Lavesson, Niklas; Davidsson, Paul
“AMORI: A Metric-based One Rule Inducer”
SIAM International Conference on Data Mining, 2009, Sparks, Nevada

AMORI: A Metric-based One Rule Inducer

Niklas Lavesson[*] Paul Davidsson[†]

Abstract

The requirements of real-world data mining problems vary extensively. It is plausible to assume that some of these requirements can be expressed as application-specific performance metrics. An algorithm that is designed to maximize performance given a certain learning metric may not produce the best possible result according to these application-specific metrics. We have implemented A Metric-based One Rule Inducer (AMORI), for which it is possible to select the learning metric. We have compared the performance of this algorithm by embedding three different learning metrics (classification accuracy, the F-measure, and the area under the ROC curve), on 19 UCI data sets. In addition, we have compared the results of AMORI with those obtained using an existing rule learning algorithm of similar complexity (One Rule) and a state-of-the-art rule learner (Ripper). The experiments show that a performance gain is achieved, for all included metrics, when using identical metrics for learning and evaluation. We also show that each AMORI/metric combination outperforms One Rule when using identical learning and evaluation metrics. The performance of AMORI is acceptable when compared with Ripper. Overall, the results suggest that metric-based learning is a viable approach.

1 Introduction

A common problem in data mining is that of generating classifiers from data sets consisting of instances where the class membership is known. The objective is to generate classifiers that can be used to accurately predict the class of unknown data instances of the same kind. Supervised concept learning algorithms have shown to be quite reliable in solving this task. As a consequence, supervised concept learners are increasingly applied to solve real-world problems, for example by finding useful patterns in large databases.

1.1 Motivation Experimental machine learning research often involves evaluating algorithms and classifiers using a particular evaluation metric due to its perceived or proven ability to measure predictive performance. However, the requirements of real-world applications often imply that a suitable trade-off between several important criteria is desired [18]. In addition, the importance of different criteria and the notion of what is a suitable trade-off vary between different applications. For example, a certain application might depend on finding classification rules with few antecedents and that achieve a low false positives rate for a particular class while the selected supervised concept learning algorithm might be designed to maximize accuracy. If the class distribution is skewed in such a way that most instances do not belong to the application important class, the generated classifier might actually contribute negatively to the application metrics.

The basic approach for handling this issue is to evaluate a number of algorithms using different metrics for all the important criteria and select the algorithm that is the most successful at balancing the relevant trade-off. For instance, one might decide on the algorithm that is the best at optimizing a certain single or multi-criteria performance metric that captures an application-specific trade-off.

A more sophisticated approach, however, is to try to increase the performance of the candidate algorithms during the actual learning phase by adapting them to optimize metrics relevant to the problem at hand, and then continue with the evaluation to be able to pick the best algorithm.

All supervised concept learning algorithms are biased as a consequence of their data structure (representational bias) and the way they select, or search for classifiers (algorithmic bias). This selection or search is augmented through the use of an embedded performance metric.

We investigate the relationship between the embedded metric and the target evaluation metric. That is, we are interested in finding out if it is possible to achieve better performance, given a certain metric, if the identical metric is also used as the embedded performance metric. If this is indeed the case, this type of *metric-based learning* could provide us with a new way to increase the potential of existing learning algorithms and it may also enhance our understanding of the relation-

[*] Department of Software and Systems Engineering, Blekinge Institute of Technology, Box 520, SE-372 25 Ronneby, Niklas.Lavesson@bth.se.

[†] Department of Software and Systems Engineering, Blekinge Institute of Technology, Box 520, SE-372 25 Ronneby, Paul.Davidsson@bth.se.

ship between the representational and the algorithmic biases.

1.2 Outline

The remainder of this paper is organized as follows. In Section 2 we give a more detailed description of the problem domain. We describe the concept of embedded and evaluation metrics and discuss related work. We continue by describing metric-based learning and present a rule learning algorithm that is based on this concept in Section 3. The empirical experiments and their results are presented in Section 4. Finally, we analyze and discuss the results in Section 5 and end with conclusions and pointers to future work in Section 6.

2 Background

We shall now discuss metrics in general and particularly the difference in using metrics inside algorithms as opposed to using them for evaluation or validation. In the area of supervised concept learning, a metric is commonly used as a basis for evaluating or comparing learning algorithms, or their generated classifiers. The metric tries to capture to which extent an algorithm or classifier meets some given criterion. In a majority of the cases, the criterion is related to performance. There are several performance related criteria, like time and space. However, the most common criterion for machine learning and data mining applications is arguably classification performance. This can be illustrated by the fact that one of the largest empirical studies of machine learning metrics focuses entirely on classification performance metrics [3]. The accuracy metric is often used to measure the performance of a classifier in terms of correctness. However, other criteria like complexity, comprehensibility, and interestingness are sometimes discussed [6, 22].

In many cases, more than one metric can be used to evaluate a certain criterion. We then have to choose the most appropriate metric or take more than one metric into consideration, e.g., by using multi-criteria metrics. It can be difficult to determine how suitable a metric is for assessing a certain criterion. It is quite common to choose a metric on the basis of its appropriateness for a particular application [18].

Finally, it seems to be the case that certain metrics are more popular than others in some areas of research. Some key characteristics of the data sets may vary between different areas and researchers need to investigate which metrics best suit their particular domain. Although, we could speculate that some of the area-specific choices of metrics can be attributed to tradition or a limited knowledge of what has been proven to work in similar research domains.

2.1 Learning Metrics

Given an arbitrary data set it is possible to define the classifier space, that is, the complete set of possible classifiers for that particular data set. The inductive bias of a supervised concept learning algorithm narrows down the classifier space to a subset of classifiers, which can be described by the algorithm representation, and also dictates how to select a classifier from this subset. Thus, we can conceptually divide the inductive bias into two components: the representational bias that delimits the searchable space and the algorithmic bias that dictates how this space is searched.

For example, the network size and structure affect the representational bias of traditional back-propagated neural network learners while the back-propagation weight-adjusting training algorithm affects the algorithmic bias. Similarly, the allowed number of rules and the number of possible antecedents for each rule affect the representational bias of rule inducers while the rule generator and pruning methods affect the algorithmic bias.

An important aspect of these weight adjusters, rule generators, pruning techniques, and other training algorithms, is the ability to determine if one hypothesis is better than another. For this purpose, the algorithms use one or more learning performance metrics.

For example, back-propagated neural networks [28] use mean-squared error or zero-one loss, rule learners such as Ripper [4] and IREP [10] use zero-one loss, coverage and information gain, and the C4.5 decision tree inducer [26] uses information gain and can optionally prune using zero-one loss and complexity.

Whereas the representational bias in most cases is rather explicit and easily understood by the user, the algorithmic bias is often implicit and difficult to grasp. The general goal of this work is to make the algorithmic bias more explicit, simple to comprehend, and even to a certain extent controllable by the user.

The underlying idea is to capture as much as possible of the algorithmic bias in an explicit metric which is used as a replacement for the original learning metric by the algorithm to generate the most appropriate classifier, given a set of application requirements that can be expressed as one or more application-specific metrics. We will refer to this class of supervised concept learning algorithms as metric-based algorithms. Moreover, we will use the term embedded metric, or learning metric, to denote a metric that modifies the algorithmic bias.

2.2 Evaluation Metrics

The traditional method for evaluating classifiers has been to estimate the predictive accuracy via statistical tests, for example, cross-validation [30] or the bootstrap [8]. More recently, it has been shown that performing ROC analysis is more suitable than estimating predictive accuracy for many problems [25] since ROC analysis does not depend on class distribution or misclassification cost.

However, algorithms designed to generate classifiers that maximize classification accuracy may not produce classifiers that achieve the best possible ROC result [5]. Additionally, there are a large number of other metrics, which are more or less appropriate, for different classes of problems [3]. In fact, real-world applications often have to meet several criteria, which would require the use of customized multi-criteria metrics [18]. Therefore, it seems plausible to assume that if the algorithmic bias of a learning algorithm is modified to optimize a particular metric it would possibly yield a better result compared to the original algorithm, when evaluated using that particular metric.

2.3 Definitions We first define the instance space, I, as the set containing all possible instances for a particular problem. Moreover, we define the class space, K, as the set containing all possible classes, or outcomes, for this problem. For the purpose of our study, we may then use the common definition of a classifier, c, as a mapping from instance space, I, to class space, K [20]:

$$c : I \to K \tag{2.1}$$

The classifier space, C, is then defined as the set of all possible classifiers, that is, all possible mappings between instance space and class space:

$$C = I \times K \tag{2.2}$$

Additionally, we define a metric to be a function, m, that assigns a scalar result value, v, to each pair of classifier, $c \in C$, and data set, D:

$$m(c, D) = v \tag{2.3}$$

The inductive bias of an algorithm, a, can be divided into the representational bias and the algorithmic bias. The former limits the classifier space, thus it indirectly generates a subset, $C_a \subset C$, of this space. The latter dictates how to search in, or select from C_a.

The definition of a metric, as given in Equation 2.3, is sufficient for theoretical discussions about some of the most commonly used metrics. However, it is difficult to know what requirements to impose on metrics that are to be used as embedded metrics for algorithms. For example, the learning algorithm itself might restrict which metrics can be used in terms of the range and which information is provided as input for the metric. In addition, one may need to trade off several metrics. To be able to easily change the learning metric of an algorithm we need to add some generic mechanism to allow for embedding alternative metrics without changing the algorithm fundamentally.

In a recent paper [21] we identified a number of attractive properties of existing multi-criteria evaluation metrics and presented a generic multi-criteria metric that we designed with these properties in mind. This metric, called the Candidate Evaluation Function (CEF), has the main purpose of combining an arbitrary number of individual metrics into a single quantity.

CEF normalizes the metrics in order to get a uniform output domain and it is also possible to specify explicit weights for each metric to ensure that application-specific trade-offs can be properly represented. CEF itself does not dictate which metrics to use; it merely dictates how metrics are combined. Finally, CEF makes it possible to specify the acceptable range for each metric, pertaining to a particular application.

We define m_j as a metric with index j from an index set, J, over the selected set of metrics. Each metric is associated with a weight, w_j, and an acceptable range, $r = \left[b_j^l, b_j^u\right]$. The lower bound, b_j^l, denotes the least desired acceptable score. Similarly, the upper bound, b_j^u, denotes the desired score. Note that, in the original CEF definition a metric was normalized according to the best and worst score of that particular metric obtained from the studied set of classifiers. The current normalization uses the lower and upper bound to generate a distribution from 0 (least desired) to 1. CEF is now defined as specified in Equation 2.4.

(2.4)

$$\mathrm{CEF}(c, D) = \begin{cases} 0 : \exists j(\overline{m}_j(c, D) < 0) \\ \sum_{j \in J} w_j \overline{m}_j(c, D) \ \text{otherwise} \end{cases}$$

where $\sum_{j \in J} w_j = 1$ and

$$\overline{m}_j(c, D) = \begin{cases} 1 : \frac{m_j - b_j^l}{b_j^u - b_j^l} > 1 \\ \frac{m_j - b_j^l}{b_j^u - b_j^l} \ \text{otherwise} \end{cases} .$$

2.4 Related Work Several studies have found that optimizing a metric different from the metric being evaluated can bring better results than optimizing the same metric [15, 27, 29, 33]. On the contrary, a recent study claim that such findings come from the use of wrong statistical testing strategy [16]. The common denominator for these studies is that they investigate the relationship between a selection metric and a goal metric. The selection metric is different from the learning metric in that it is used to select classifiers from a set of generated classifiers whereas the learning metric is used during the generation of classifiers. Several papers investigate how to optimize different metrics by embedding them as learning metrics.

The first approach tries to optimize a single metric by replacing the learning metric of popular algorithms. Notable examples of such studies include the optimization of ROC using decision trees [9] and gradient-descent [13], as well as optimization of the F-measure using support vector machines [24].

The second approach aims to optimize more than one metric, either by replacing the learning metric of an existing algorithm with a multi-criteria metric, or by developing a new algorithm that optimizes such a metric. For example, the support vector machines algorithm has been generalized to optimize multi-criteria non-linear performance metrics [18], and dynamic bias selection has been implemented for prediction rule discovery [31].

In addition, one study [1] presents an approach called measure-based evaluation and describes how to implement hill-climbing learning algorithms that optimize a multi-criteria metric, called the measure function. By designing a measure function for a particular problem it is possible to get an explicit distinction between the problem formulation, that is, specifying the measure function, and problem solving: finding a classifier that maximizes the measure function. By making this distinction, it is possible to isolate the meta-knowledge necessary for classifier selection from the details of the learning algorithms.

3 Metric-based Learning

Many researchers have studied the relationship between the algorithmic and the representational bias. For example, see the study on evaluation and selection of biases in machine learning [12] or an earlier study about the need for biases in learning generalizations [23]. Additionally, another study [19] tries to empirically evaluate the generic performance of different learning algorithms by formulating metrics that summarize performance over a large set of classifiers generated by different algorithm configurations.

The focus of this study is close to the mentioned relationship since we investigate the possibility of increasing the performance of classifiers generated by a learning algorithm with a certain representational bias by changing its algorithmic bias.

One question that is only slightly addressed in this study is how much impact this change will have for a certain representational bias, that is, how much potential there is in using a particular representational bias for metric-based learning. Obviously, one has to compare the performance of one particular embedded metric in conjunction with algorithms that use different representational biases in order to address that question more fully.

Researchers in the field of meta-learning have pointed out the importance of going beyond the engineering goal of producing more accurate learners to the scientific goal of understanding learning behavior [11]. We believe that it is possible to move closer to this goal by conducting research into some of the core concepts of metric-based learning. However, in this study we focus on one specific representational bias only.

3.1 A Metric-based One Rule Inducer It has been shown that very simple classification rules perform quite well on most commonly used data sets [14]. The study featured a supervised concept learning algorithm, called One Rule that actually achieved results comparable to those achieved by the C4.5 decision tree inducer. However, the algorithm only used accuracy as a learning metric and the evaluation results were also only provided for the accuracy metric.

We have developed A Metric-based One Rule Inducer (AMORI). As the name suggests the algorithm tries to generate one rule that is optimal, given a selected embedded metric. The pseudo code for the algorithm is given in Figure 1. We have implemented AMORI in Java by extending the Weka [32] *Classifier* class. Thus, the algorithm implementation is compatible with the Weka ARFF data set format and can be evaluated using the Weka *Evaluation* class.

3.1.1 The AMORI Algorithm The algorithm takes a set of training instances as input and generates one rule, consisting of zero or more antecedents and one consequent. Each antecedent consists of an attribute, a conditional operator, and a value. The implementation currently only supports two-class (binary) target attributes and input attributes of types *nominal* and *numeric*, which are handled as follows.

The rule can only use a specific nominal attribute in an antecedent once (using the = operator). However, attributes of numeric type can be used twice, since there are two operators available ($>$ and $\leq$). The possible values, which an attribute can be compared with, are taken from the training set. Thus, it is possible to calculate an upper bound on the number of possible rules that can be generated for a certain data set. The algorithm selects the minority class as consequent and classifies all uncovered instances as belonging to the majority class.

After calculating the default rule score, that is, the performance score for an empty rule (consisting of zero antecedents and one consequent), the algorithm

```
Require: data - a set of training instances
Require: metric - a CEF metric
  consequent $\Leftarrow$ leastFrequentClass
  while running do
    ruleScore $\Leftarrow$ evaluation(data,metric)
    for $att = 0$ to $att <$ numAttributes do
      if $att$ is not used then
        antd $\Leftarrow$ bestAntd(att,data)
        rule.add(antd)
        tempScore $\Leftarrow$ evaluation(data,metric)
        rule.removeLast()
        if tempScore $>$ bestAntdScore then
          bestAntdScore $\Leftarrow$ tempScore
          bestAttribute $\Leftarrow$ $att$
          bestGlobalAntd $\Leftarrow$ antd
        end if
      end if
    end for
    if bestAntdScore $<$ ruleScore then
      running $\Leftarrow$ false
    else
      rule.add(bestGlobalAntd)
      mark bestAttribute as used
    end if
  end while
```

Figure 1: Pseudo code for A Metric-based One Rule Inducer (AMORI)

performs an exhaustive search in which each attribute is matched with the compatible operator(s) and the values that exist for the attribute in the training set. Each possible antecedent is added to the rule, prior to a temporary metric evaluation, and then removed again. When the exhaustive search is complete, the best found metric score is compared to the overall rule score (which is the default rule score during the first run).

If the best found metric score is worse than the overall rule score, the training is stopped. Otherwise, the corresponding antecedent is added to the rule and an exhaustive search is performed yet again, on the remaining attributes. For each attribute, the algorithm checks to see if it has been used before and omits any evaluation using that particular attribute if it is already in use, according to the usage restrictions expressed earlier.

The generated classifier assigns the class label that corresponds to the consequent of the created rule for instances that are covered by the rule. If an instance is not covered by the rule, the classifier instead assigns the majority class.

3.1.2 AMORI Complexity The representational bias of AMORI limits the classifier space, C, into a subset, C_{AMORI}. In order to calculate the number of possible classifiers, we first define a general rule for a data set, D, of x attributes:

$$r = \alpha_1 \wedge \ldots \wedge \alpha_x \tag{3.5}$$

The possible number of antecedents for a particular nominal attribute, y, can then be defined as $|\alpha_y| = |V_y| + 1$, where V_y is the set of possible values, found in the training set.

Similarly, the number of possible antecedents for a particular numeric attribute, z, can then be defined as $|\alpha_z| = 2\,|V_z| + 1$, where V_z is the set of possible values, found in the training set. The addition of 1 to each of these sizes is used to represent the case when the particular attribute is not used at all.

The size of C_{AMORI} can then be defined as:

$$|C_{\text{AMORI}}| = |r| = \prod_{l=1}^{l<x} |\alpha_l| \tag{3.6}$$

3.2 Metrics for Metric-based Algorithms Machine learning evaluation metrics are predominantly used as components in classifier evaluation methods. If instead, they are to be used as embedded metrics there are a number of issues that need to be addressed.

3.2.1 Required Number of Instances First of all, it is important to recognize the fact that some metrics are calculated on a per instance basis, while others summarize the performance over a set of instances. For example, a back-propagated neural network learner processes the data set one instance at a time, which works well in conjunction with the regular learning error metric.

However, in order to employ metrics such as the area under the ROC curve (AUC), which requires evaluation and ordering of sets of instances, one can only use algorithms that evaluate performance on sets of instances. Consequently, if a back-propagated neural network learner must be used, the choice of embedded metric is restricted to those metrics that can be calculated on a per instance basis and therefore we cannot use AUC.

3.2.2 Required Types of Information Additionally, there exist metrics that require a completely generated classifier for calculation, whereas other metrics can be calculated during the learning process. For example, the Ripper rule learner creates a rule set by generating one rule at a time. When the complete rule set has been generated the algorithm calculates some statistics for each rule and these statistics can be used to give the probability that a particular instance belongs to a certain class.

Consequently, it is important to consider that the embedded metric can be put in several locations within the algorithm. One approach would be to put it at the generation of each rule or during the pruning of individual rules. Another approach would be to put it in the optimization stage of the algorithm. Depending on how the metric is integrated, Ripper will provide a different set of information elements for metric calculation.

The statistics and information that can be extracted to be used for calculating metrics also vary between different learning algorithms. For example, the area under the ROC curve metric depends on a ranking of data instances according to the probability that they belong to a certain class. Although many classifiers can provide this probability information, some algorithms can only output classifications.

4 Experiments

Our main experiment features three instances of AMORI, each using a different embedded metric. The objective of this experiment is to determine if the use of identical evaluation and embedded metrics yields a higher performance in comparison to the use of one particular embedded metric independently of which metric is used for evaluation. In a complementary experiment we also compare the performance of AMORI with some existing rule induction algorithms.

In essence, we want to test the null hypothesis that, for a given evaluation metric, the difference in performance between any of the included algorithms is zero. Since testing this hypothesis involves the comparison of more than two classifiers on multiple data sets we use the non-parametric Friedman test and the corresponding Nemenyi post hoc test [7].

The Friedman test is based on the average ranks of each algorithm rather than average performance. The best performing algorithm is awarded the rank of 1, the second best the rank of 2, and so on. In the case of a tie, the average ranks of the tied algorithms are assigned. We can now state the null hypothesis more formally. Given that R_j represents the average rank of the j-th out of k classifiers, we wish to test $H_0 : R_1 = \cdots = R_k$.

Since it has been shown that Friedman's statistic, χ^2_F, is too conservative [17] we employ the recommended alternative statistic based on the F-distribution, as defined in Equation 4.7 where N is the number of included data sets and k is the number of algorithms to compare. Moreover, we perform hypothesis testing at $p < 0.05$ and with $k-1$ and $(k-1)(N-1)$ degrees of freedom.

$$F_F = \frac{(N-1)\,\chi^2_F}{N\,(k-1) - \chi^2_F} \tag{4.7}$$

If the null hypothesis is rejected, the Nemenyi post hoc test can be used to find out if the performance of two particular classifiers is significantly different. In order for this difference to be significant, the corresponding average ranks of the two classifiers must differ by at least the Critical Difference (CD), which is defined in Equation 4.8, where q_α is the critical value for the two-tailed Nemenyi test at $p < \alpha$.

$$CD = q_\alpha \sqrt{\frac{k\,(k+1)}{6N}} \tag{4.8}$$

We use stratified 10-fold cross-validation tests to measure the performance of each algorithm according to our three selected metrics. The partitioning into folds is carried out by the Weka *Evaluation* class using a random seed of 1 for each cross-validation test. Thus, given a particular data set, each algorithm has access to identical folds. In essence, this setup also increases the reproducibility of the results.

We now proceed by first presenting the three metrics that are used for the evaluation of all included algorithms and as embedded metrics for AMORI. We then explain the process of data set selection, present the featured data sets, and give a more detailed description of each experiment.

4.1 Metrics Many of today's most frequently applied metrics are used for evaluating the same basic trade-off, consisting of different proportions of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) [32]. The question of which trade-off is more useful to consider varies across different problem domains. For instance, the F-measure (FME) and precision vs. recall are quite common in the area of information retrieval while sensitivity and specificity are more prominent in medical applications. Classification accuracy (ACC), or success rate, has traditionally been the most commonly used metric for evaluating the before mentioned trade-off.

4.1.1 Classification Accuracy The main question that we try to answer in this study is whether it is possible to achieve a performance gain with respect to a certain metric if that particular metric is also used as a component of the algorithmic bias. Thus, the decision on which metrics to incorporate is important. Our first choice is to include classification accuracy as it is still one of the most popular evaluation metrics in machine learning. Coincidentally, it is also frequently used as a learning metric.

4.1.2 The Area under the ROC Curve As the first alternative metric, we select the area under the ROC curve metric (AUC), which evaluates the same trade-off as expressed above but in a quite different way compared to the accuracy metric. AUC has become a popular metric for evaluating supervised concept learners because, as opposed to ACC, it does not depend on equal class distribution or misclassification costs. Many researchers argue that ROC analysis is better than, or at least complements, predictive accuracy estimation as a tool for evaluating classifiers. Although some information is lost when summarizing a ROC curve using this single metric instead of analyzing the actual curve, AUC has still been proven to work quite well as an evaluation metric.

4.1.3 The F-measure The third and last metric we choose is the F-measure. Again, it tries to capture the same basic trade-off as ACC and AUC do but it is most prominently used in information retrieval. The three selected metrics have a lot in common but what separates them is how they capture the basic trade-off and the preconditions that must be met in order to use them. This makes them good candidates for our particular study.

4.1.4 Conversion to CEF In order for ACC, AUC, and FME to be used in AMORI they need to be converted to CEF metrics. For the purpose of this particular study, the conversion is effortless. Each metric already has a range of $[0, 1]$ and we do not need to use weights since each CEF instance will only include one metric. However, the abstraction of CEF in AMORI makes it straight-forward to incorporate multiple metrics without changing the algorithm. Essentially, CEF can be seen as a container for metrics. The motivation for not using multi-metric CEF metrics in this paper is that the primary objective is to investigate whether there is a correlation in performance between the learning metric and the evaluation metric.

4.2 Data Sets The experiments feature 19 data sets from the UCI Machine Learning Repository. A large collection of UCI data sets have been converted to the Weka ARFF file format. We first selected all two-class data sets from this collection and removed data sets that featured unsupported attribute types, e.g., the *string* data type. The compilation of data sets and their statistics are presented in Table 1. In summary, we have performed a systematic selection of data sets on the basis of compatibility with the studied algorithms. Thus, we have not removed or added any data set based on any other criteria or in favor of any algorithm or metric.

4.3 Experiment 1

4.3.1 Description In this experiment we compare three instances of the AMORI algorithm, each using a different embedded metric. From now on, we use the expression, AMORI(X), to denote a particular AMORI instance that uses X as its embedded metric. For instance, AMORI(ACC) uses classification accuracy as its embedded metric. We wish to investigate if the use of an arbitrary embedded metric would result in a better performance, according to that particular metric, compared to the use of other embedded metrics.

Note that, in this experiment, we are not interested in comparing different representational biases and their possible impact on performance. Although, in the more general case it makes sense that some representational biases are more suitable than others when it comes to optimizing different embedded metrics.

For instance, two back-propagated neural networks, each with a different setup of neurons, both try to optimize the same metric (accuracy), but their performance in doing so could differ greatly.

We divide the experiment into three different cases. In each case we use a different metric for evaluation and calculate the average rank of each AMORI instance according to the performance related to this metric.

Table 1: Data Set Characteristics

Data set	Instances	Attributes		Class1	Class2	#Missing	Missing (%)
		Numeric	Nominal				
backache	180	6	27	155	25	0	0.0%
biomed	209	7	2	75	134	15	0.8%
breast-cancer	286	0	10	201	85	9	0.3%
colic	368	7	16	232	136	1927	22.8%
credit-a	690	6	10	307	383	67	0.6%
credit-g	1000	7	14	700	300	0	0.0%
cylinder-bands	540	18	22	228	312	999	4.6%
diabetes	768	8	1	500	268	0	0.0%
haberman	306	2	2	225	81	0	0.0%
heart-statlog	270	13	1	150	120	0	0.0%
hepatitis	155	6	14	32	123	167	5.4%
ionosphere	351	34	1	126	225	0	0.0%
kr-vs-kp	3196	0	37	1669	1527	0	0.0%
labor	57	8	9	20	37	326	33.6%
liver-disorders	345	6	1	145	200	0	0.0%
mushroom	8124	0	23	4208	3916	2480	1.3%
prnn-synth	250	2	1	125	125	0	0.0%
schizo	340	12	3	177	163	824	16.2%
sick	3772	7	23	3541	231	6064	5.4%

Table 2: Experiment 1 results

Data set	AM(ACC)			AM(AUC)			AM(FME)		
	ACC	AUC	FME	ACC	AUC	FME	ACC	AUC	FME
backache	0.861	0.500	0.000	0.828	0.632	0.367	0.839	0.554	0.216
biomed	0.809	0.745	0.661	0.828	0.789	0.731	0.828	0.786	0.727
breast-cancer	0.731	0.578	0.306	0.720	0.665	0.529	0.720	0.665	0.529
colic	0.859	0.829	0.789	0.859	0.829	0.789	0.859	0.829	0.789
credit-a	0.846	0.849	0.835	0.848	0.853	0.840	0.848	0.853	0.840
credit-g	0.698	0.503	0.032	0.724	0.634	0.471	0.562	0.567	0.443
cylinder-bands	0.672	0.612	0.366	0.654	0.606	0.425	0.583	0.601	0.592
diabetes	0.721	0.636	0.470	0.725	0.689	0.592	0.719	0.700	0.613
haberman	0.745	0.578	0.316	0.722	0.669	0.514	0.719	0.663	0.506
heart-statlog	0.711	0.695	0.629	0.744	0.732	0.685	0.726	0.719	0.681
hepatitis	0.813	0.582	0.293	0.774	0.638	0.426	0.800	0.643	0.436
ionosphere	0.832	0.773	0.706	0.823	0.761	0.687	0.823	0.763	0.690
kr-vs-kp	0.752	0.741	0.657	0.756	0.746	0.668	0.663	0.677	0.738
labor	0.842	0.798	0.743	0.825	0.784	0.722	0.789	0.734	0.647
liver-disorders	0.664	0.616	0.442	0.672	0.633	0.498	0.577	0.587	0.563
mushroom	0.784	0.776	0.711	0.784	0.776	0.711	0.900	0.897	0.886
prnn_synth	0.840	0.840	0.839	0.836	0.836	0.835	0.840	0.840	0.841
schizo	0.559	0.544	0.272	0.582	0.573	0.437	0.488	0.508	0.652
sick	0.974	0.831	0.760	0.958	0.915	0.717	0.973	0.890	0.781
Average	0.774	0.686	0.517	0.772	0.724	0.613	0.750	0.709	0.641

Table 3: Average ranks and Friedman test scores for all the AMORI instances

Algorithm	Metric ACC	 AUC	 FME
AM(ACC)	1.737	2.474	2.605
AM(AUC)	1.921	1.605	1.868
AM(FME)	2.342	1.921	1.526
F_f	1.917	4.311	7.863

4.3.2 Results In Table 2 we present the accuracy (ACC), the F-measure (FME), and area under the ROC curve (AUC) scores for the three included AMORI algorithm instances. Moreover, Table 3 shows the average rank for each algorithm and metric combination (a low rank indicates good performance). The null hypothesis was tested separately for each of the three evaluation metrics using the Friedman test (as described in Section 4). The test statistic, F_f, is distributed according to the F-distribution with 2 and 36 degrees of freedom. At $p < 0.05$ the hypothesis was rejected for the AUC and FME metrics since $F_f > 3.259$. The subsequent Nemenyi post-hoc test revealed that both AMORI(FME) and AMORI(AUC) were significantly better than AMORI(ACC) when evaluated using their respective metrics since the differences in average rank were higher than the critical difference ($CD = 0.760$). AMORI(ACC) did perform better than the two other AMORI instances on the ACC metric but the gain was not significant.

We have now seen that we are able to bias the learning towards a specific goal, but it also important to determine how AMORI performs compared to state-of-the-art rule learners.

4.4 Experiment 2

4.4.1 Description For the second experiment, we compare AMORI with other existing rule induction algorithms, in order to find out if our metric-based algorithm can be competitive against the state-of-the-art. For this purpose, we choose to compare AMORI with two quite different rule learners; Ripper [4] and One Rule [14]. Since we use the Weka implementations of these algorithms we will onwards use their Weka class names, JRip and OneR, when making references.

The rationale for comparing AMORI with these particular algorithms is basically that they represent the state-of-the-art in simple and complex rule inducers, respectively. Complexity-wise, it could be argued that AMORI is located between these two algorithms.

4.4.2 OneR OneR generates rules based on one single attribute. The name of the algorithm is in fact somewhat of a misnomer since it actually generates a set of rules for the selected attribute. OneR ranks attributes according to error rate and generates a classifier using the best attribute. The two major differences between AMORI and OneR are that AMORI can generate rules that operate on more attributes and, when using AMORI, it is possible change the embedded metric, which is the basis for the algorithm to choose one rule over another.

4.4.3 JRip The JRip algorithm is primarily based on the Incremental Reduced Error Pruning (IREP) algorithm. What separates JRip from IREP are three modifications; the pruning metric and stopping criterion have been replaced with more efficient alternatives, and a post-processing technique has been introduced. As stated before, OneR generates a set of rules for one particular attribute and AMORI generates one rule that could include several attributes. JRip, however, can generate a set of AMORI-like rules. In addition, neither AMORI nor OneR performs any post-processing.

4.4.4 Results Table 5 shows the performance per data set and Table 4 shows the average rank of each algorithm for each of the three evaluation metrics. The null hypothesis was again tested separately for each of the three metrics using the Friedman test and for the F-distribution we used 2 and 36 degrees of freedom. The hypothesis was rejected for the ACC and AUC metrics (with $p < 0.05$). We proceeded with the Nemenyi post hoc test, which revealed that JRip was significantly better than OneR for both metrics. This result is not particularly controversial since JRip is often regarded as one of the better rule-based algorithms. However, no other significant differences could be found. The order according to average rank was identical for all metrics: JRip, AMORI, and OneR.

5 Discussion

When analyzing the average ranks in Table 3, it is clear that each AMORI instance performs better than the other two instances if its embedded metric is also used for evaluation. However, this performance gain was only statistically significant at $p < 0.05$ for AMORI(AUC) and AMORI(FME) over AMORI(ACC). That is, AMORI(ACC) did not significantly outperform the other instances on its own metric.

There are several issues that need to be further investigated since they might have influenced the result (in either way). For example, some embedded metrics might work better for a particular class of problems.

Table 4: The Friedman test score for each metric and the average ranks of AMORI, JRip, and OneR

Algorithm	Metric		
	ACC	AUC	FME
AMORI	1.921	1.947	1.895
JRip	1.632	1.474	1.684
OneR	2.447	2.579	2.421
F_f	3.714	7.992	3.029

Thus, if there are a similar number of data sets in favor of each metric this would even out the performance gain for each metric.

Related to the discussion about a particular class of problems, we can observe that several of the studied data sets seem to favor certain embedded metrics, resulting in top scores for all three metrics. For instance, observe the use of ACC as an embedded metric for the labor data set or the use of AUC for the heart-statlog data set.

Finally, the performance gain might be related to the AMORI algorithm itself. More research into this topic is needed in order to fully understand the impact of the representational bias of different algorithms when trying to optimize a particular embedded metric.

6 Conclusions and Future Work

The requirements of many real-world data mining applications can be formalized into application-specific metrics that need to be optimized. Supervised concept learning algorithms are usually designed to maximize a predetermined metric, which we denote the learning metric. Consequently, these algorithms may not generate classifiers that achieve the best possible performance when evaluated using the application-specific metrics.

In order to address this problem, data mining and machine learning researchers have tried various approaches to modify the algorithmic biases of existing learning algorithms so that they optimize application-specific metrics instead of their original learning metric. However, these solutions are often customized for a particular pair of algorithm and metric. That is, they might increase the performance according to the application-specific metrics for a certain real-world application but they are usually not very generic.

As a consequence, we are interested in studying in more general terms if the performance according to an arbitrary application-specific metric can be boosted by incorporating the metric into the algorithmic bias. For this purpose, we have developed A Metric-based One Rule Inducer (AMORI), for which it is possible to select the learning metric in a simple manner.

We demonstrate the applicability of our approach by conducting an empirical comparison of the performance of three instances of AMORI, each using a different embedded metric. The example metrics used in this study were classification accuracy (ACC), the area under the ROC curve (AUC), and the F-measure (FME). The results for AMORI, at least for the studied problems, indicate that the embedded metric exchange yielded a performance gain. The gain was statistically significant for AMORI(FME) and AMORI(AUC) over AMORI(ACC) with $p < 0.05$.

Naturally, there are some aspects that need to be further studied. For example, if there are a similar number of data sets in favor of each metric this would even out the performance gain. More research into this topic is needed in order to fully understand the impact of the representational bias of different algorithms when trying to optimize a particular embedded metric.

We also compared AMORI with a rule inducer of similar complexity, called One Rule (OneR), as well as with Ripper (JRip), which is a state-of-the-art rule set inducer with post-processing capabilities. JRip was not able to significantly outperform AMORI on any of the metrics. JRip was significantly better than OneR for the ACC and AUC metrics. AMORI outperformed OneR for all three metrics but the gain was not significant in any of the cases.

We have identified a number of interesting directions for future work. First off, it might be the case that the metrics included in this study are too similar to be used for the purpose of trying to establish whether or not it is beneficial to use identical embedded and evaluation metrics. Consequently, it could be fruitful to select a more diverse set of metrics for similar experiments. Another related direction would be to compare different representational biases using the same embedded metric, thus making it possible to measure the impact of metric exchange for different combinations of representational biases and embedded metrics.

In addition, more research is needed to understand how to select an appropriate embedded metric based on a particular problem. As we observed in this study, some embedded metrics seem to be more suitable than others for certain data sets. A first step in this direction could be to establish objective metrics for some of the widely used data sets, like those from the UCI machine learning repository. These objective metrics could be used as benchmark evaluation metrics when comparing different algorithms but, more importantly for our purposes, they could be used as embedded metrics.

Up until now we have only discussed some specific

Table 5: Experiment 2 results

Data set	OneR			JRip		
	ACC	AUC	FME	ACC	AUC	FME
backache	0.856	0.497	0.000	0.828	0.495	0.061
biomed	0.861	0.827	0.785	0.885	0.895	0.844
breast-cancer	0.657	0.542	0.310	0.710	0.598	0.428
colic	0.815	0.814	0.764	0.842	0.823	0.775
credit-a	0.855	0.862	0.850	0.858	0.874	0.843
credit-g	0.665	0.532	0.264	0.717	0.593	0.428
cylinder-bands	0.496	0.548	0.596	0.652	0.666	0.559
diabetes	0.730	0.667	0.543	0.760	0.739	0.629
haberman	0.732	0.585	0.349	0.729	0.613	0.443
heart-statlog	0.711	0.706	0.669	0.789	0.781	0.751
hepatitis	0.832	0.652	0.458	0.781	0.664	0.393
ionosphere	0.809	0.785	0.724	0.897	0.900	0.858
kr-vs-kp	0.665	0.657	0.586	0.992	0.995	0.991
labor	0.754	0.684	0.563	0.772	0.779	0.667
liver-disorders	0.548	0.535	0.458	0.646	0.653	0.527
mushroom	0.985	0.985	0.984	1.000	1.000	1.000
prnn_synth	0.832	0.832	0.831	0.844	0.824	0.840
schizo	0.868	0.869	0.867	0.826	0.877	0.818
sick	0.964	0.892	0.733	0.982	0.948	0.860
Average	0.770	0.709	0.597	0.816	0.775	0.669

trade-offs, captured in different ways by established metrics from various areas of research. However, as described in the related work section, quite a few studies advocate the use of more application-specific multi-criteria metrics. It would therefore be interesting to use the multi-criteria functionality of the CEF metric, which is built into AMORI, to be able to trade off several existing metrics during the learning phase.

In addition, one could investigate if it is possible to convert other existing supervised concept learning algorithms into generic metric-based algorithms by incorporating CEF into their algorithmic biases.

Acknowledgments

We would like to thank Fahad Khalid for his ideas about metric-based learning and for valuable discussions in general. In addition, we thank Dr. Nathalie Japkowicz as well as the anonymous reviewers for their insightful comments and suggestions that helped us improve this paper.

References

[1] A. Andersson, P. Davidsson, and J. Lindén, *Measure-based Classifier Performance Evaluation*, Pattern Recognition Letters, 20(11–13), 1999, pp. 1165–1173.

[2] A. Asuncion and D. J. Newman, *UCI Machine Learning Repository*, `http://www.ics.uci.edu/~mlearn/MLRepository.html`, University of California, School of Information and Computer Science, Irvine, CA, 2007.

[3] R. Caruana and A. Niculescu-Mizil, *An Empirical Comparison of Supervised Learning Algorithms*, International Conference on Machine Learning, pages 161–168, ACM Press, NY, 2006.

[4] W. Cohen, *Fast Effective Rule Induction*, International Conference on Machine Learning, pages 115–123, Morgan Kaufmann Publishers, CA, 1995.

[5] C. Cortes and M. Mohri, *AUC Optimization vs. Error Rate Minimization*, Advances in Neural Information Processing Systems, 16 (2004).

[6] S. Dehuri and R. Mall, *Predictive and Comprehensible Rule Discovery Using a Multi-objective Genetic Algorithm*, Knowledge-based Systems, 19 (2006), pp. 413–421.

[7] J. Demzar, *Statistical Comparisons of Classifiers over Multiple Data Sets*, Journal of Machine Learning Research, 7 (2006), pp. 1–30.

[8] B. Efron, *Computers and the Theory of Statistics: Thinking the Unthinkable*, SIAM Review, 21(4), 1979, pp. 460–480.

[9] C. Ferri, P. Flach, and J. Hernandez-Orallo, *Learning Decision Trees using the Area under the ROC Curve*, International Conference on Machine Learning, pages 139–146, Morgan Kaufmann Publishers, CA, 2002.

[10] J. Fürnkranz and G. Widmer, *Incremental Reduced Error Pruning*, International Conference on Machine Learning, pages 70–77, Morgan Kaufmann Publishers, CA, 1994.

[11] C. Giraud-Carrier, R. Vilalta, and P. Brazdil, *Introduction to the Special Issue on Meta-learning*, Machine Learning, 2004, pp. 187–193.

[12] D. F. Gordon and M. DesJardins, *Evaluation and Selection of Biases in Machine Learning*, Machine Learning, 20(1–2), 1995, pp. 5–22.

[13] A. Herschtal and B. Raskutti, *Optimising Area under the ROC Curve using Gradient Descent*, International Conference on Machine Learning, ACM Press, NY, 2004.

[14] R. C. Holte, *Very Simple Classification Rules Perform Well on Most Commonly Used Datasets*, Machine Learning, 11 (1993), pp. 63–91.

[15] J. Huang, C. Ling, *Evaluating Model Selection Abilities of Performance Measures*, AAAI06 Workshop on Evaluation Methods for Machine Learning, AAAI Press, CA, 2006.

[16] J. Huang, C. Ling, H. Zhang, S. Matwin, *Proper Model Selection with Significance Test*, European Conference on Machine Learning, Lecture Notes in Artificial Intelligence, 5211, Springer, Berlin/Heidelberg, Germany, 2008.

[17] R. L. Iman and J. M. Davenport, *Approximations of the Critical Region of the Friedman Statistic*, Communications in Statistics, 1980, pp. 571–595.

[18] T. Joachims, *A Support Vector Method for Multivariate Performance Measures*, International Conference on Machine Learning, pages 377–384, ACM Press, NY, 2005.

[19] N. Lavesson, P. Davidsson, *Quantifying the Impact of Learning Algorithm Parameter Tuning*, AAAI National Conference on Artificial Intelligence, pages 395–400, AAAI Press, CA, 2006.

[20] ———, *Evaluating Learning Algorithms and Classifiers*, International Journal of Intelligent Information & Database Systems, 1(1), 2007, pp. 37–52.

[21] ———, *Generic Methods for Multi-criteria Evaluation*, SIAM International Conference on Data Mining, pages 541–546, SIAM Press, 2008.

[22] K. McGarry, *A Survey of Interestingness Measures for Knowledge Discovery*, Knowledge Engineering Review, 20 (2005), pp. 39–61.

[23] T. M. Mitchell, *The Need for Biases in Learning Generalizations*, Readings in Machine Learning, 1980, pp. 184–191.

[24] D. Musicant, V. Kumar, and A. Ozgur, *Optimizing f-measure using Support Vector Machines*, International Florida Artificial Intelligence Research Society Conference, pages 356–360, AAAI Press, CA, 2003.

[25] F. Provost, T. Fawcett, and R. Kohavi, *The Case Against Accuracy Estimation for Comparing Induction Algorithms*, International Conference on Machine Learning, pages 445–453, Morgan Kaufmann Publishers, CA, 1998.

[26] J. R. Quinlan, *C4.5 Programs for Machine Learning*, Morgan Kaufmann Publishers, CA, 1993.

[27] S. Rosset, *Model Selection via the AUC*, International Conference on Machine Learning, ACM Press, NY, 2004.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*, Parallel Distributed Processing, 1 (1986), pp. 318–362.

[29] D. B. Skalak, A. Niculescu-Mizil, R. Caruana, *Classifier Loss Under Metric Uncertainty*, European Conference on Machine Learning, Lecture Notes in Artificial Intelligence, 4701, Springer, Berlin/Heidelberg, Germany, 2007.

[30] M. Stone, *Cross-validatory Choice and Assessement of Statistical Predictions*, Royal Statistical Society, B, 36 (1974), pp. 111–147.

[31] E. Suzuki and T. Ohno, *Prediction Rule Discovery Based on Dynamic Bias Selection*, Pacific-Asia Conference on Methodologies for Knowledge Discovery and Data Mining, Lecture Notes In Computer Science, 1574, ACM Press, NY, 1999.

[32] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann Publishers, CA, 2005.

[33] S. Wu, P. Flach, C. Ferri, *An Improved Model Selection Heuristic for AUC*, European Conference on Machine Learning, Lecture Notes in Artificial Intelligence, 4701, Springer, Berlin/Heidelberg, Germany, 2007.