}}(R) + \text{Loss}_{\text{RP}}(R)}$$

5 Experiments

In this section, we describe rule-ordering experiments on different real-world tasks. 1) named-entity (NE) annotation that relied on hand-crafted rules in MUC-6 and MUC-7 – we are not aware of any automatic rule-ordering work for this. 2) The second application we consider is rule-based multi-class text classification. We order rules learnt on benchmark text classification datasets and observe consistent improvements by merely re-ordering rules learnt by other rule learners.

5.1 Named Entity Annotation

Rule-based named entity annotation is a natural instance of a decision list problem. Typically, rule-based NE annotation systems (Cunningham et al., 2002) require rules to be manually written as well as ordered manually. In this section, we show that our proposed rule-ordering algorithms perform better than the natural heuristic. Note that we do not intend to build a rule-based decision list which performs better than existing methods.

Setup: In our problem formulation of MAXDL , the set of instances $\mathcal{T}$ and mapping A from instances to actual annotations, together form a training set. We have access to a set of documents $D = \{d_1, d_2, \ldots, d_m\}$, that have all its named entities annotated. To generate pairs $(\mathcal{T}, A)$ using the set of documents D, let T_{d_i} represent the set of token sequences that are annotated in a document $d_i \in D$. Let $A(t)$ be the actual annotation for an instance $t \in T_{d_i}$. Given a set of rules $\mathcal{R}$ and a document collection D, each rule $R \in \mathcal{R}$ is applied to each document $d_i \in D$. The set of token sequences (instances here) which R covers ($\texttt{Inst}_R$), is included in the set of instances $\mathcal{T}$. For all instances $t \in T_{d_i}$, we add a mapping $t \rightarrow A(t)$ in A. For all other instances $t \in \{\texttt{Inst}_R - T_{d_i}\}$, we have a mapping $t \rightarrow null$ included in A. We perform these additions for each document and rule pair. Finally, we add a rule $R_* = (*, null)$ to the rule set $\mathcal{R}$. The pattern $*$ matches every instance $t \in \bigcup_{R \in \mathcal{R}, R \neq R_*} \texttt{Inst}_R$ and associates a $null$ annotation with the instance.

We report experiments on the MUC-7 NE dataset. We only consider "person name", "organization" and "place name" annotations. We use two different rule sets containing about 30 rules each.

The rules were in the JAPE (Cunningham, 1999) format. The dictionaries for these rules were collected from the GATE (Cunningham et al., 2002) package. We present experiments with Score_{SP}, Score_{WP} and $\text{Score}^{PR}_{\text{WP}}$.

Results: We compare our methods on the MUC-7 training data set as well as the test data set in Table 1 with Score_{SP} as the baseline. We also evaluated $Random$ rule-ordering as a strawman which gave train and test accuracies of 52.5% and 38.3% respectively on rule-set 1, and 48.5% and 41.5% respectively on rule-set 2. All our proposed methods performed about 25% better than $Random$. In all the cases our proposed methods perform better than Score_{SP}. The result shows that our proposed methods generalize better than simple Score_{SP}.

Rule-sets	Accuracy	Score_{SP}	Score_{WP}	$\text{Score}^{PR}_{\text{WP}}$
Rule-set 1	Trng	76.4	76.7	**78.9**
	Test	50.0	52.7	**54.5**
Rule-set 2	Training	70.1	71.6	**73.3**
	Test	49.1	51.4	**52.0**

Table 1: Accuracies (in %) for different algorithms

Dataset (avg. # rules)	Acc--uracy	JRip	Score_{SP}	Score_{WP}	$\text{Score}^{PR}_{\text{WP}}$
la2s (37)	Trng	86.16±0.39	86.02±0.16	86.68±0.16	**87.04±0.17**
	Test	76.93±0.43	77.88±0.16	78.05±0.17	**78.1±0.15**
oh5 (28)	Trng	86.95±0.41	88.26±0.21	88.8±0.16	**89.06±0.17**
	Test	76.43±0.58	79.08±0.37	79.37±0.38	**79.24±0.35**
tr45 (17)	Trng	91.88±0.38	92.61±0.18	92.84±0.23	**93.3±0.21**
	Test	78.9±0.47	80.99±0.29	81.19±0.28	**81.3±0.3**

Table 2: Accuracies (in %) for *RipRules*

Data set	Accu--racy	Multi-class J48	Multi-class NaiveBayes	Score_{SP}	Score_{WP}	$\text{Score}^{PR}_{\text{WP}}$
la2s (18)	Trng	94.75±0.39	85.78±0.29	94.64±0.14	95.9±0.03	**95.99±0.01**
	Test	73.43±0.64	73.68±0.37	78.0±0.21	78.46±0.23	**78.64±0.29**
oh5 (30)	Trng	95.08±0.21	**99.56±0.09**	96.27±0.14	98.43±0.09	98.45±0.09
	Test	78.08±0.76	74.16±0.77	82.72±0.25	83.16±0.24	**83.98±0.26**
tr45 (30)	Trng	97.91±0.11	87.16±1.18	97.71±0.14	98.93±0.06	**98.98±0.05**
	Test	85.25±1.02	69.91±1.33	84.06±0.44	86.1±0.39	**86.42±0.41**

Table 3: Accuracies (in %) for *BinRules*

5.2 Ordering classification rules

In this section, we show another application of our algorithms in ordering classification rules (Witten and Frank, 2005). The antecedent of a classification rule is a series of tests on the input and the consequent gives the class label. Since different rules can assign conflicting classes, rule-ordering becomes important in choosing a correct class. These rules come from a variety of sources and could be hand-crafted or machine-learnt. Machine learnt rules could be generated using association mining (Agrawal and Srikant, 1994), inductive logic programming (Lavrac and Dzeroski, 1994), or Ripper (Cohen, 1995). Even classifiers can be seen as rules, *e.g.*, linear discriminants are rules that assign one of two classes to exclusive partitions of input space. There is great value in approaching text classification in this setting because real-world settings employ hand-tuned rule-based

classifiers (Lewis et al., 2003). Due to domain specificity and unavailability of hand-tuned rules we illustrate rule-ordering on: (1) rules induced by Ripper (Cohen, 1995) (*RipRules*), and (2) a heterogeneous set of rules obtained from naive Bayes and decision trees (*BinRules*).

Setup: We used benchmark text classification datasets (Forman, 2003) available from the Weka site[1]. These multi-class datasets represent 229 binary text classification problems, with positive class size avg. 149, and class skews avg. $1:31$. These are subsets of various benchmark tasks like Reuters, TREC, and Ohsumed (oh). We present only a subset of the results (with only $\mathrm{Score}_{\mathrm{WP}}$ and $\mathrm{Score}^{PR}_{\mathrm{WP}}$) here for lack of space. We report experiments over 10 random $50:50$ train-test splits. The training split is used to learn rules and their ordering. The orderings are evaluated on the test split and average train and test accuracies reported.

Results:

The *RipRules* setting: We induce rules (from the train split) using the JRip implementation in Weka[2] (Witten and Frank, 2005). We apply our various algorithms to merely re-order the rules output by JRip. In Table 2 we present results comparing JRip output with their re-ordered versions obtained from $\mathrm{Score}_{\mathrm{SP}}$, $\mathrm{Score}_{\mathrm{WP}}$ and $\mathrm{Score}^{PR}_{\mathrm{WP}}$. Along with the name of each data set, the average number of rules induced from the training splits are also mentioned in parentheses. The best accuracies are marked in bold. We observe that the re-ordered rule-sets using $\mathrm{Score}_{\mathrm{WP}}$ and $\mathrm{Score}^{PR}_{\mathrm{WP}}$ perform better than both baselines $\mathrm{Score}_{\mathrm{SP}}$ and JRip with lower deviations.

The *BinRules* setting: For an n-class problem we obtain classification rules by training a heterogeneous collection of one-vs-rest binary classifiers. Each classifier is either a naive Bayes or a decision tree classifier trained to discriminate one class from the rest ($2n$ classifiers). We treat each binary classifier as a classification rule that *covers* an instance if the binary classifier assigns its associated class to that instance. In addition, corresponding to every class, we introduce a default classification rule that assigns the associated class to any instance it encounters. This gives us $3n$ rules. We used the naive Bayes and J48 implementations in Weka to obtain binary rules, ordered using $\mathrm{Score}_{\mathrm{WP}}$ and $\mathrm{Score}^{PR}_{\mathrm{WP}}$, and compared with $\mathrm{Score}_{\mathrm{SP}}$ baseline in Table 3. We also show individual classifier accuracy, and the best are marked bold. It is encouraging to note that all our rule-ordering techniques always outperform their multi-class counterparts on the test data set. We outperform the baseline $\mathrm{Score}_{\mathrm{SP}}$ method on all data sets with lower deviations.

[1] http://www.cs.waikato.ac.nz/ml/weka/index_datasets.html

[2] http://www.cs.waikato.ac.nz/ml/weka/

6 Conclusions

In this paper, we formulated and studied the MAXDL problem. We proved the hardness of the problem. We then proposed some heuristic approaches and established the usefulness of our methods experimentally. We observed improved performance in classification task by merely reordering the rules obtained by an existing decision list learning algorithm. In future work, we would like to explore how rule-ordering formulation can be applied to ordering heterogeneous classifiers in the ensemble learning setting.

References

Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast algorithms for mining association rules. In *VLDB*, pages 487–499.

D. Appelt, J. Hobbs, J. Bear, D. Israel, M. Kameyama, D. Martin, K. Myers, and M. Tyson. 1995. Sri international fastus system: Muc-6 test results and analysis. In *MUC6 '95: Proc. of the 6th conf. on Message understanding*.

A. Borthwick. 1999. *A Maximum Entropy Approach to Named Entity Recognition*. Ph.D. thesis, New York University.

Eric Brill. 1992. A simple rule-based part-of-speech tagger. In *Proceedings of ANLP*.

M. E. Califf and R. J. Mooney. 1998. Relational learning of pattern-match rules for information extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*.

William W. Cohen. 1995. Fast effective rule induction. In *ICML*, pages 115–123.

H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2002. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of ACL*.

H. Cunningham. 1999. JAPE: a Java Annotation Patterns Engine. Research Memorandum CS – 99 – 06, Department of Computer Science, University of Sheffield, May.

George Forman. 2003. An extensive empirical study of feature selection metrics for text classification. *JMLR Special Issue on Variable and Feature Selection*, 3:1289–1305.

M. R. Garey and D. S. Johnson. 1979. *Computers and Intractability*. Freeman.

R. Grishman. 1997. Information extraction: Techniques and challenges. In *SCIE '97: Intnl. summer School on Information Extraction*.

Hui Han, Eren Manavoglu, C. Lee Giles, and Hongyuan Zha. 2003. Rule-based word clustering for text classification. In *SIGIR*, pages 445–446. ACM Press.

Furnkranz J. and Widmer G. 1994. Incremental reduced error pruning. In *Machine Learning: Proc. of the Eleventh International Conference*.

Nada Lavrac and Saso Dzeroski. 1994. *Inductive Logic Programming:Techniques and Applications*. Ellis Horwood, New York.

David D. Lewis, Rayid Ghani, Dunja Mladenic, Isabelle Moulinier, and Mark Wasson. 2003. Workshop on operational text classification. In *conjunction with SIGKDD*.

Hang Li and Kenji Yamanishi. 1999. Text classification using ESC-based stochastic decision lists. In *CIKM*.

D. Maynard, V. Tablan, C. Ursu, H. Cunningham, and Y. Wilks. 2001. Named entity recognition from diverse text types. In *RANLP*.

Ellen Riloff and Michael Thelen. 2000. A rule-based question answering system for reading comprehension tests. In *ANLP/NAACL 2000 Workshop on Reading comprehension tests as evaluation for computer-based language understanding sytems*.

Ronald L. Rivest. 1987. Learning decision lists. *Machine Learning*, 2(3):229–246.

Minoru Sasaki and Kenji Kita. 1998. Rule-based text categorization using hierarchical categories. In *Proceedings of SMC-98, IEEE International Conference on Systems, Man, and Cybernetics*, pages 2827–2830.

Sundar Varadarajan, Kas Kasravi, and Ronen Feldman. 2002. Text-mining: Application development challenges. In *Proceedings of the Twenty-second SGAI International Conference on Knowledge Based Systems and Applied Artificial Intelligence*.

Ian H. Witten and Eibe Frank. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann.

D. Zuckerman. 2006. Linear degree extractors and the inapproximability of max-clique and chromatic number. In *STOC*.

A Approximation Ratios for Special Cases of the MAXDL Problem

As shown by Corollory 1, the MAXDL problem cannot be approximated within any constant factors. In this section, we consider special cases of the problem and prove approximation gurantees.

The special cases are obtained by placing restrictions on the nature of instances and rules covering them. For $\alpha > 0$, let α-MAXDL problem be the special case where for each instance t, among the rules covering t at lease α fraction of them cover it correctly. Equivalently, we require that $h(t) \geq \alpha$. By adapting the proof of Theorem 1, we can show that α-MAXDL is NP-hard, for any $\alpha > 0$.

Let $\mathrm{Greedy}[\mathrm{Score}_{\mathrm{WP}}]$ denote the greedy algorithm with $\mathrm{Score}_{\mathrm{WP}}$ as the scoring function. We show that the above algorithm is an α-approximation algorithm for the α-MAXDL problem. The claim follows from a more generic result discussed below.

Consider the scoring function $\mathrm{Score}_{\mathrm{WP}}$. For a MAXDL input $\mathcal{I}$, we define

$$\mathrm{HQ}(\mathcal{I}) = \sum_{t \in \mathcal{T}} h(t)$$

Notice in the case of α-MAXDL inputs $\mathcal{I}$, $\mathrm{HQ}(\mathcal{I}) \geq \alpha$. We prove the following result regarding any MAXDL input.

Theorem 2 *Let $\mathcal{I}$ be any* MAXDL *input and L be the list output by* $\mathrm{Greedy}[\mathrm{Score}_{\mathrm{WP}}]$ *on input $\mathcal{I}$. Then,* $|\mathrm{Happy}(L)| \geq \mathrm{HQ}(\mathcal{I})$

Proof Sketch: The proof uses the following claim.

Claim: In any input $\mathcal{I}$, there exists a rule R^* such that $|\mathtt{Inst}^+_{R^*}| \geq \Sigma_{t \in \mathtt{Inst}_{R^*}} h(t)$.

Proof of Claim: We consider the following sum and derive an equality:

$$\begin{aligned}\sum_{R}\sum_{t\in\texttt{Inst}_R} h(t) &= \sum_{t}\sum_{R\in\texttt{Rules}(t)} h(t)\\ &= \sum_{t}\texttt{Rules}^{+}(t)\\ &= \sum_{R}\texttt{Inst}^{+}_{R}.\end{aligned}$$

The claim is obtained by equating the first and the last terms, and by applying an averaging argument. □

The theorem is now proved via a simple inductive argument on the number of rules in the given input. Let $\mathcal{I}$ be any MAXDL input having n rules and let L be the greedy algorithm's output on $\mathcal{I}$. Let R_1 be the first rule in L and L' be the rest of the list, so that $L = R_1 \circ L'$. Then,

$$|\text{Happy}(L)| = |\texttt{Inst}^{+}_{R_1}| + |\text{Happy}(L')|. \quad (1)$$

We shall derive lowerbounds for the above two terms.

Since L' has one less rule, using the induction hypothesis, we can show that

$$|\text{Happy}(L')| \geq \sum_{t\notin\texttt{Inst}_{R_1}} h(t). \quad (2)$$

Next, consider the term $|\texttt{Inst}^{+}_{R_1}|$. The claim above provides a rule R^* such that $\texttt{Inst}^{+}_{R^*} \geq \Sigma_t h(t)$. Notice that $\text{Score}_{\text{WP}}(R_1) \geq \text{Score}_{\text{WP}}(R^*)$, since the algorithm selected R_1. From this fact, we can show that

$$|\texttt{Inst}^{+}_{R_1}| \geq \sum_{t\in\texttt{Inst}_{R_1}} h(t). \quad (3)$$

Combining Equations 2 and 3 with Equation 1, we get $|\text{Happy}(L)| \geq \text{HQ}(\mathcal{I})$. □

Let $\mathcal{I}$ be a α-MAXDL input having n instances. Then, $\text{HQ}(\mathcal{I}) \geq \alpha n$. It follows from Theorem 2 that $\text{Greedy}[\text{Score}_{\text{WP}}]$ outputs a list having at least αn happy instances. Since the optimal can have at most n happy instances, we get that the above algorithm is an α-approximation algorithm.

Theorem 3 *For any $\alpha > 0$, the $\text{Greedy}[\text{Score}_{\text{WP}}]$ is an α-approximation algorithm for the α-MAXDL problem.*