Evaluation of different rule learning algorithms

Evaluation verschiedener Implementierungen von Regellernen
Bachelor-Thesis von Christoph Speh aus Bad Sobernheim
Tag der Einreichung:

1. Gutachten: Johannes Fürnkranz
2. Gutachten: Markus Zopf

TECHNISCHE
UNIVERSITÄT
DARMSTADT

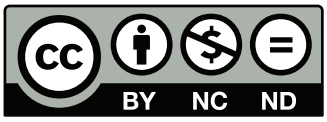
Fachbereich Informatik
Knowledge Engineering

Evaluation of different rule learning algorithms
Evaluation verschiedener Implementierungen von Regellernen

Vorgelegte Bachelor-Thesis von Christoph Speh aus Bad Sobernheim

1. Gutachten: Johannes Fürnkranz
2. Gutachten: Markus Zopf

Tag der Einreichung:



Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den April 16, 2019

(Christoph Speh)

Contents

List of Figures

List of Algorithms

List of Tables

Abstract

Predictive rule learning is a machine learning task which aims to learn a model from training data in order to predict the class of a given, possibly previously unseen, data instance. The learned model is based on an arrangement of **IF** *X* **THEN PREDICT** *Y* rules, which can be used to classify a given instance by assigning the prediction *Y* should all conditions of *X* evaluate to true.

In contrast to many other currently popular statistical machine learning approaches like neural networks, rule based models have the advantage of maintaining high interpretability of the learned model for humans. Therefore rule based models may not only be used to classify, but also to gain valuable insight into the data. This makes rule based algorithms an attractive proposition, especially on smaller datasets where statistical methods will generate little or, in the case of black box approaches, no insight.

Due to the variety of rule learning algorithms, this thesis to evaluates a variety of established and recently developed algorithms in concrete implementations on a larger amount of datasets. The selection of rule learners consist of CN2, JRip, Part, Ant-Miner, CORELS and OFRL. Since proper understanding of a rule learners performance requires some insight into its functionality, this thesis also includes an introduction to the respective algorithms, introduces some of their shared patterns, and gives a brief overview of the implementations that were chosen for this thesis. To get an unbiased as possible comparison, all algorithms receive the exact same training and test sets, which limits the evaluation datasets to those that only contain binary attributes or are converted to such. Multi-class problems are broken up into a series of binary classification tasks, the resulting models of which are combined by a simple meta-classifier to obtain predictions for all instances. Algorithms that have the ability to deal with data that is also symbolic, continuous or multi-class problems are also evaluated on the non preprocessed datasets. This gives a more comprehensive picture of the algorithm's performance and effects of the applied discretization method.

Informed by these result, parameter optimization was used to gain further insight into they behaviour of some of the algorithms on their worst performing datasets. Additionally the environment that was used to run and evaluate the algorithms in a uniform way is also briefly introduced.

1 Introduction

1.1 Motivation

Rule learning is the task of constructing a model based on a set of IF → THEN rules from a given set of training examples, which can then be applied to unseen instances in order to classify them.

$$\textbf{IF } \{manufacturer = BMW\} \text{ AND } \{engine = V8\} \textbf{ THEN PREDICT } ticket$$

Figure 1: An example for a simple rule that predicts speeding tickets.

Due to the easily understandable and interpretable logic of the resulting rules, rule based algorithms have remained popular in many applications that benefit from human insight into the discovered models. Contrastingly many statistical methods that have exploded in popularity exhibit a black box nature that makes it hard to understand the underlying decision making process and are therefore not suited to be used in environments that require a large degree of transparency and accountability. Black box systems that have been used in the criminal justice systems or job recruitment for example have been accused of producing models that are racially biased and not interpretable enough for stakeholders to aid the decision making process or reason about the systems predictions.
The desire to produce transparent models that have comparable predictive power to intransparent models has recently yielded new rule learning algorithms like CORELS [Angelino et al., 2017] and FRL/OFRL [Wang and Rudin, 2015] / [Chen and Rudin, 2018]. In their respective papers, CORELS (**C**ertifiable **O**ptimal **R**ul**E** **L**ist**S**) is introduced as an alternative to the black box COMPAS risk prediction tool that predicts the recidivism risk of criminal offenders, while FRL (**F**alling **R**ule **L**ists) is evaluated in a medical setting, where it performs the grouping of patient into risk groups based on their observed symptoms. Both of these environments require a system of trust and transparency and as such are prime examples for the benefits of human interpretable models.
This thesis aims to evaluate CORELS and OFRL on a wider field of datasets from different domains and to compare them to the well established rule learners CN2, JRip, Ant-Miner, and PART.
Popular machine learning suites like WEKA or Orange offer easy to use and popular implementations of the classic rule learning algorithms which, together with CORELS and OFRL, were integrated into a python environment that evaluates all the algorithms uniformly.
Since not all algorithms can process all kinds of data, this requires the input data and prediction task to be restricted to a shared level of complexity. Most generally one distinguishes between data that is either numerical, symbolic or binary. Binary data is the most limited form of data representation as only two possible values can be taken. Symbolic data can take an arbitrary but finite amount of values and is therefore slightly less restricted. Numeric data is the most complex form of data, as it can take an infinite amount of values. Similarly, the prediction of a binary class label is called binary classification, while the the prediction of a symbolic label is known as multi-class classification. Numeric labels are not part of this thesis. Since CORELS and OFRL only support binary data and classification, the uniform evaluation environment is built to facilitate the transformation of arbitrary datasets into fully binary datasets to enable training for all classifiers, regardless of their ability to to deal with continuous data or multi-class problems. Datasets with multiple labels are broken up into a series of binary classification problems in order to learn rule sets for the individual classes. These individual rule sets are later used in the evaluation as part of a uniform meta classifier that predicts classes for the original multi-class problem. While this approach is objective in the sense that every algorithms has to solve the exact same problem, it does not necessarily reflect the performance on the original dataset should the algorithm employ its own methods to deal with continuous data or multi-class problems. To rectify this, the algorithm is also trained separately on the original dataset and the resulting changes in accuracy and model size are part of the evaluation.

1.2 Organization of this work

This thesis is organized into the following sections:

- **Chapter 2** introduces the fundamental concepts and terminology of rule learning. Rules and the difference between binary, symbolic and continuous data are explained, as well as how to transform attributes into the binary case. This chapter also covers rules list that combine single rules into more complex models.
- **Chapter 3** gives and overview over common concepts of evaluation. This includes the evaluation of single rules and how heuristics guide algorithms during the construction of a rule list. More importantly, this chapter also covers how to evaluate the learned models on separate test sets and what challenges are commonly faced when comparing multiple algorithms. Cross-validation and its influence on bias and variance are introduced, as well

as measures that aim to make the obtained results more consistent. The last sub-section introduces different statistical tests that can aid in assessing whether the observed differences are significant.

- **Chapter 4** highlights two of the most common patterns shared by rule learning algorithms, namely separate-and-conquer and classification by association, as well as main points of differentiation. It then moves on to an introduction of each algorithm that is evaluated as part of this thesis. The six algorithms are: CORELS, FRL, RIPPER, CN2, Ant-Miner and PART. For each algorithm, an overview of its functionality is given with brief sections dedicated to how the search-space is structured and how over-fitting is prevented.
- **Chapter 5** covers how the data was preprocessed, which tools were used and how the results were obtained and aggregated. It explains how the individual binary classification rule sets are combined to perform multi-class classification and why the respective choices were made.
- **Chapter 6** presents the results in terms of achieved accuracy, run-time and model complexity. The results are discussed in each sub-section and statistical test are applied. Additionally a few further experiments were conducted based on observations from the obtained data.
- **Chapter 7** contains the results of parameter optimization on some of the datasets for CORELS, softFRL and FRL. The results are graphically presented in order to judge how robustly these algorithms respond to parameter changes.

2 Overview of the basic concepts and terminology employed in rule learning

In this section concepts and terminology that are common in rule learning and used in this thesis are introduced. Rules and rule lists are formally defined and the transformation of multi-class to binary classification is explained.

2.1 A basic definition of rules

As previously defined, predictive rule learning is the task of building a model on the basis of rules to predict the class of an instance. An *instance*, or in this thesis interchangeably an example, is described by a set number k of *attributes*

$$A = \{A_1, A_2, ..., A_k\}.$$

For each instance, each attribute of the instance has an assigned value v that build the representation of the instance. Attributes can be broken up into the three different groups of *binary*, *symbolic* or *numeric* based on how many different values they can be assigned. Members of the conceptually most basic group, binary attributes, can take one of only two possible values $v \in \{0, 1\}$. Symbolic attributes can take an arbitrary, but finite amount of values, represented by a set of symbols $v \in \{s_1, s_2...s_l\}$. Numeric attributes have the potential to take infinitely many values $v \in \mathbb{R}$ and as such represent the most powerful of attribute types. Sometimes the term continuous is used in rule learning literature instead while numeric values are of the subset $v \in \mathbb{N}$. This thesis does not make that distinction.
Among the attributes used to described an instance there is one attribute of special interest called the *class attribute*. The value assigned to the class attribute is commonly described as the class or the label of the instance. In the case of a binary class attribute this thesis uses the term *binary classification* for predicting the class label of an instance, and *multi-class classification* should the respective class attribute be symbolic.
While a variety of rule representations exist [Fürnkranz, 1999], all algorithms in this thesis use *selectors*. A selector is a condition of the form $A_i \# c$. $\#$ is a relation like $=, \leq, >$ etc. and c is a valid value of A_i. If none of the attributes are numeric, the set of relations is reduced to only $\{=\}$. In this thesis, the term *antecedent* denotes a conjunction of some valid selectors. Growing an antecedent a, or making it more specific, adds a new condition *cond* to a and is denoted by $a = (a, cond)$. Often the terms rule and antecedents can be used interchangeably under the assumption that an antecedent can easily be transformed to a rule by assigning the best class label as prediction.

$$a = \{A_i \# c_i \wedge ... \wedge A_j \# c_j\}$$

Let an instance be described by a set of attributes $A = \{A_1, ..., A_k\}$ and v_i be some valid value of A_i. Then an instance is defined by the tuple

$$(v_1, v_2, ..., v_k).$$

Let A_k be the class attribute. A rule **IF** X **THEN** y consists of the antecedent X and the prediction y, where y is some valid value of a_k. A rule can be used for classification of an instance, if $A_i \# v_j$ evaluates to true for all selectors of the antecedent. The antecedent evaluating to true for an instance is usually referred to as the antecedent or rule capturing or covering the instance.
With this definition of an instance, we can also describe the transformation of a numeric or symbolic attribute to binary data. The conversion for symbolic attribute A_i can be achieved trivially by replacing it with a series of binary attributes: For every possible value $s_i \in \{s_1, s_2...s_l\}$ of symbolic attribute A_i, we introduce a new attribute A_{si} with values $v_{si} \in \{0, 1\}$. Every instance formerly described by the tuple $(v_1, ..., s_i, ..., v_k)$ is now represented by the tuple $(v_1, ..., v_{s1}, ..., v_{sl}, ..., v_k)$ with $v_{si} = 1$ and $v_{si'} = 0$.
The conversion of numeric attributes inherently leads to a loss of information since we are limited to a finite set of attributes. A numeric attribute can be transformed into a symbolic attribute by creating a number of discrete bins, each capturing a certain range of the numeric attribute. Each bin corresponds to a symbol. Different ways of determining the number and sizes of bins exist, some unsupervised like equal width bins and some more complex involving own supervised machine learning methods.

2.2 Rule lists

For many rule learning task, a single rule is simply not versatile enough to represent an accurate model of the data. Multi-class classification is one such problem since a model needs multiple rules to predict the different class labels. One challenge of naively applying rule sets is that a single instance might be captured by multiple rules making differing predictions, known as the overlap problem [Fürnkranz, 1999]. A common approach to specifying an order in which to apply a rule set is a *decision list* or *rule list*. Formally we will define a rule list $d$of length $|d| = k$ as a list of k antecedent label tuples $d = ((a_0, y_0), (a_1, y_1), ..., (a_k, y_k))$. Growing a rule list d simply appends a new rule $r = (a_r, y_r)$ to the end of the list and is denoted by $d = (d, r)$. To classify an instance, it gets tested against the antecedent of every rule along the list until it is classified by the first rule that captures the instance. Usually a *default rule* is included as (a_k, y_k) with $a_k = \{true\}$ in order to capture any remaining instances.

IF age=18-20 AND sex=male **THEN PREDICT** *yes*
ELSE IF age=21-23 AND priors=2-3 **THEN PREDICT** *yes*
ELSE IF priors>3 **THEN PREDICT** *yes*
ELSE PREDICT *no*

Figure 2: A rule list constructed by CORELS [Angelino et al., 2017].

For later use, we will introduce some terminology in regards rule lists used by [Chen and Rudin, 2018]. A *prefix* e of a rule list d is the rule list d without the final **ELSE** clause, $e = ((a_1, y_1), ..., (a_k - 1, y_k - 1))$. This can easily expanded to the k-prefix which excludes the last k clauses.

3 An overview over evaluation of rules and rule sets

In this section an overview of evaluation in rule learning is provided. The aim of an evaluation is to judge the quality of a constructed model for a given task. The first sub-section introduces some measures which are commonly used in algorithms during model construction to judge the quality of a generated rule, while the second sub-section concerns the evaluation of finished models and their suitability for classification of previously unseen instances. The last sub-section regards statistical test to determine how algorithms perform relative to one another and whether those differences are statistically significant.

3.1 Evaluation as part of model construction

During construction of a model, it is often required to have a measure of quality for single model components to decide which parts are to be refined, pruned or kept. Mainly due to limitations in computational complexity its is often unfeasible to certify the optimality for a given objective. Sometimes it might even be desired to select a model that has sub-optimal performance, but performs well among metrics that are intuitively understood or practically shown to produce good results.
Such a metric is known as a *heuristic*. A commonly used heuristic for antecedent construction is *entropy* and the related *Information gain* or some variation thereof. Entropy for a set of instances where n class labels appear with probability $p(c_i)$ is defined as $-\sum_{i=1}^{N} p(c) \log p(c))$ and measures the purity of an instance set in regards to the class label. High entropy implies a low degree of purity/information content decreasing the entropy results in an information gain. Increasing purity among instances captured by a condition suggest a high relevance of the condition for the discrimination of the class label, which makes it an intuitive heuristic for greedy antecedent construction. Different variations of heuristics based on information gain have been proposed in order to deal with scenarios where pure entropy does not produce the intended result, i.e. attributes with a unique value for every instance.

3.2 Evaluation of a model on unseen data

After a model has been constructed, it needs to be evaluated on actual data. To get a representative result, one typically splits the used dataset into two sub sets, a large *training set* and a smaller *test set*. First, a model for the class label is learned on the training set which is then successively given instances of the test set on which to predict the class labels. In the case of a binary classification with a positive and negative class label, the term true positive (tp) and true negative (tn) is used for a correct prediction of the respective label by the model. A misclassification of an instance with the negative class label as positive and vice versa is a false positive/negative. The number of true/false positives/negatives and their rates for a given dataset can be presented as a *confusion matrix*, which can also be extended for multi class classification problems.

Many common performance metrics can be computed from a given confusion matrix. The following measures are relevant for this thesis:

- **Precision:** The fraction of true positives among all instances predicted as positive. $Pr = \frac{\#tp}{\#tp+\#fp}$

Predicted Class \ Actual Class	True	False
True	**#tp**	#fp
False	#fn	**#tn**

Predicted Class \ Actual Class	a	b	c	d
a	**4**	0	1	0
b	0	**3**	0	1
c	2	0	**7**	0
d	0	4	0	**10**

Figure 3: Schema for a binary confusion matrices and a concrete multi-class confusion matrix

- **Accuracy:** The fraction of correct classifications. In terms of the confusion matrix, accuracy can be computed as $Ac = \frac{\sum \text{diagonal entries}}{\sum \text{total entries}}$ and is therefore easily extended to multi-class problems.

As the goal of this thesis is the evaluation of algorithms and not singular models, we need to perform more than one model evaluation per dataset. To reduce the potential effects of an unfavourable train-test split, *n-fold cross-validation* is commonly used. The dataset gets split into n equally sized folds, with each fold serving as test set once, while all other folds combined are used as training set. Averaging the results from n evaluations will give more reliable results, while the spread between individual folds can be used as an indicator for robustness of the generated model.
Bias and *variance* are two of the main culprits when measuring predictive power. To generate appropriate models, every algorithm employs certain biases, that prefer some kinds of rules over others, even if it goes against consistency on the observed training instances. I.e. a rule with only 2 conditions might be preferred to a rule with 5, because a more general model is expected to perform better on yet unseen examples. Variance on the other hand measures how sensitive a classifier is to small changes in the data. The term bias–variance trade-off is used to describe the property that an algorithm with a high bias will exhibit a lower degree of variance and vice-versa. It is therefore difficult to minimize both of these properties at the same time and an appropriate decision regarding this trade-off needs to be made for every application. [Santafé et al., 2015] notes, that an evaluation process comparing multiple algorithms with each other should focus efforts on reducing variance. Proposed measures include stratification of the training data (sampling from each class independently) and the preference of repeated cross-validation with larger folds over a single small fold one. [Dietterich, 1998] determined that a 5 times 2 fold cross-validation leads to a more accurate estimation of the variance and mean because it reduces correlation between the training sets and might be preferable for test that rely on these statistics.

3.3 Statistical testing

Once the results for the individual data sets are available, one common method of ranking the algorithms is to compare the performance of all algorithms on each individual dataset and assign a rank in decreasing order based on their relative performance. This ranking is then used in conjunction with statistical tests to determine whether there is a significant difference between the algorithms. [Demsar, 2006] and [Santafé et al., 2015] both stress the point that statistical tests are often misunderstood, misused or manipulated, but at the same time offer at least some reassurances that, at the same time, should not be overstressed.
The base assumption that all algorithms perform the same is the null hypothesis H_0, while the alternative hypothesis H_1 states that one of the algorithms is either better or worse. The significance level p denotes the probability that H_0 is rejected in favour of H_1 even though H_0 is the correct hypothesis. A value of p below some threshold is therefore a measure for how confidently the null hypothesis can be rejected. This rejection of H_0 is performed through an omnibus test, which is followed up by a set of pairwise post-hoc tests to determine the p-levels between individual algorithms. One challenge is the family-wise error that one of the many pair-wise comparison wrongfully rejects the null hypotheses that both algorithms perform equally. [Demsar, 2006] proposes the Friedman test with the Iman Davenport correction as a the best omnibus test, as it is less conservative than the older Friedman chi-squared test. One possible post-hoc test is the Nemenyi test that calculates a critical difference by which the average ranks need to be separated in order to declare it significant with some p-level. The critical distance can be presented graphically by placing the average rank of each algorithm on an axis and connecting them if they their distance does not exceed the threshold. As a second post-hoc test, this thesis includes the Conover-Friedman test with the p-adjustment by Hommel to control for family-wise error. While this test does not offer an intuitive graphical representation like Nemenyi, it should be more powerful and less conservative than the Nemenyi test.

4 The algorithms

In this section some of patterns shared among many rule learning algorithms, as well as points of differentiation are introduced. Additional sub-sections summarize how the individual algorithms that are part of this thesis function, how they differ, and what kind of parameters they expose to the user. A few odds and ends regarding the implementations are included as well.

4.1 Common shared patterns in rule learning

[Fürnkranz and Kliegr, 2015] identify two common rule learning strategies shared among many algorithms. The first is the *Covering* or *seperate-and-conquer* algorithm. A covering algorithm repeatedly learns a single rule in every iteration. After a rule has been added, all instances captured by that rule are removed from the considered dataset and the next rule is learned. The algorithm stops after some kind of stopping criterium is reached or all instances are covered.

Algorithm 1 Covering algorithm taken from [Fürnkranz and Kliegr, 2015]

Input: *Examples*, a set of positive and negative examples for a class c.

```
procedure COVERING(Examples, Classifier)
    R = ∅                                              ▷ initialize rule set
    while not all positive examples are covered do
        r = FINDPREDICTIVERULE(Examples)               ▷ find the best rule for the current examples
        if R ∪ r is good enough then                   ▷ check if we need more rules
            break while
        end if
        Examples = Examples \ {examples covered by r}  ▷ remove covered examples
        R = R ∪ r                                      ▷ add rule to list
    end while
end procedure
```

Output: the learned rule set R

The second common pattern is *classification by association*. This approach is based on association rule learning, a machine learning method that aims to discover interesting relations of the kind BODY → HEAD between attributes.

BRUSH, CANVAS → PAINT, SCRAPPER, THINNER

The above rule could for example specify, that people that buy a brush and a canvas are also likely to buy paint, a scrapper and paint thinner. Two often used metrics to asses the importance of association rules are *support* and *confidence*. The support of a rule $supp(\text{BODY} \cup \text{HEAD})$ specifies which fraction of the considered dataset satisfies the body and head of a association rule. Confidence is defined as $supp(\text{BODY} \cup \text{HEAD})/supp(\text{BODY})$, with higher confidence suggesting a larger correlation between the body and head of a rule.
Classification by association algorithms use a pre-mined list of all association rules with the class as part of the head as their basis for the search space. In each iteration a heuristic is used to determine the best rule, which is then added to the rule set.

Algorithm 2 Classification by association algorithm as described by [Fürnkranz and Kliegr, 2015]

Input: *Examples*, a set of positive and negative examples for a class c; *Heuristic*, a function to asses the quality of an association rule for classification.

```
procedure INDUCEFROMASSOCIATIONRULES(Examples, Classifier, Heuristic)
    A = GENERATEBESTASSOCIATIONRULES(Examples)
    A = {a ∈ A | c ∈ HEAD of a}
    R = ∅                                              ▷ initialize rule set
    while R not good enough do
        r = FINDBESTRULE(A, Heuristic)                 ▷ find the best rule according to the heuristic function
        A = A \ r                                      ▷ remove already used rule from A
        R = R ∪ r                                      ▷ add rule to list
    end while
end procedure
```

Output: the learned rule set R

4.2 Main points of differentiation between rule learning algorithms

[Fürnkranz, 1999] suggests three main axis of bias on which to differentiate algorithms. The three biases are: **Language Bias, Search Bias, and Over-fitting Avoidance Bias**.
Language Bias is the result of different representations forms algorithms use for their models. While one concept might be expressed elegantly in one model language, it may be completely impossible or highly complicated to represent in another. The chosen form of language can therefore be an important source of bias.
The Search Bias decides how the search space of the algorithm is explored. Since the space of possible models is usually way to large for exhaustive search, it is the task of the employed search bias to guide the search along avenues that promise the highest quality results.
Over-fitting Avoidance Bias characterizes the way in which the algorithms tries to select models that will generalize well to yet unseen data. Due to noise, incompleteness or other factors it is often impossible to learn a complete and consistent model that will correctly classify every possible instance. Therefore it is often desirable to use heuristics which generate models that do not fit the training data to tightly.

In the introduction sections for the algorithms that are part of the evaluation, special attention will be paid to how they differ in their biases and what parameters they expose to the user to adjust them. Because all of the considered algorithms use selector based rules and very similar ways to arrange them, Language bias is not applicable, so the organisation of the search-space and over-fitting avoidance will be the primary focus.

4.3 CORELS

CORESL (Certifiable Optimal RulE ListS) [Angelino et al., 2017] is a supervised algorithm for finding an optimal rule list that minimizes the objective function $R(d, x, y) = \text{misc}(d, x, y) + c \cdot \text{length}(d)$ where d is the rule list, x the training set, y the provided class labels, $\text{misc}(d, x, y)$ the fraction of misclassified examples, and c a user defined regularization parameter that penalizes larger rules. All features as well as the classification need to be binary.
The search space is organized in form of a prefix tree of rule lists where the first level $j = 0$ of the tree represents rule lists that contains a single rule consisting of an antecedent a_0 and the binary prediction y_0 that minimizes the fraction of misclassified examples in $n_{j,d,x}$, the second level rule lists with two rules and so on. An initial rule list can also be supplied instead of the empty rule list. Subtrees that have a larger lower bound for the objective than the current minimum are not further explored since all further descending subtrees are guaranteed to have a larger objective as well, while trees that offer a smaller objective are placed in a priority queue that manages which rule list will be explored next. CORELS offers a variety of curiosity metrics like Breadth-First Search or Depth-First Search to determine the order in which the rule lists are explored, as well as the ability to define custom metrics. The additional pruning of subtrees with antecedents that capture to many or to few training instances according to a user specified constant is also supported, as well as of subtrees that represent rule lists that are symmetric to already seen rule lists. The search terminates after either the entire tree has been searched, or a user set number of tree nodes has been visited.

Search bias: CORELS searches from the most general rule set to more specific ones. Since CORELS does not prune subtrees in a greedy way, meaning it explores trees that might still yield a lower objective even if they would be deemed to have a low probability to do so by a some heuristic, the search space has the potential to be very wide. The user has a great amount of influence on how to explore that search space through the ability to choose the curiosity metric. An optimal curiosity metric would immediately find the best rule list and then quickly certify its optimality by pruning all remaining subtrees [Angelino et al., 2017]. Should the curiosity metric only find a distinctively good rule list very late in the process, it could heavily limit the amount of pruning that can be performed.
Over-fitting avoidance bias: CORELS does not expose many direct ways to influence over-fitting. The primary parameter is the length penality that instructs CORELS to prefer shorter and therefore more general rule lists. The earlier mentioned flexibility in shaping the search space also gives the ability to preferably explore rule lists that might lead to less over-fitting i.e a breadth-first curiosity metric that tests all shorter rule lists before exploring longer ones. Since CORELS does not perform greedy pruning, there are not many options to favour one antecedent over another beyond the minimum support parameter that dynamically excludes antecedents that are to specific or general. Another possible option is to supply a different set of possible antecedents and exclude antecedents that one expects to not generalize well, which would require changes outside of CORELS.
Implementation specifics: CORELS mainly operates on a list of antecedents as opposed to the actual training examples, which is why the input consist of a a list of pre-mined rule antecedents, followed by a bit-vector of length l = number of examples that describes which training examples get captured by each antecedent. Any method of generating suitable antecedents can be used, but just as the original CORELS paper, this thesis decided to go with the simplest method of enumerating all possible antecedents up to a set length. Earlier versions of CORELS required this set of antecedent bit-

Algorithm 3 CORELS [Angelino et al., 2017]

Input: a set An of antecedents, training data (x, y), initial best rule list d^0.

procedure BRANCH-AND-BOUND(S, (x, y),d^0)
  $R^0 = R(d^0, x, y)$ ▷ objective of initial rule list
  $(d^c, R^c) = (d^0, R^0)$ ▷ initialize best rule and objective
  $Q =$ queue([()]) ▷ initialize queue with empty prefix
  while Q not empty **do**
    $e_p = Q$.pop() ▷ Remove first prefix from queue
    $d =$ SETBESTLABELPREDICTIONS(e_p) ▷ sets the labels that minimize training error for all rules
    if $b(e_p, x, y) < R^c$ **then** ▷ check if e_p has a lower bound for error then current best
      if $R(d, x, y) < R^c$ **then**
        $(d^c, R^c) = (d, R(d, x, y))$ ▷ update if new best rule found
      end if
      for a in An **do**
        if a not part of e_p **then**
          Q.push(e_p, a) ▷ add all children of e_p to the queue
        end if
      end for
    end if
  end while
end procedure

Output: the optimal rule list d^c with objective R^c

vector pairs to be written to a file that was then passed to the C implementation of CORELS. Due to the need to transform numerical values into binary features, some of the data sets that were used in the evaluation part of this thesis feature of over 100 attributes. The number of all possible antecedents on n different attributes is the cardinality of the power-set $|\mathscr{P}(An)| = 2^{|An|}$, which means the training file consists of an exponential number of bit vectors, each with length = number of training instances. This leads to a very fast blow up in file size, and typically limits the mined antecedents to length two or three. A training file with 120 attributes and a maximum antecedent length 3 can already reach a size of close to 1 GB. It is also worth noting that a large antecedent count can result in large RAM utilization, which can be combated by disabling some the acceleration structures that are utilized per default. Later versions of CORELS offer a more advanced Python binding that integrates CORELS directly through the Python C-extension Cython and does not require the data to be written to a separate file. Instead, a standard CSV file can be read with a convenience function.

4.4 OFRL

Optimized FRL (Falling Rules list)[Chen and Rudin, 2018] is a supervised rule learning algorithm for binary classification that produces an ordered rule list where each rule's probability of predicting the positive class label is monotonically decreasing along the list. If the probability of the positive class label Y given features X, denoted as $P(Y \mid X)$, is interpreted as a risk, the rule list naturally groups the examples into a descending order of risk-tiers.

	antecedent	$P(Y \mid X)$
IF	poutcome=success AND default=no	0.65
ELSEIF	60 ≤age≤100 AND default=no	0.28
ELSEIF	17 ≤age≤ 30 AND housing=no	0.25
ELSEIF	previous≥ 2 AND housing=no	0.23
ELSEIF	campaign= 1 AND housing=no	0.14
ELSEIF	previous≥ 22 AND education=tertiary	0.13
ELSE		0.07

Figure 4: Example of a falling rules list with probabilities. Table data from [Chen and Rudin, 2018]

Based on an earlier, also FRL called algorithm based on Bayesian modelling [Wang and Rudin, 2015], Optimized FRL is an optimization based approach that aims to decrease computation time through tighter algorithmic bounds while producing similar rule lists to standard FRL.
OFRL seeks to find the rule list d which minimizes the risk of misclassification $r(d, (x, y), w) = \frac{1}{n}\left(w \cdot \#f_n + \#f_p\right)$ on the positive class label on the test data, with the number of misclassified positive examples $\#f_n$ receiving a user defined

weight w. $\#f_p$ is the number of false positives. Additionally a penalty for the length of the rule list $C \cdot |d|$ is added, which gives the objective function $R(d,(x,y),w,C) = r(d,(x,y),w) + C \cdot |d|$.
To limit the search space, OFRL works on the assumption that accurate rule lists can be learned from a set of pre-mined association rules generated by the FPgrowth algorithm [Han et al., 2000], with the maximum length and minimum support of the mined rules being specified by the user.
In every iteration OFRL constructs a new rule list by successively appending new antecedents to a prefix rule list e, that do not violate the monotonicity constraint. Construction is stopped early if the calculated objective bound guarantees no possible improvement by further appending to the prefix. The curiosity function weighs between the fraction of positive examples in the newly captured instances and the support of the antecedent among all positive training instances uncaptured by e through a user defined parameter λ. The assigned probability $P(a \in S, e, D)$ is the value of the curiosity function normalized over all antecedents.
An iteration terminates with a set probability, or when no more eligible antecedents are available. The generated rule list is then evaluated with the objective function and compared to the previously best seen rule list.

Algorithm 4 OFRL [Chen and Rudin, 2018]

Input: a set An of antecedents, training data (x, y).

```
R^c = inf                                   ▷ initial rule list and objective
d^c =
for 1, ..., T do
    d = e = {}
    while better rule with d as prefix e could exist do
        GoTo terminate: with probabilty p_terminate
        S = FilterAntecedents(An)           ▷ only consider antecedents that result in a falling rule list
        if ( thenS ≠ ∅)
            pick a_p ∈ S with probability P(a_p, e, (x, y))
            d = (e, a_p)                    ▷ all antecedents predict the positive label
            d = e
        else
            GoTo terminate:
        end if
    end while
    terminate:
    if R(d,(x,y),w,C) < R^c then            ▷ update best rule list and objective d^c = d R^c = R(d,(x,y),w,C)
    end if
end for
```

Output: the best rule list d^c with objective R^c

Search bias:OFRL exhibits a stochastic search pattern through random early termination of rule list construction and random antecedent selection. This keeps the potential search space wide, while still pruning bad rule list early through the bound on the prefix. With the inclusion of a weight w for the positive instances, OFRL can be made especially sensitive to data that has a low fraction of positive examples or that is cost sensitive. [Chen and Rudin, 2018] notes that the optimal threshold for a rules possible inclusion is $P(Y \mid X) > 1/(1+w)$ and therefore higher weights will introduce rules with lower probability estimates, which might be desirable if the cost of a false negative is high. One should note the difference between the optimized objective and $P(Y \mid X)$, which while related, are not the same.
Over-fitting Avoidance Bias: Besides the regularization parameter C that penalizes rule list length, tuning λ is the main way to combat over-fitting. λ close to 1 will prefer antecedents with a tighter fit, while a smaller λ will prefer antecedents that cover more examples. Like CORELS, OFRL operates on pre-mined antecedents which can be restricted to exclude antecedents that might cover to few examples.
Implementation specifics: OFRL is implemented entirely in Python 2.7 and can be supplied with standard CSV files. The usage of FFPgrowth as association rule miner limits the current implementation to binary datasets, but in principle any association rule learning algorithm can be substituted. The easier to use implementation, shorter training times and easier to understand algorithm structure were the main factors for choosing OFLR over FRL and subsequently both terms will be used interchangeably. Additionally the implementation includes the variant softFRL that relaxes the monotonicity constraint on the rule list and antecedent selection. Instead of a strict enforcement, a penality term controlled through a parameter C_1 is added to the objective function.

4.5 RIPPER

JRip is the Weka implementation of the separate-and-conquer algorithm RIPPER(Repeated Incremental Pruning to Produce Error Reduction) proposed by [Cohen, 1995]. RIPPER builds a rule list for a class in two stages. The building stage generates a first list of rules which then enters an optimization stage where alternative rules are generated and worse performing rules are discarded. The building stage itself consists of two main stages, the methods grow and prune, that are repeated until the rule list either covers all positive instances, an error rate of 50% is exceeded or some maximum description length of the rule list is reached.
The grow method greedily builds a completely accurate rule on a subset of the instances by repeatedly evaluating every possible value of every attribute according to an information gain heuristic and appending the best condition to the antecedent of the rule. The method prune then uses a separate prune set to delete any final set of conditions until the pruning heuristic does no longer improve. The optimization stage compares each rule created in the building phase with a replacement rule as well as a revisioned rule and chooses the rule that minimizes the error of the entire rule list. The revision grows and prunes an entirely new rule much like the building stage, while the refinement does the same with the original rule as starting point. Should any uncovered positive instances remain, additional rules are grown that do not increase the description length.
Multi-class problems are treated with a similar separate-and-conquer approach, by building rule lists in ascending order of class prevalence and removing instances that are already covered from the dataset.

Algorithm 5 RIPPER [Cohen, 1995]

Input: training data (x, y)

Building stage
$d = \{\}$ ▷ initialize as empty
while d description length not to big, or error rate <= 50%, or $(x, y) \neq \emptyset$ **do**
gSet, pSet = SPLIT((x, y)) ▷ reserve some examples for growing and for pruning
procedure GROWANDPRUNE(gSet, pSet, {})
r = GROW(gSet, {}) ▷ grow new fully accurate rule
r = PRUNE(r, pSet) ▷ prune until pruning does no longer improve heuristic
return r
end procedure
$d = (d, r)$ ▷ add new rule
$(x, y) = (x, y) \setminus$ {instances covered by r}
end while

Optimization stage
for k optimization steps **do**
gSet, pSet = SPLIT((x, y))
for each rule r_i in d **do**
r_{rep} = GROWANDPRUNE(gSet, pSet, {}) ▷ grow replacement and refinement rule
r_{ref} = GROWANDPRUNE(gSet, pSet, r_i) ▷ pruning in this stage for error of entire d
choose best of $r_i, r_{\text{rep}}, r_{\text{ref}}$ for d
end for
while description length does not grow **do**
r =GROWANDPRUNE(gSet, pSet {})
$d = (d, r)$
end while
end for

Output: a rule list d

Search bias: RIPPER is a prime example of a greedy separate-and-conquer algorithm that follows the path of the local optimum to grow an increasingly specific rule. To somewhat reduce the risk of getting trapped in a local optimum, RIPPER does employ a random component in the optimization stage in form of the two additional rule constructions that use different random grow and prune sets. RIPPER does not expose any parameters that could allow the user to contribute domain specific priors during the construction of the model. Due to its greedy nature, it is expected that RIPPER will scale quite well to larger datasets, since the search space will always remain very limited.
Over-fitting Avoidance Bias: RIPPER intentionally over-fits rules in the grow stage. The over-fitting during rule construction can be relaxed through a parameter that specifies a minimum level of support that every rule needs to reach. Immediately after, greedy pruning is applied, with further pruning to optimize the performance of the entire rule set in

the optimization stage. The resulting rule set can be optimized repeatedly. While the use of independent grow and prune sets does reduce the amount of data that is available for training, it should result in good over-fitting characteristics. The size of the pruning set can be adjusted, which gives the ability to react to i.e. the expected amount of noise in the data, but might require additional optimization iterations to reach the necessary amount of rule refinement. Should a tightly fitted model be desired, it is possible to disable the pruning stages entirely and use all data for training.
Implementation specifics: Ripper can deal with numeric attributes as well as multi-class problems and has few tuneable parameters. Together with its integration in Weka and portable Java implementation it requires little effort to setup and run on any dataset. It can easily be integrated into a Python environment or can be used through the Weka provided GUI.

4.6 CN2

CN2 is the oldest algorithms in this evaluation, initially proposed by [Clark and Niblett, 1989] and further improved by [Clark and Boswell, 1991]. It learns rules in a separate-and-conquer stile until no more 'good' rules can be found. The last rule is always the default rule that predicts the majority class. Rule construction consists of a step by step specialization of the most general rule with the antecedent 'true'. Instead of only considering one potential rule at a time, a beam search that keeps the k best antecedents is performed. A antecedents is considered as new best antecedents only if it passes a significance test to exclude to specific potential rules. [Clark and Niblett, 1989] proposes entropy as measure of rule quality and likelihood ratio as significance test. [Clark and Boswell, 1991] suggest the improvement of using Laplace accuracy $\frac{tp+1}{tp+fp+k}$, with k being the number of classes, as heuristic instead, which will prefer more general rules. Because the rules are already more general, the significance test will only result in earlier stopping of rule construction. Additionally the ability to build unordered rule sets as opposed to rule list was proposed. Long rule list can result in poor interpretability of later rules since the effects of all earlier rules must be considered as well. Rule sets resolve this issue by considering each rule independently during classification. Potentially conflicts resulting of different predictions by overlapping rules are handled through a class distribution of all covered examples attached to each rule. The distributions of all firing rules are added together, and the most common class is predicted.

Search bias: CN2 is a refinement of a separate-and-conquer general to specific rule learner. It offers a clear trade-off between a wide search space through an increased beam-width and a deeper search through a decreased confidence value.
Over-fitting Avoidance Bias: CN2 gives very little control over how it handles over-fitting beyond the simple stopping criterium.
Implementation specifics: CN2 is implemented as part of the machine learning suite Orange. It can be used through a supplied GUI or a Python 2.7 module and offers easy interoperability with Weka through shared usage of arff files, while providing another set of preprocessing and visualization as part of the Orange toolset. It can deal with numeric attributes as well as multi-class problems and profits from its intuitive structure and easy to comprehend parameters.

Algorithm 6 orderedCN2[Clark and Boswell, 1991]

Input: training data (x, y), beam width k, significance level c

$d = \{\}$ ▷ initialize rule set
while no more rules are found **do**
  $A = \{\{true\}\}$ ▷ start with the most general rule in candidate set
  best = null ▷ initialize that there currently is no best rule
  while $A \neq \emptyset$ **do** ▷ repeat until no more candidates remain
    $A_{new} =$
    for each $a \in A$ **do**
      for each condition cond not already used in a **do**
        $a_{spec} = (a, \text{cond})$ ▷ making a more specific by adding a condition
        if a_{spec} is better than best and is significant with level c **then**
          best $= a_{spec}$
        end if
        $A_{new} = A_{new} \cup a_{spec}$ ▷ add more specialized rule candidate to candidate set
        if $|A_{new}| > k$ **then**
          $A_{new} = A_{new} \setminus$ worst $a \in A_{new}$ ▷ only keep the k best rules
        end if
        $A = A_{new}$
      end for
    end for
  end while
  $a =$ best
  if $a =$ null **then**break while ▷ check if we need more rules
  end if
  $c =$ most common class in instances covered by a
  $r =$ **IF** a **THEN** c
  $d = (d, r)$
  $(x, y) = (x, y) \setminus$ {instances covered by r}
end while$c =$ most common class
$r =$ **IF** true **THEN** c

Output: the learned rule list d

4.7 Ant-Miner

Ant-Miner is an ant-colony optimization based separate-and-conquer rule learner for symbolic data. The algorithm is inspired by the behaviour of real ants finding the shortest path between destinations through pheromone trails. Shorter paths result in a higher utilization, which leads to more pheromone accumulation and in turn incentives other ants to choose the same path. Eventually non optimal pheromone trails will evaporate sufficiently and the ants will converge to a single path. To adapt ant-colony optimization for rule learning, each rule consisting of n conditions is interpreted as a path length n that can be taken by an ant with a probability dependent of the pheromone count and a heuristic function based on entropy. Initially all paths (rules) have the same amount of pheromone associated with them and a rule is constructed by adding conditions until some minimum of instances is covered or all attributes have been used once. The rule is then pruned in a post-processing step that deletes conditions until the rule quality $Q = \frac{tp}{tp+fn} \cdot \frac{tn}{fp+tn}$ no longer improves. This quality measure is then also used to update the pheromone trails by adding to the value of each used path and reducing the pheromone count of all others. Additional ants are sent to explore paths until either a maximum of ants is reached or a set number of consecutive ants took the same path, indicating convergence. The best constructed rule is chosen and added to the rule list. All covered instances are removed from the training data and the next rule is learned. This process is repeated until a user-defined threshold of maximum uncovered instances is reached.

Algorithm 7 Ant-Miner [Parpinelli et al., 2002]

Input: training data (x, y), a maximum number of ants n_{ants}, a threshold for convergence n_c, a maximum number of uncovered instances $n_{\text{max\_i}}$

```
d = {}                                                  ▷ initialize as empty
while |(x, y)| > n_max_i do
    P = {p_0, p_1...}. = c                              ▷ initialize all trails with same value R = {}
    for i = 1..n_ants do
        r_i = GROW((x, y), P, {})                       ▷ grow new rule until minimum of instances covered
        r_i = PRUNE(r)                                  ▷ prune until pruning does no longer improve heuristic Q
        UPDATEPHEROMONE(P,r_i)                          ▷ increase pheromone of p_i associated with r_i
        NORMALIZEPHEROMONE(P)                           ▷ normalize, effectively decreasing p of all other trails
        R = R ∪ r_i
        if r_i same as previous n_c rules then
            GoTo terminate:
        end if
    end for
    terminate:
    r = MAXQ(R)
    d = (d, r)                                          ▷ add best rule according to heuristic Q
    (x, y) = (x, y)\ {instances covered by r}
end while
```

Output: a rule list d

Search Bias: Ant-Miner mainly diverges from other separate-and-conquer algorithms in the rule construction phase. Through initializing every path with the same amount of pheromones, Ant-Miner performs a less greedy search and explores a wider search-space. This will lead to construction of bad rules by early ants, but is very robust against local optima in the attributes. The repeated rule construction and the following evaluation has the potential to be very costly on datasets with a huge number of attributes.

Over-fitting Avoidance Bias: Just like RIPPER, Ant-Miner performs post-processing of rules to reduce over-fitting, though it does not use a separate pruning set and does not perform additional global optimization through pruning the completed rule list. [Parpinelli et al., 2002] also notes, that while the influence of pruning on classification performance is limited, it does dramatically improve the size and interpretability of the produced rule set. As an additional measure a threshold for a minimum number of covered examples per rule can specified.

Implementation specifics: This thesis uses an implementation of Ant-Miner by [Otero, 2017] in Java. While offering no GUI or direct Weka integration, it operates on standard arff files which eases interoperability with other algorithms, while also offering a host of Ant-Miner variations and improvements. [Otero et al., 2008] proposes an extension to Ant-Miner, called cAnt-Miner, that handles continuous data dynamically and therefore is no longer limited to symbolic data. A further extension, cAnt-MinerPB [Otero et al., 2013] allows for additional optimization by letting ants learn and exchange information about the entire rule lists instead of only single rules.

4.8 PART

PART is a rule learning algorithm that aims to combine principles from Decision Trees with those of popular separate-and-conquer algorithms like RIPPER. It does not rely on a global optimization stage and instead aims to improve the immediate local pruning by utilizing partial decision tress from which rules are extracted.
Following the separate-and-conquer structure, PART grows a decision tree in every rule building iteration by choosing the split point that results in the largest information gain and descending to the resulting child with the lowest entropy while not expanding any other children. Once a leaf is pure, the algorithm backtracks, expanding trees that are adjacent to the leaf. The growing stage ends with a partially expanded decision tree as soon as a node with fully expanded children is encountered, starting a pruning stage where sub-trees are replaced with leaves and the algorithm backtracks. A parent node is pruned to a leaf if this reduces the estimated error. Traditionally a pessimistic error estimate is obtained using confidence intervals with a threshold c on the training data. A higher confidence value will result in less aggressive pruning. [Witten et al., 2011] notes that this estimate is based on a number of shaky statistical assumptions, but works reasonably well in practice.
Should the child of a node that is currently considered for pruning be a non expanded subtree, the pruning stage is interrupted, the subtree is expanded and then pruned as previously described. Rule building terminates if a node is not pruned, choosing the path from the root to the leaf with the largest coverage as antecedent and the majority class in the leaf as label.

Algorithm 8 PART [Frank and Witten, 1998]

Input: training data (x, y), a confidence level or a prune set for error estimation

```
R = ∅                                                          ▷ initialize rule set
while not all positive examples are covered do
    n = (x, y)
    Expand:                                                    ▷ recursive label if more expansion is required
    while entropy of n can be decreased do
        n = ExpandTreeAndDescend(n)                            ▷ find the best rule for the current examples
    end while
    Prune:                                                     ▷ recursive label if more pruning is required
    if not all children of n are expanded then                 ▷ only allowed to prune after the consequences are known
        n = GetUnexpandedAdjacentNode(n)                       ▷ expand only nodes that so far have not been expanded
        GoTo: Expand
    end if
    if ReplaceWithLeaf(n) lowers error estimate then           ▷ check if we need to prune, if not terminate rule construction
        n =ReplaceWithLeaf(n)
        n = Parent(n)                                          ▷ Backtrack up the tree
        GoTo: Prune
    end if
    r = GetLargestCoverage(n)                                  ▷ read path of leaf with largest coverage as rule
    (x, y) = (x, y) \ {(x, y) covered by r}                    ▷ remove covered examples
    R = R ∪ r                                                  ▷ add rule to list
end while
```

Output: the learned rule set R

Search Bias: PART works much like many separate-and-conquer algorithms during rule construction, but applies less greedy pruning. To prevent a rule from being pruned to early, PART only prunes rules after having fully explored them. PART mainly aims to show that results similar to algorithms like RIPPER can be achieved with simpler schemes that do not need to perform global optimization of the rule list.
Over-fitting Avoidance Bias: Besides the pessimistic error estimation, the Weka implementation of PART offers the more well founded alternative of reduced error pruning using a separate pruning set, but in contrast to RIPPER, PART does not have a global optimization stage that can be iterated with different pruning sets. Information contained in the pruning set will therefore not find its way into the model.
Implementation specifics: Like RIPPER, PART can deal with numeric attributes as well as multi-class problems. It is conceptually simple with easy to understand effects of its few tuneable parameters. Together with its integration in Weka and portable Java implementation it requires little effort to setup and run on any dataset. It can easily be integrated into a Python environment or can be used through the Weka provided GUI.

5 Evaluation

This section elaborates on the steps that were taken to transform the data to be suitable for binary classification and briefly mentions the tools used.

5.1 The evaluation environment

As previously mentioned, all algorithms are to be evaluate using the exact same datasets for training and testing. To achieve this, three main functions need to be performed:

- Reading the data and preprocessing it so it is suitable for each of the algorithms.
- Breaking up the multi-class classification task into a series of binary classifications and training a model for each of them.
- Evaluation of the generated rules and aggregation of the results.

Preprocessing is mostly implemented using the machine learning suite Weka through a Python wrapper. Developed by the University of Waikato, Weka offers many useful tools for data conversion and analysis while also providing the used implementations for RIPPER and PART. Implementations for all the other algorithms used in this paper can be obtained easily through the standard python packet manager, with the exception of Ant-Miner which needs to be called as separate program. Evaluation was implemented as part of a meta-classifier that generates multi-class predictions from the binary rules. The results of all folds were aggregated and visualised for classification accuracy, runtime and model complexity. Additionally parameter optimization for some of the datasets was run using the python library Hyperopt. As a final step the difference in accuracy and model complexity using binary, discrete and non-preprocessed datasets was compared.

5.2 Preprocessing

The preprocessing consists of multiple steps, all of which can be easily implemented using Weka. All the datasets are provided using the standard Weka arff file format. To get more consistent test results, every evaluation used 5 times repeated 2-fold cross-validation. Since CORELS and OFRL are limited to processing only fully binary datasets, the next step is a discretization of the data. The Weka provided supervised discretization filter is applied, which transforms all numeric attributes to symbolic using an algorithm by [Fayyad and Irani, 1993]. As a supervised method is used, it is required to generate the filter on each of the training folds individually before applying them to the train and test sets. While a discrete test set is not required, it eases later evaluation tremendously, as attributes that are used in the rules are consistent with those in the test set. Next, a one-hot encoding as described earlier in this thesis is applied for all attributes except the class label, converting them into a series of disjointed binary attributes. For datasets with many numeric attributes, this combination of discretization and one-hot encoding can lead to a large number of around 100 binary attributes. As some of the algorithms do not only require binary training data, but also a restriction to binary classification, the class attribute needs to be processed separately. Two different kinds of multi-class to binary classification transformations exist: *one-against-all* and *one-against-one*. One-against-one classification for a class label with n different values creates a set of $n(n-1)/2$ classification problems, each pitting one class label against another, while all instances that do not belong to either class are ignored. For each of these problems a separate classifier is learned. During classification of an instance, each classifier makes a prediction, which then gets combined with all other predictions to assign a final class label. This combination process can be a simple vote or more complex method involving its own meta classifier using the individual predictions as features. While this approach tends to produce the most accurate predictions, this thesis opted for a simpler approach with the following justifications:

- This thesis mainly aims at a providing a comparison between algorithms and not a comprehensive analysis on specific datasets, the performance delta is of greater interest than absolute accuracy values.
- As a result of the applied data transformations, the performance will not necessarily be representative for the original data even using a more sophisticated problem transformation.
- A complex meta classifier effectively adds another algorithmic layer between the generated rules and the final prediction, making results harder to interpret and possibly less representative.
- Many algorithms that are able to process multi-class problems like RIPPER use an approach similar to that of one-against-all by learning rules for each class individually.

In this thesis, we opted for the approach of learning rules for each class individually. For each class label a filter is applied that merges all other classes to a negative class label, while the respective class is kept as positive, producing n files on which n classifier are trained. The implementations of CN2, RIPPER, PART and Ant-Miner all support arff as file formats, while OFRL and CORELS are supplied with csv files. As csv files do not contain any metadata that identifies the class label, it is required to pass it as an argument. To simplify this process another filter renames the class attribute to a constant value for all files. As a final step, the datasets are saved as arff and csv files using Weka provided converters.

5.3 Training and evaluation

The training is performed on multiple cross-validation runs, drawn from stratified data to lower variance. All classifier produce a string representation of the learned rules, which a parsed into a uniform format for evaluation on the test set. As the one-vs-all approach trains n different classifier, it also is the case that every example is classified n times. These classifications will often contradict each other making it necessary to resolve conflicting predictions. An intuitive way to achieve this is choosing the rule with the highest Laplace corrected precision $\frac{tp+1}{tp+fp+2}$ on the training data. Applying the Laplace correction has the effect of discouraging the selection of extremely specific rules that capture only a few instances, while having no effect on more general rules. One observed issue with this approach can arise should an algorithm learn two accurate default rules that predict the positive class label. The more precise rule will always overshadow the competing default rule and cut it off from any potential examples it could capture. The individual classifications of the test examples are aggregated in a confusion matrix for every fold and repeated run of the cross-validation. To obtain the accuracy over all folds and runs, the individual matrices are added together. Additionally, the overall elapsed time for training of the classifiers is measured, as well as the total amount of produced rules and antecedents. As a measure of robustness, the standard deviation of accuracy between the individual folds and runs is also taken.
All algorithms were run with their default settings in this initial run, with the sole exceptions of FRL and softFRL. The number of iterations for both of these were cut to 300 from 3000 and 6000 respectively due to their excessive run time on some of the datasets. On smaller datasets the effect on accuracy was observed to be moderate in some trial runs. This first set of results was achieved with a the already mentioned five times repeated run of 2-fold cross-validation to achieve a sensible trad-off between variance and runtime. As can be seen in the time table, this run took a total of roughly 30 CPU hours on an i5 6600k on stock clocks purely for training, with additional time being needed for the repeated evaluation. Besides tables listing the results, some statistical test as described in the earlier evaluation chapter of this thesis are used and graphically presented.
Following the conclusions of these initial results, some further runs with tweaked parameters were conducted.

6 Results

In this section the results of the evaluation will be presented and discussed. It is divided into separate sections for accuracy, runtime, and complexity. For each subsection some additional runs with tweaked parameters were used to judge what kind of results could be achieved with some obvious adjustments to the default parameters. In a later sub-sections a more systematic approach with hyper-parameter optimization was used for a small part of datasets.

6.1 General observations

The first apparent observation stemming from **Table 3** showing the average accuracy and relative rank of each algorithm, is that CN2 is not at all competitive. While it achieves the highest rank 3 times, it does perform worse than the majority on all but one dataset. As can be seen in **Table 5** CN2 generates by far the longest rule lists, which indicates that it over-fits the data strongly, showing the limitations of its simple confidence level approach. **Table 4** shows it is also among the slowest algorithms in this comparison, together with FRL and softFRL. All of these algorithms are implemented in Python which probably contributes significantly to the performance loss compared to all other algorithms that are written in compiled languages like Java and C++. CORELS is the second worst performing algorithm, which can be attributed to a strong underfitting of the data. **Table 5** shows it consistently producing some of the smallest rule sets, often quite significantly so.
Surprisingly softFRL performs considerably worse than FRL in both accuracy and runtime, while producing slightly larger rules. The runtime delta is to be expected as softFRL needs to consider more possible rules in each iteration, but even though it has a less restricted search-space it produces a less accurate rule set. While in theory surprising, these results seem to be in line with [Chen and Rudin, 2018]. They also note that, due to the low default rule length penality, it can happen that the relaxation of the monotonicity constraint leads to over-fitting. FRL performs quite competitively both in terms prediction accuracy and model size, which should lead to its rule lists being some of the most interpretable ones. cAntMinerPB is the best performing algorithm overall and is significantly more accurate than all algorithms except RIPPER,PART and FRL, though it also produces larger models than some of the other algorithms. PART, like CN2, prunes based on confidence levels and produces some of the largest models, but contrastingly manages to maintain very good accuracy. Ripper is even faster and more accurate than PART, all while producing some of the smallest models.
Figure 55 illustrates how the algorithms stack up in terms of average accuracy instead of ranks. Due to su-par results on some of the datasets (i.e. anneal, autos, vehicle) CORELS and both version are listed lower than their average rank would suggest. The results of a parameters optimization on some of these datasets are further analysed in a later section of this thesis in order to gain some insight into this unstable behaviour.

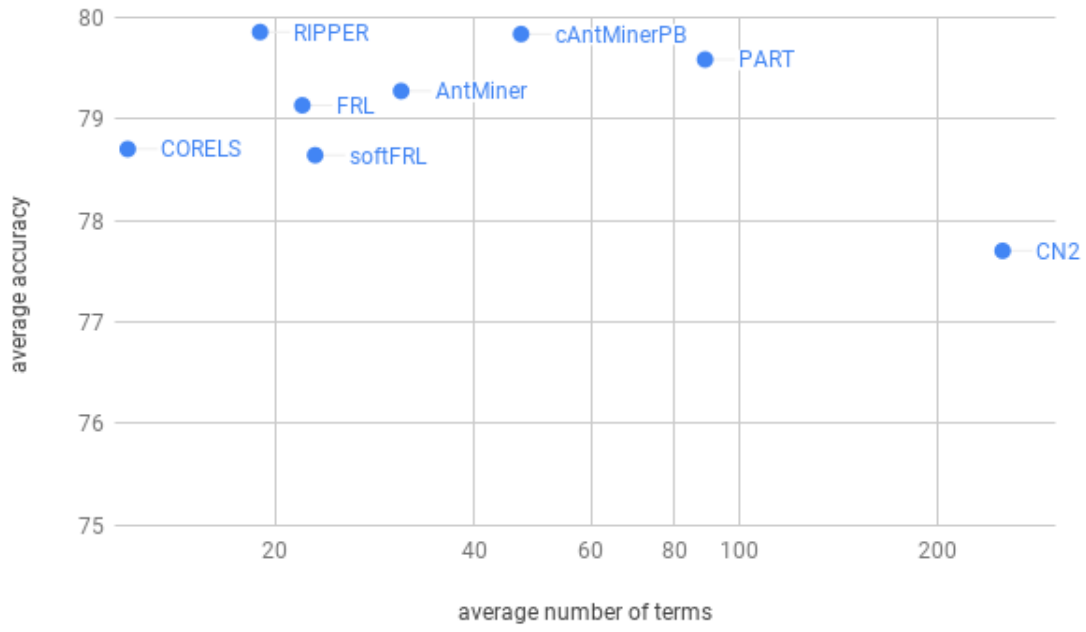


Figure 5: Average accuracy over logarithmic complexity

6.2 Statistical tests

The statistical test described in the evaluation section of this thesis were applied in order to determine whether the measured differences in accuracy are significant. The Friedman omnibus test passes, allowing us to use post-hoc tests for further analysis. The Nemenyi test computes a critical difference of 1.89, according to which cAntMinerPB, RIPPER, PART, FRL and AntMiner do not differ significantly in prediction accuracy with a p-level of 0.1.

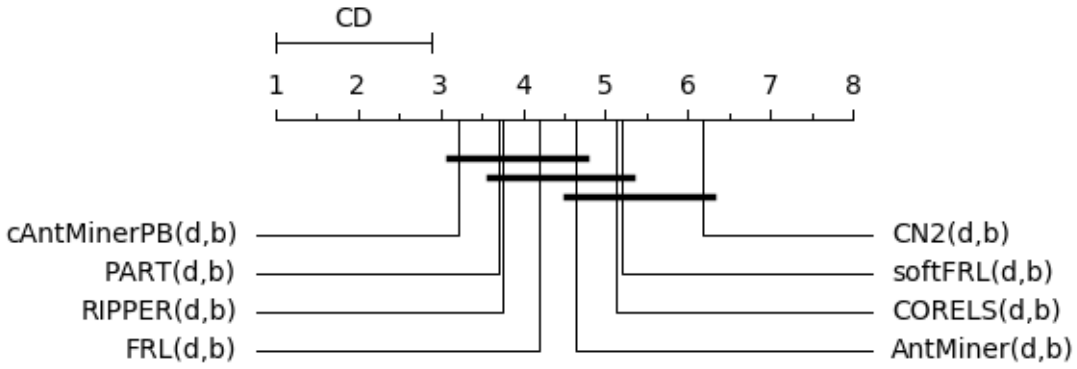


Figure 6: Result of the Nemenyi test with p-level 0.1

Applying the pairwise Conover-Friedman test with the p-adjustment by Hommel results in less conservative estimate that looks that has been colour coded for p-levels of 0.1, 0.05 and 0.01. The less conservative nature shows in that it measures a significant difference between cAntMinerPB and AntMiner, as well as other cases where the critical difference computed by the Nemenyi test is close to being exceeded.

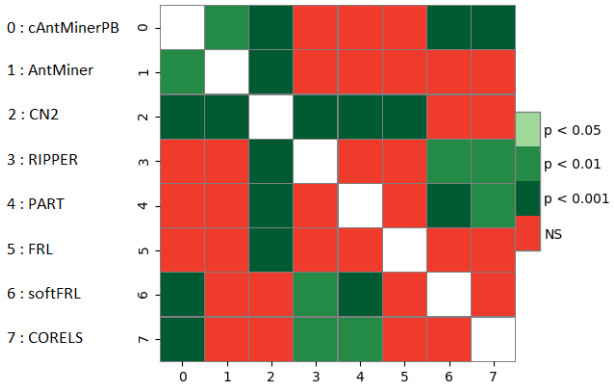


Figure 7: Result of the Conover-Friedman test

Regarding the relative performance to each other, it is an interesting observation that both cAnt-MinerPB and RIPPER are the least likely to fall into the bottom quantile of Ranks 7 and 8, while CN2 and CORELS fall into this bracket most often. The upper performance quantile is dominated by cAnt-MinerPB, with a considerable 42% share of its results falling into this category, while RIPPER achieves these upper ranks only 23% of the time. Only CN2 and softFRL achieve a lower share. RIPPER performs consistently well, but rarely exceptionally so, and also exposes few parameters to the user with which it could be adopted to perform better on specific problems. CORELS achieves a relatively high share of 27% for the upper quantile considering its also large share of 42% of the worst ranks. This suggests that CORELS is very reliant on a good parameter choice and requires an understanding of the data and rules that will perform well.

0.25 and 0.75 rank quantiles	cAntMinerPB	AntMiner	CN2	RIPPER	PART	FRL	softFRL	CORELS
Rank 1 & 2	42%	31%	12%	23%	35%	31%	7%	27%
Rank 7 & 8	0%	27%	62%	8%	15%	12%	35%	42%

Table 1: Share of achieved results in the 0.25 and 0.75 rank quantiles

Another important factor to consider is the standard deviation of accuracy shown in **Table 2**, which, even with the various

applied measures for variance reduction, remains relatively high.

standard deviation	cAntMinerPB	AntMiner	CN2	RIPPER	PART	FRL	softFRL	CORELS
max	6.7	4.9	7.2	7.5	7.2	8.6	9.5	7.8
avg	2.2	2.4	2.6	2.4	2.6	2.7	2.5	2.3

Table 2: maximum and average standard deviation over all results

While a deviation of around 2.5 percentage points seems small, the algorithms are so closely matched that the measured differentiation often falls below the standard deviation of the algorithms. For example considering FRL and cAntMinerPB, only 8 out of 26 datasets show a separation in accuracy that exceeds the standard deviation of cAntMinerPB.

% accuracy & ranks	cAntMinerPB		AntMiner		CN2		RIPPER		PART		FRL		softFRL		CORELS	
anneal	98.42	3	98.66	2	99.06	1	98.17	5	98.33	4	96.17	6	95.77	7	93.56	8
autos	65.17	1	63.80	2	60.78	6	62.05	4	62.93	3	58.93	7	55.12	8	60.98	5
balance-scale	74.72	6	73.34	8	77.06	1	76.10	5	76.22	3	76.19	4	74.62	7	76.67	2
breast-cancer	68.95	6	70.91	4	63.57	8	72.52	1	65.59	7	69.72	5	72.03	3	72.31	2
breast-w	94.88	6	95.25	2	93.53	8	94.96	5	95.05	3	95.42	1	95.02	4	94.85	7
colic	82.17	5	81.68	6	75.11	8	85.27	3	78.97	7	85.49	1	84.89	4	85.43	2
compas	66.94	1	66.29	5	66.42	4	66.87	3	66.91	2	66.19	6	65.38	8	65.93	7
credit-a	85.77	1	84.23	6	78.96	8	85.57	2	83.68	7	85.01	5	85.13	4	85.51	3
credit-g	71.52	1	70.44	6	67.30	8	71.00	4	69.62	7	71.46	2	71.22	3	70.90	5
diabetes	74.45	3	73.59	8	73.75	7	74.77	1	74.43	4.5	74.43	4.5	74.06	6	74.51	2
glass	59.81	2	59.25	4	59.44	3	59.07	5	60.28	1	56.64	7	55.98	8	58.32	6
heart-c	78.28	5	77.56	7	73.93	8	79.67	4	80.07	3	80.66	1	80.33	2	77.89	6
heart-h	79.80	6	80.54	4	81.09	1	79.12	8	81.02	2	80.34	5	80.82	3	79.18	7
heart-statlog	79.33	2	77.19	7	75.19	8	79.04	4.5	79.93	1	79.04	4.5	79.26	3	78.81	6
hepatitis	83.61	1	83.48	2.5	82.32	6	81.29	7	83.10	4	83.48	2.5	82.45	5	80.26	8
hypothyroid	99.20	2	99.18	3	98.67	8	99.06	4	99.21	1	98.87	5	98.86	6	98.85	7
ionosphere	90.26	3	89.52	7	90.09	4.5	90.43	2	89.74	6	91.00	1	90.09	4.5	87.24	8
iris	94.67	5	94.80	2	93.07	8	94.67	5	94.67	5	94.8	2	94.40	7	94.8	2
kr-vs-kp	98.10	4	94.47	5	98.24	3	98.69	1	98.62	2	91.59	6	91.22	7	90.49	8
labor	83.16	3.5	83.86	2	80.35	8	83.16	3.5	82.46	5	81.05	7	81.40	6	84.91	1
lymph	78.11	2	78.65	1	74.59	8	77.43	5	76.76	6	77.57	4	75.81	7	77.70	3
primary-tumor	36.81	3	37.40	2	33.45	8	36.64	4	40.29	1	34.34	6	34.51	5	34.16	7
sick	97.75	1	97.35	7	97.19	8	97.61	3	97.65	2	97.41	6	97.48	4.5	97.48	4.5
sonar	70.77	4	70.38	6	68.46	8	71.44	2	70.87	3	71.83	1	70.67	5	68.75	7
vehicle	67.59	2	64.54	5	65.34	4	65.93	3	67.90	1	64.18	6	61.89	7	60.90	8
vote	95.45	5	94.53	7	93.24	8	95.68	3.5	94.80	6	95.68	3.5	96.09	1	95.72	2
average rank	3.27		4.58		6.17		3.65		3.79		4.17		5.17		5.19	

Table 3: Accuracy and rank between classifier on 5 runs of 2-fold CV

6.3 Runtime

Table 4 contains the aggregate training time of each algorithm over all 5 runs of 2 fold cross-validation. As already mentioned, CN2, FRL and softFRL are by far the slowest algorithms, while all other algorithms return results for all datasets in a couple of minutes, with RIPPER and PART only requiring a few seconds each.

time in seconds	cAntMinerPB	AntMiner	CN2	RIPPER	PART	FRL	softFRL	CORELS	sum
anneal	346	177	5143	**13**	**13**	4377	8778	83	18929
autos	247	173	2945	**5**	5	2723	8924	106	15129
balance-scale	88	62	6	5	**4**	21	45	32	264
breast-cancer	113	175	697	**2**	5	422	1115	25	2552
breast-w	70	136	26	**4**	**4**	183	500	59	982
colic	109	116	1175	**2**	3	1236	3500	22	6165
compas	820	1243	306	**27**	41	221	289	213	3161
credit-a	134	320	779	**4**	5	422	1133	20	2816
credit-g	296	216	4050	**6**	18	1201	2658	22	8467
diabetes	62	59	10	**4**	**4**	31	71	35	276
glass	71	71	6	**3**	**3**	43	123	36	356
heart-c	48	31	36	**2**	**2**	143	458	18	739
heart-h	50	56	24	**2**	**2**	76	212	19	440
heart-statlog	24	30	9	**1**	**1**	45	140	23	275
hepatitis	32	21	9	**2**	**2**	92	278	18	453
hypothyroid	348	235	1535	35	**29**	1243	2018	59	5503
ionosphere	101	39	290	**3**	4	4170	8666	20	13293
iris	24	19	1	2	**1**	11	28	16	103
kr-vs-kp	367	93	214	26	**18**	773	1519	30	3039
labor	21	13	8	**1**	**1**	89	287	14	433
lymph	62	47	113	3	**2**	444	1460	27	2157
primary-tumor	360	285	141	16	**14**	499	1453	404	3172
sick	299	395	1221	20	**15**	553	894	32	3427
sonar	63	25	511	**2**	**2**	257	748	19	1626
vehicle	293	623	1167	**8**	11	1094	3200	39	6435
vote	38	20	10	**2**	**2**	86	257	73	488
sum	4487	4681	20432	**200**	212	20452	48755	1463	108020

Table 4: Combined runtime for 5 runs of 2-fold CV

All algorithms show an increase in runtime with growing attribute count, but the antecedent driven approach of FRL and softFRL seems to struggle especially with datasets containing a large number of attributes as can be seen in **Figure 8** below. CORELS does not exhibit the same behaviour even though it too relies on a pre-minded set of antecedents, implying that the used pruning bounds on the search space are very effective. **Figure 8** also shows a separation into 3 different groups on the logarithmic time scale, corresponding to the expected runtime of seconds for RIPPER and PART, minutes for CORELS, AntMiner, cAntMinerPB, and, depending on the dataset, hours for FRL, softFRL and CN2.
The number of instances seems to be a negligible factor for the slowest performing algorithms, as can be seen in **Figure 9**. RIPPER and PART are more sensitive to the instance count than the number of attributes, while AntMiner and cAntMinerPB runtimes seem to increase with both. CORELS once again is somewhat of an anomaly, probably due to the fact that its runtime is heavily influenced by how effectively it can perform pruning of the search space more so than attribute or instance count.

As many applications are not very time sensitive, another run of FRL with its original default parameter of 3000 iterations was conducted, increasing its runtime to roughly 56 CPU hours. There were no improvements in terms of average accuracy and none of the changes exceed 2 percentage points with no significant change in the relative ranking of the algorithms. A slight drop in the number of learned terms by 3 percentage points was observed, indicating a slightly higher rule quality.
RIPPER was also rerun with 20 instead of 2 optimization runs, as its very fast runtime leaves a lot of potential headroom. The 10 fold increase in optimization iterations resulted in almost no change in average accuracy, but a slight improvement of 0.2 ranks relative to the other algorithms. Rule length increased by 5% and runtime remained among the fastest algorithms.

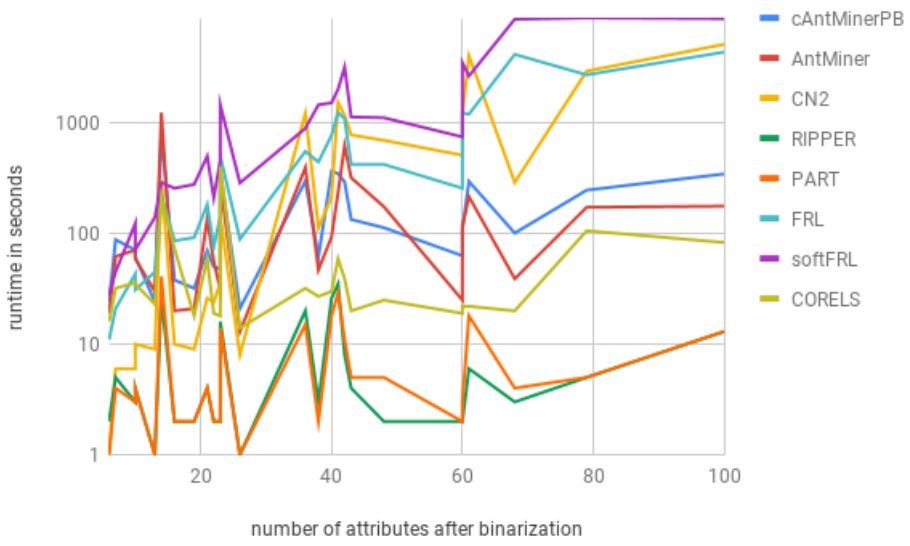


Figure 8: Plot of runtime on the dataset over the number of attributes

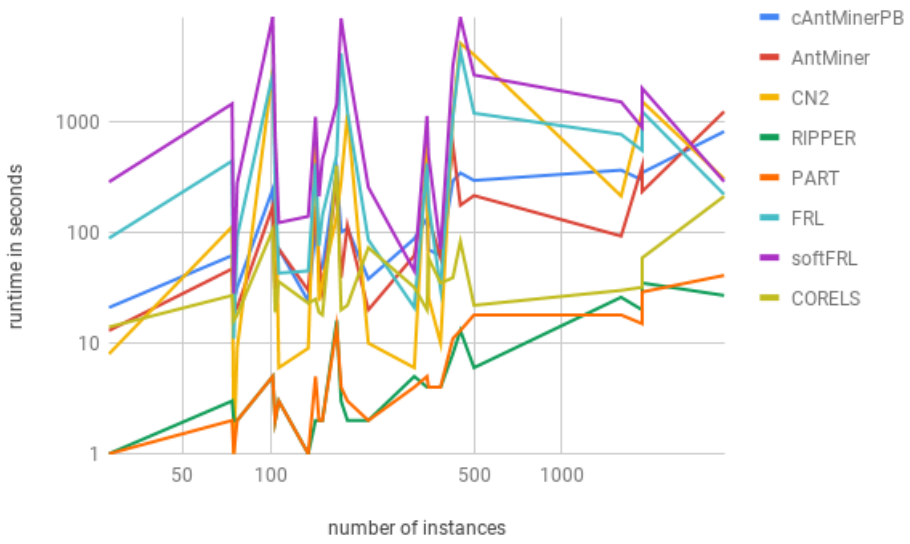


Figure 9: Plot of runtime on the dataset over the number of instances

6.4 Number and length of rules

One obvious conclusion following **Table 5** showing the aggregate number of rules and average rule length is, that CORELS massively underfits the data. As a consequence, CORELS was rerun with an adjusted rule length penality parameter of $C = 0.001$, 1/10th of its default value. This value was chosen after experimentation on some datasets with the aim of bringing the number of terms close to those learned by RIPPER. While improvements were minor in absolute terms and only averaged to 0.2 percentage points, they lead to an improvement in CORELS average rank by 0.73, showing how closely all algorithms perform to one another. The loosening of the rule length penality resulted in an increase of about 25 percent of rule terms, brining them in line with RIPPER. Surprisingly this change also had the side effect of reducing the overall runtime of CORELS by around 30 percent. One possible explication might be an interaction of the rule length penality with the default best-first search policy employed during the exploration of nodes. A lower length penality leads to more specific rules being learned earlier, which could result in more aggressive pruning later on and would explain the lower observed runtime.
A further reduction by another 1/2 of the rule length penality and an increase in the permitted cardinality of antecedents were tried, but resulted in a loss accuracy.

PART on the other end of the spectrum creates amongst the most complex rule sets. As mentioned in the algorithm section, it also offers the possibility of using reduced error pruning with a hold out set instead of a confidence level approach, promising better pruning results. The rule lists produced by reduced error pruning were only half as long as those generated through confidence based pruning, but also significantly less accurate. On average, PART lost almost an entire rank relative to the other algorithms. This significant loss in accuracy is a can be traced to the further reduced size of the already small train set resulting from the need to keep a hold out set and the lack of an additional optimization stage like RIPPERs allowing this information to be recovered.
Another interesting consideration is the spread in the number of learned rules amongst the best performing algorithms. cAnt-MinerPB and PART learn some of the most complex models, while RIPPER manages to maintain an almost equal level of performance with a far smaller number of rules.

rules & terms	cAntMinerPB		AntMiner		CN2		RIPPER		PART		FRL		softFRL		CORELS	
anneal	239	1.74	162	1.68	821	1.04	150	1.49	167	2.11	178	1.71	199	1.75	**101**	**1.31**
autos	287	2.6	211	2.65	1804	1.03	158	1.64	268	2.68	290	1.78	295	1.79	**137**	**1.56**
balance-scale	178	1.36	168	1.42	602	2.4	121	2.18	180	1.69	**100**	**1.7**	104	1.71	104	1.7
breast-cancer	129	3.47	100	3.14	1397	1.63	**46**	**1.3**	322	5.89	103	1.81	112	1.81	50	1.6
breast-w	128	1.88	104	2.02	664	1.51	93	1.82	158	2.3	117	1.81	126	1.83	**58**	**1.66**
colic	144	3.03	102	3.37	1365	1.52	53	1.64	214	3.64	106	1.81	127	1.84	**48**	**1.58**
compas	269	1.78	184	2.58	2867	3.34	113	2.34	490	3.59	105	1.74	103	1.77	**58**	**1.66**
credit-a	189	3.21	118	3.39	2248	1.79	85	1.82	346	3.84	91	1.77	98	1.79	**42**	**1.52**
credit-g	460	5.57	145	4.46	4491	1.86	63	2.41	888	6.48	135	1.85	133	1.84	**44**	**1.55**
diabetes	135	1.39	114	1.65	683	2.64	67	1.99	144	2.15	87	1.75	94	1.76	**62**	**1.68**
glass	206	1.39	202	1.35	520	1.3	116	1.62	210	1.46	**122**	**1.51**	123	1.51	147	1.57
heart-c	139	2.29	100	2.02	894	1.75	75	1.89	206	2.44	111	1.82	118	1.82	**62**	**1.68**
heart-h	114	2.44	98	2.24	636	1.88	**57**	**1.56**	134	2.88	83	1.75	89	1.78	66	1.67
heart-statlog	117	1.73	100	1.56	740	2.27	79	1.75	186	2.19	98	1.8	112	1.81	**78**	**1.74**
hepatitis	76	1.8	76	1.89	440	1.43	**52**	**1.52**	106	2.17	97	1.77	118	1.81	68	1.68
hypothyroid	194	1.94	169	1.8	1425	1.26	97	2.07	152	1.73	125	1.68	120	1.66	**80**	**1.5**
ionosphere	123	2.32	72	2.47	548	1.01	106	1.25	88	3.16	152	1.86	157	1.83	**54**	**1.63**
iris	79	1.04	85	1.15	169	1.06	**61**	**1.18**	65	1.17	79	1.52	79	1.58	65	1.32
kr-vs-kp	278	1.87	76	2.26	1674	1.67	262	2.85	356	2.92	77	1.73	87	1.75	**60**	**1.67**
labor	56	1.5	52	1.73	174	1	**55**	**1.15**	52	1.46	68	1.66	67	1.7	56	1.61
lymph	148	1.72	118	1.81	575	1.14	**91**	**1.45**	159	1.8	165	1.75	164	1.74	100	1.43
primary-tumor	648	1.45	794	1.68	4441	1.36	280	1.52	704	2.08	334	1.42	340	1.42	**265**	**1.26**
sick	154	2.26	120	2.08	1904	2.2	79	2.16	198	3.02	79	1.7	84	1.73	**40**	**1.5**
sonar	135	1.85	88	1.91	556	1.67	72	1.58	164	2.24	108	1.8	113	1.82	**62**	**1.68**
vehicle	486	2.74	253	2.21	3780	2.07	166	2.57	761	3.98	191	1.79	206	1.8	**103**	**1.6**
vote	95	1.59	94	1.36	550	1.6	52	1.5	116	1.57	80	1.66	91	1.71	**50**	**1.36**
average	200	2.35	150	2.08	1383	1.81	101	1.87	262	3.37	126	1.72	133	1.73	**79**	**1.53**
rank_avg	5.46		4.38		8		2.38		6.29		3.56		4.37		**1.56**	

Table 5: Total number of rules and average antecedents per rule for 5 runs of 2-fold CV

7 Parameter optimization

Table 3 shows that CORELS and FRL do very well amongst a number of datasets and perform considerably worse than all other algorithms in some others. In order to gain more insight into what causes this behaviour and which parameters to select for a given problem, hyper-parameter optimization was performed for datasets that produced subpar results with the default settings. To make this computationally feasible, the datasets were restricted to those with a shorter runtime (see **Table 4**) and only parameters that do not directly increase the runtime of the algorithm or can cause erratic behaviour were optimized. The used method was developed by [Bergstra et al., 2013] for automatic hyper-parameter optimization in computer vision tasks. For each algorithm, it's default parameters were used as initial exploration point, and restricted to very broad sensible intervals. The objective to maximize was the classification accuracy on the test sets, with the number of antecedents and the runtime being secondary deciders should the results be otherwise identical. Directly optimizing on the test sets will obviously limit the conclusions that can be drawn in terms of parameter robustness and comparability with previous results. To gain some insight nonetheless, we will take a look at all the experiments conducted by the optimization algorithm for two representative datasets each, one with good or mediocre and the other with bad results, to see how large the range of well performing parameters is.

7.1 Optimal FRL parameters

A first attempt to optimize all parameters for FRL lead to disappointing results. The parameter for random termination introduced a lot of noise and lead to very high termination values for some of the datasets with only small improvident in accuracy. One possible reason for this behaviour is that a more robust encoding should have been chosen for prob_terminate during optimization. A second attempt was made, with the default value of prob_terminate = 0.01 and all other parameters being constrained to the following intervals:

- $C \in [0, 0.1]$
- max_card $\in \{1, 2\}$
- minimum support $\in [0, 1]$

The results can be seen in **Table 6**. The best parameters are much closer to one another than to the default parameters. Especially the length penality C is much larger, indicating that the default parameters lead to rules that are to long. The values for λ and min_supp interact with each other, in that a high value for λ, which results in a tight fit to the data, indicates a high value for min_supp, which will result in a looser fit, suggesting a trade-off between the two. Only if the min_supp parameter is very large does the λ approach or exceed the default value. Choosing a higher value for min_supp is hugely beneficial to the runtime of the algorithm, but carries the risk of being much less robust than an adjustment of the λ value. In regards to maximum cardinality, all files showed a clear improvement in accuracy if left at the default value of 2 instead of being lowered to 1. It should also be mentioned that further increases in cardinality would sometimes lead to large improvements, but the large increase in runtime made further optimization with values over 2 unfeasible.

FRL	balance-sale	diabetes	glass	heart-h	heart-statlog	default
C	0.0016	0.0002	0.0113	0.0059	0.055	0.000001
min_supp	0.15	0.28	0.72	0.67	0.09	0.1
λ	0.6374	0.2541	0.5037	0.1449	0.1804	0.8
time diff	-5	-10	-32	-44	-13	
rule diff	-1	-11	-41	-20	-41	
acc gain	0.45	0.34	0.47	1.36	1.63	

Table 6: best found parameters for FRL

Figures 10 and 11 show all trials that were run during the 650 iterations of parameter optimization for the diabetes and glass datasets. The top 10% of best results are marked green, while blue indicates that the trial has at least one parameter that lies outside a bound were performance degrades significantly. These bound were chosen manually and are noted in **Table 7** below. The single red dot, if present, is the result achieved with default parameters.
Both figures show, that λ behaves as expected and mostly interpolates between 0 and the maximum value for minimum support that still produces good results. Adjustments in λ seem to result in steady changes of the accuracy, but because of the speed advantage, the optimization procedure converged towards higher values for the minimum support threshold instead of exploring the lower range of λ values. The C value seems much harder to optimize. It essentially introduces

thresholds outside of which any model generated is significantly worse. If a certain complexity of the model is required to accurately describe the data, a to high C value will consistently prevent such a model from being learned. A very low default value for C seems therefore to be more robust, but might introduce additional over-fitting. Especially for datasets that prefer a very low value for C it seems challenging to weight these two properties.

bounds	diabetes	glass
C	> 0.05	-
min_supp	> 0.35	> 0.85
lambda	-	-
max_card	< 2	< 2

Table 7: bounds used to mark results produced with very bad parameter settings

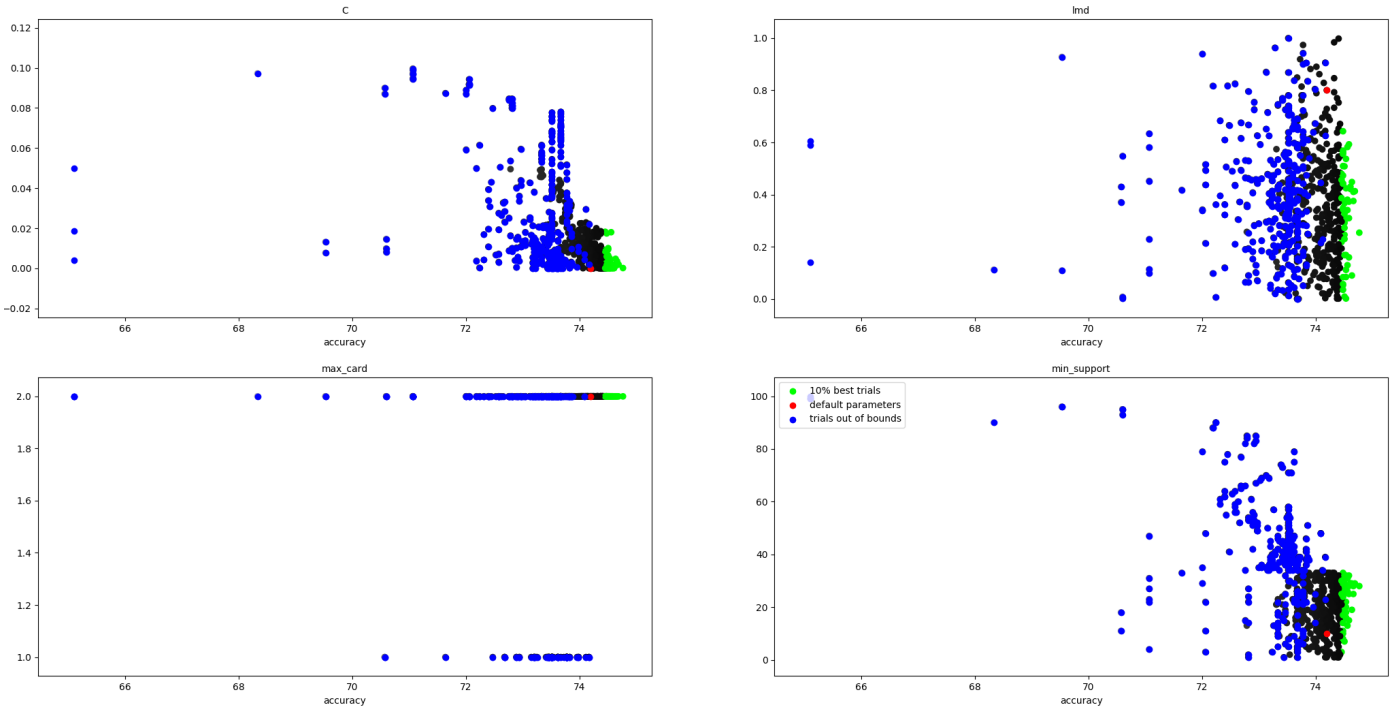


Figure 10: Plot of accuracy on the diabetes dataset for different FRL parameters

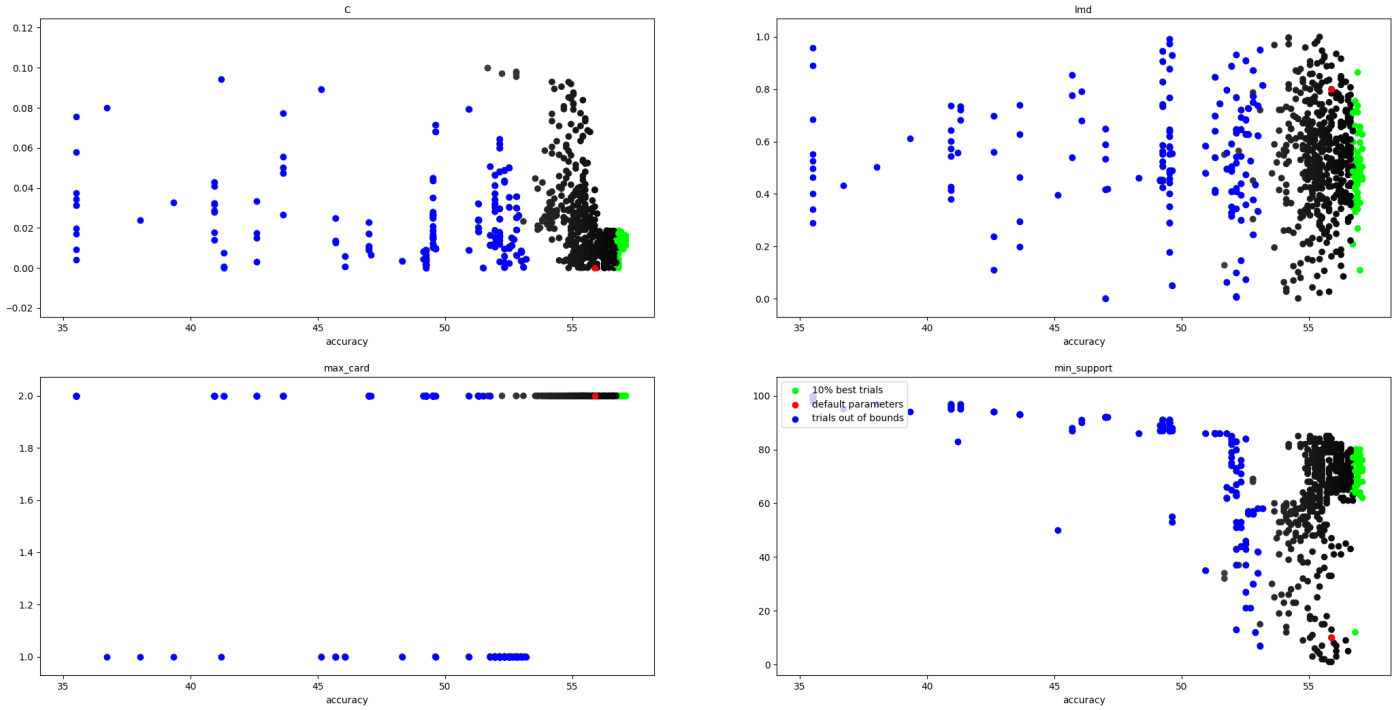


Figure 11: Plot of accuracy on the glass dataset for different FRL parameters

7.2 Optimal softFRL parameters

The search for softFRL parameters were constrained to the same values as for FRL.

- $C \in [0, 0.1]$

- $C1 \in [0, 2]$
- max_card $\in \{1, 2\}$
- minimum support $\in [0, 1]$

For most files, softFRL performed very similar to FRL after parameter optimization. As a result, the bounds to exclude obviously bad parameters are the same ones used for FRL in **Table 7**. For these datasets, values for C1 that are small usually resulted in a performance degradation, sometimes significantly so. A. Only for two files did softFRL considerably improve over FRL, while everywhere else small improvements with slightly larger models were recorded. The diabetes dataset profits from a much better C value that was found by the optimization algorithm, as this dataset seems to prefer very small C values. The other significant gain is observed on the glass dataset. Here a very small value for C1 sees huge gains in accuracy, showing that the monotonicity constraint can lead to substantial deficits on some datasets, while providing a useful source of bias for others.

softFRL	balance-sale	diabetes	glass	heart-h	heart-statlog	default
C	0.0016	0.0000009	0.0003	0.0439	0.0037	0.000001
min_supp	0.15	0.03	0.44	0.71	0.38	0.1
λ	0.6761	0.7131	0.9989	0.4892	0.3032	0.8
C1	0.5583	0.8074	0.0004	0.8292	1.7457	0.5
time diff	3	0	-34	-160	-79	
rule diff	2	8	115	-49	-16	
acc gain	2.37	0.76	4.39	0.61	2.44	

Table 8: best found parameters for FRL

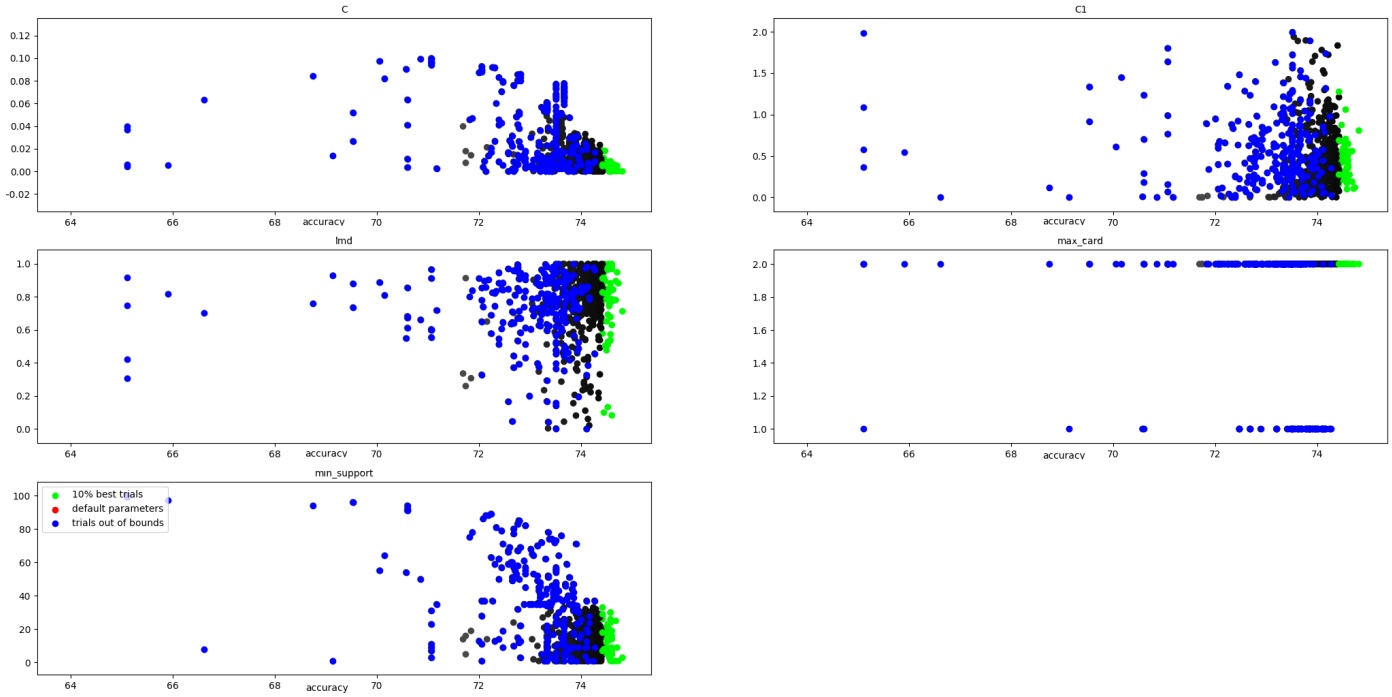


Figure 12: Plot of accuracy on the diabetes dataset for different softFRL parameters

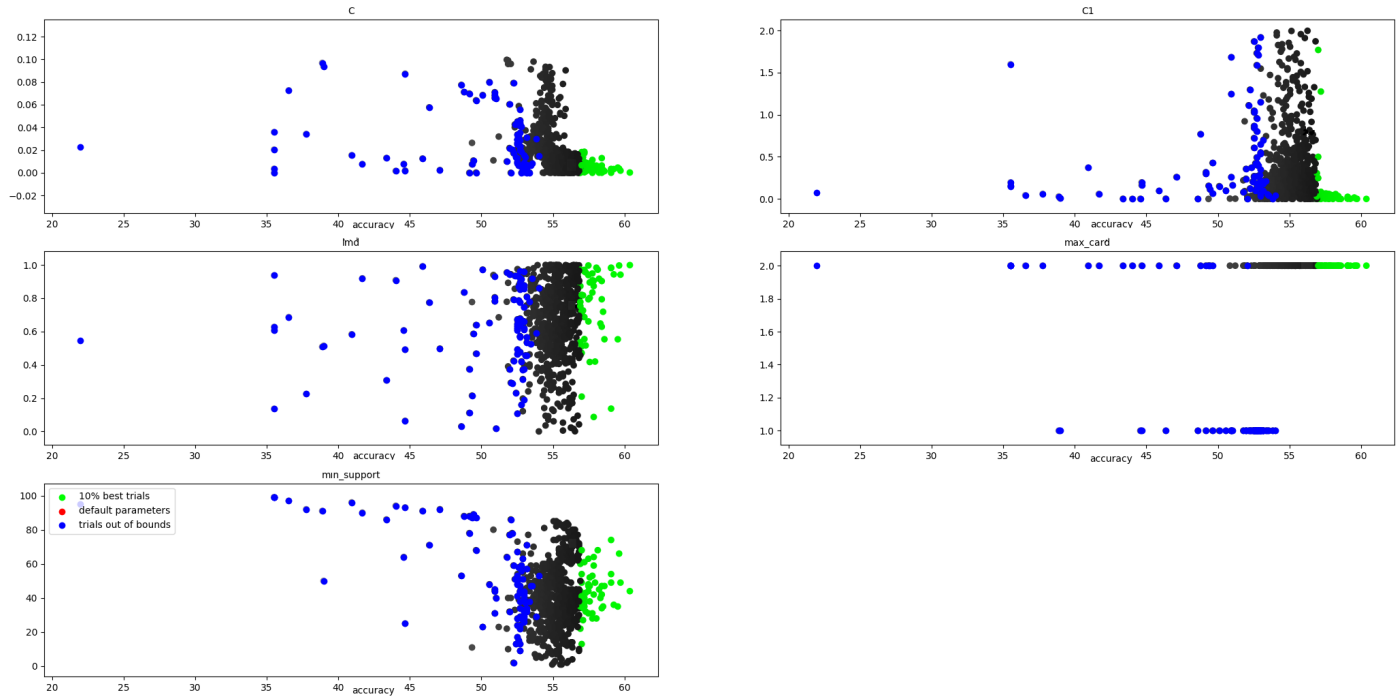


Figure 13: Plot of accuracy on the glass dataset for different softFRL parameters

7.3 Optimal CORELS parameters

The most apparent observation made during parameter optimization for CORELS was the importance of a suitable search strategy. If the chosen strategy is not able to locate a good solution somewhat early to enable pruning of the search space, CORELS essentially ceases to function. Runtimes are excessively long, and RAM usage continuously increases, sometimes to such a degree that no more RAM is available and CORELS exits with an allocation error. Breadth-first and Depth-first both suffered from this issues. The default lower-bound policy produced consistently good results, as did the curious search policy. The objective policy is a version of lower-bound that does not consider the default classification error, and usually produced slightly worse results. In the graphs these policies are listed as 1, 2 and 3 respectively. It is also interesting to note that CORELS sometimes produces better results with a lower maximum antecedent length, which is contrary to FRL always preferring the maximum available setting.

- $C \in [0, 0.01]$
- max_card $\in \{1, 2, 3\}$
- minimum support $\in [0, 0.75]$
- policy $\in$ {lower_bound(0), curious(1), objective(2)}

CORELS	diabetes	heart-h	hepatitis	ionosphere	sonar	default
C	0.006	0.0012	0.0018	0.0057	0.0062	0.01
min_supp	0.116	0.014	0.0986	0.0061	0.3089	0.1
max_card	1	3	1	1	1	2
policy	objective	lower_bound	curious	objective	lower_bound	lower_bound
time diff	-24	-4	-1	33	-6	
conditions diff	-12	4	80	22	4	
acc gain	0.04	3.27	3.87	3.65	4.13	

Table 9: best found parameters for CORELS

Figure 15 illustrates one fundamental issue with CORELS. Because of its optimization nature, CORELS will construct the exact same model for different parameters as long as the current model is still optimal. Each model is essentially a discrete bin for a certain set of parameters. Theoretically this can lead to very stable models if there is a model that is significantly better than all others, but should there be many models that are very close in prediction accuracy and length, this can lead to very unstable results. This seems to be a more pronounced version of similar effects observed with FRL, which uses a similar rule length penality. Because C and the minimum support are the only parameters to effectively control fitting behaviour, the model selection becomes extremely reliant on these parameters to land in the right "bin". Besides the wide spread amongst good results, **Figure 15** also shows, that even with relatively tight bounds many bad results remain. On the other end of the spectrum, the results for a dataset like diabetes that CORELS already performed well on with default settings are extremely stable for a wide variety of parameters.

bounds	diabetes	hepatitis
C	-	-
min_supp	> 0.15	$> 0.15 > 0.10$
policy	-	-
max_card	> 1	> 1

Table 10: bounds used to mark results produced with very bad parameter settings

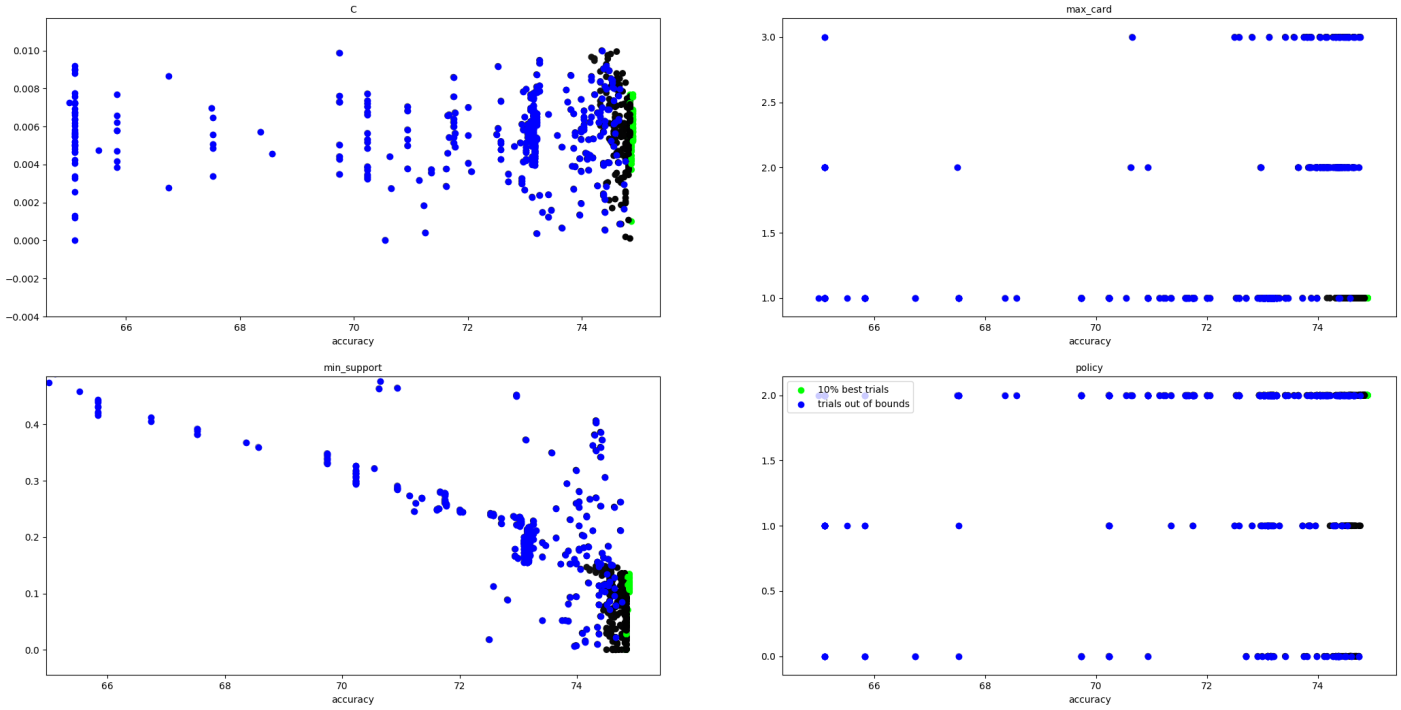


Figure 14: Plot of accuracy on the diabetes dataset for different CORELS parameters

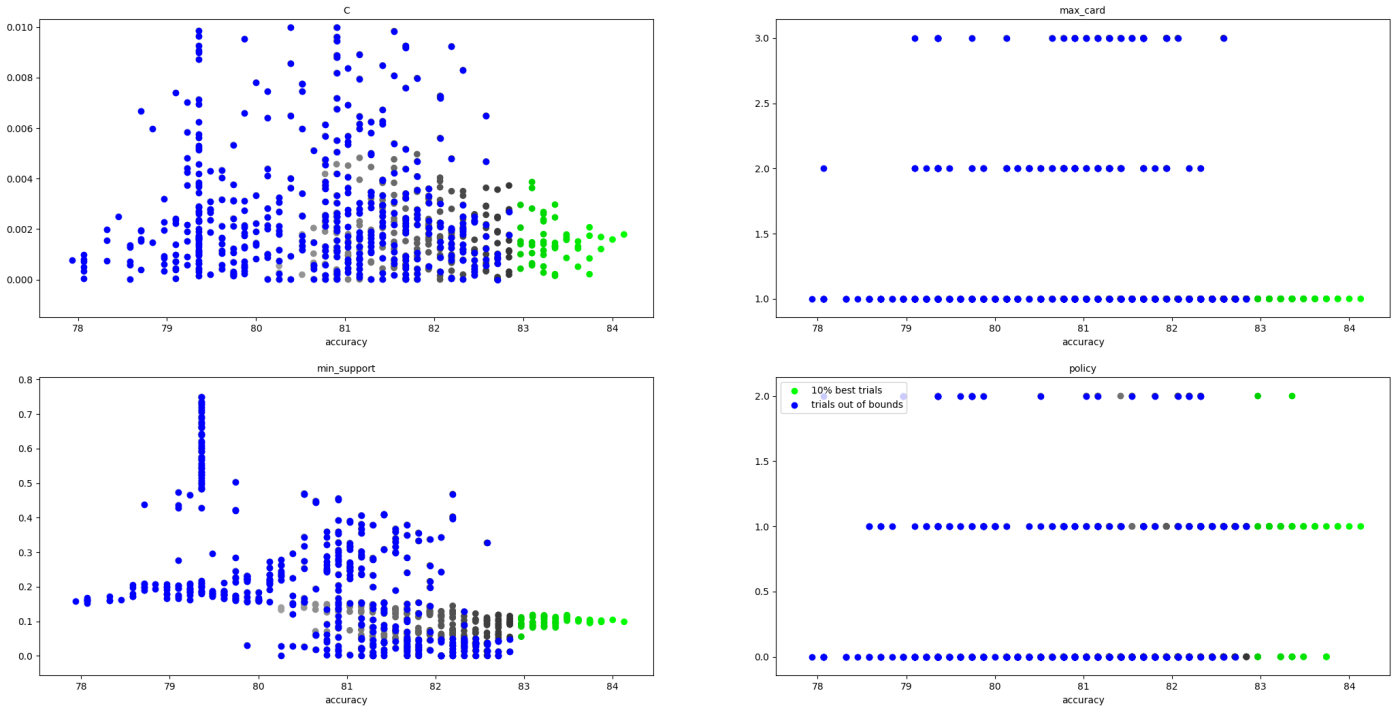


Figure 15: Plot of accuracy on the hepatitis dataset for different CORELS parameters

8 Influence of binary classification and discretization

To judge the potential skew of the results for problems that are neither discrete nor binary, a comparison between the results achieved with the binary classifier and applied static discretization, only discretization, and no significant preprocessing were compared against each other. The tables present changes in accuracy and the relative size of the discovered rule list compared to previous results.

8.1 Results without binary classification but with discretization

As shown by **Table 11**, the switch from the binary classifier to the built in methods to deal with multi-class problems leads to a very small decrease in prediction accuracy across all algorithms. At the same time, model size dropped by at around half for all algorithms, with PART being a big outlier as its model size drops to only 20% of the size achieved in a One-Vs-All scenario. This result reinforces the appearance that PART produced very confusing, redundant and hard to understand rule lists for the binary case that did not improve over a much simpler model.

change in accuracy & relative model size	cAntMinerPB		AntMiner		CN2		RIPPER		PART	
	acc	% terms	acc	% terms	acc	% terms	acc	% terms	acc	% terms
anneal	-1.63	0.749	-4.05	0.493	-1.36	0.510	-1.02	0.397	-1.07	0.310
autos	-5.46	0.355	-5.95	0.277	0.29	0.360	0.68	0.510	-1.27	0.170
balance-scale	1.66	0.450	0.70	0.364	0.29	0.397	0.83	0.545	1.25	0.283
breast-cancer	2.24	0.145	1.05	0.172	2.24	0.457	-1.33	0.667	3.50	0.059
breast-w	0.09	0.462	0	0.452	0.03	0.499	0.11	0.396	0	0.217
colic	1.09	0.161	0.11	0.235	-0.98	0.502	0.60	0.460	3.37	0.105
compas	-0.04	0.415	0.13	0.346	0.25	0.470	0.14	0.417	0.01	0.168
credit-a	0.14	0.208	0.26	0.212	0.81	0.488	-0.06	0.477	0.58	0.093
credit-g	0.54	0.133	-0.46	0.184	-0.68	0.523	-0.56	0.526	1.00	0.055
diabetes	0.05	0.455	0	0.447	-0.16	0.499	-0.42	0.406	0	0.232
glass	-0.84	0.296	-1.40	0.320	-0.09	0.439	-1.40	0.537	-0.28	0.235
heart-c	0.79	0.352	-1.58	0.391	-0.73	0.545	0.66	0.676	-2.11	0.221
heart-h	0.14	0.356	-1.29	0.382	-1.43	0.469	-0.14	0.539	-0.14	0.184
heart-statlog	0.07	0.455	0	0.436	-0.44	0.500	-0.89	0.514	0	0.228
hepatitis	-0.26	0.423	0	0.431	0.26	0.500	0.13	0.506	0	0.230
hypothyroid	-0.25	0.348	-0.43	0.344	-0.30	0.574	-0.01	0.473	-0.18	0.236
ionosphere	-0.40	0.425	-0.11	0.444	0.23	0.500	-0.85	0.470	0	0.158
iris	0	0.341	0	0.327	0.53	0.402	-0.13	0.514	-0.67	0.395
kr-vs-kp	-0.26	0.562	-0.78	0.372	0.03	0.501	-0.31	0.560	-0.09	0.173
labor	2.11	0.381	-1.40	0.311	1.05	0.506	-0.70	0.508	-0.35	0.303
lymph	-0.81	0.337	-5.00	0.282	2.84	0.571	-3.78	0.447	-0.95	0.227
primary-tumor	0.71	0.718	-2.36	0.366	-1.18	0.379	-0.47	0.366	-1.47	0.198
sick	-0.09	0.471	0.03	0.504	-0.39	0.420	-0.16	0.643	-0.01	0.169
sonar	-0.96	0.464	0.10	0.440	-0.38	0.500	-0.77	0.579	0	0.223
vehicle	-3.17	1.014	0.87	0.430	-2.06	0.488	-2.72	0.507	-0.69	0.115
vote	-0.18	0.417	0.14	0.422	0.14	0.501	-0.28	0.615	0	0.319
average	-0.18	0.419	-0.82	0.361	-0.05	0.481	-0.49	0.510	0.02	0.204

Table 11: Change of accuracy and relative size of generated model without One-Vs-All classification

8.2 Results without significant preprocessing

Much like the switch to the multi-class classification task, **Table 12** shows only slight changes in accuracy . cAntMinerPB achieves a minor drop in model size, while RIPPER seems almost unaffected and PARTs models even grow significantly. Both PART and Ripper use versions of Information Gain to split continuous attributes, suggesting a different cause of this discrepancy. One simple explanation might be that a continuous attribute can be split more often than a nominal one and therefore feeds into PARTS general tendency to build overly long rule lists.
Curiously cAntMinerPB degrades in accuracy, while all other algorithms show increases. This has the effect that much of the difference in accuracy observed on the binary classification is equalled out and all algorithms perform very similarly.

change in accuracy & relative model size	cAntMinerPB		RIPPER		PART	
	acc	% terms	acc	% terms	acc	% terms
anneal	-0.67	0.823	-0.31	1.112	-0.22	1.055
autos	-2.05	0.638	-0.78	0.917	3.41	1.18
balance-scale	1.70	1.037	1.86	1.292	3.65	2.814
breast-cancer	0	1	0	1	0	1
breast-w	-0.66	0.514	-0.06	1.045	-1.2	1.101
colic	-1.52	1.371	-0.76	0.975	-1.85	1.366
compas	0	1	0	1	0	1
credit-a	-0.06	0.865	-0.17	0.703	-1.01	1.855
credit-g	-0.36	0.853	0.42	0.7	-0.28	1.307
diabetes	-1.3	1.306	0.78	0.815	-1.09	0.778
glass	5.05	0.741	3.46	1.03	6.73	1.569
heart-c	-0.86	0.92	-2.11	0.625	-0.59	1.126
heart-h	-0.54	1.061	1.22	0.688	-1.84	1.648
heart-statlog	-1.41	0.587	-0.74	1.014	-3.04	1.183
hepatitis	-1.42	1.121	-0.9	0.8	-2.97	1.151
hypothyroid	-0.17	0.756	-0.13	0.842	0.24	1.032
ionosphere	-3.19	0.496	-0.46	0.919	-0.46	1.318
iris	-0.13	1.143	-0.93	1.108	1.07	1.067
kr-vs-kp	0	1	0	1	0	1
labor	-4.56	1.062	1.75	0.938	3.51	0.957
lymph	-0.68	0.919	1.08	0.763	1.49	1.046
primary-tumor	0	1	0	1	0	1
sick	0.57	0.793	0.37	1.564	0.38	1.475
sonar	1.35	0.491	3.27	0.955	4.42	0.659
vehicle	0.71	0.113	3.74	1.13	2.53	0.695
vote	0	1	0	1	0	1
average	-0.392	0.87	0.408	0.959	0.495	1.207

Table 12: Change of accuracy and relative size of generated model without static discretization relative to **Table 11**

9 Conclusion

This thesis presented and evaluated a variety of older and state of the art rule learning algorithms. cAntMinerPB and RIPPER appear to be the most solid all around choices in terms of classification accuracy. They perform consistently well across all datasets and require little tuning in order to run with good results. cAntMinerPB has the overall edge in classification accuracy, while RIPPER has a considerable advantage regarding the complexity of learned rule sets. FRL produces good results both regarding accuracy and complexity, being not significantly worse than the leading algorithms all while producing some of the most interpretable rule lists. CORELS offers an approach that does not rely on greedy antecedent selection and therefore gives certain guarantees on the resulting rule list, but does require the user to be much more involved in the parameter selection to produce results that are not sub-par, as does FRL to a lesser degree. AntMiner appears to be a strictly worse version of cAntMinerPB, and only offers a slight improvement in complexity for significantly worse classification performance, highlighting that the improvements in cAntMinerPB seem to be effective. Overall a variety of algorithms are very competitive, which is why it could be interesting for future work to focus more on on the aspects that rule learning already excels at, like interpretability and transparency, over minor gains in classification accuracy.

References

[Angelino et al., 2017] Angelino, E., Larus-Stone, N., Alabi, D., Seltzer, M., and Rudin, C. (2017). Learning certifiably optimal rule lists for categorical data. *Journal of Machine Learning Research*, 18:234:1–234:78.

[Bergstra et al., 2013] Bergstra, J., Yamins, D., and Cox, D. D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 115–123.

[Chen and Rudin, 2018] Chen, C. and Rudin, C. (2018). An optimization approach to learning falling rule lists. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2018*, pages 604–612.

[Clark and Boswell, 1991] Clark, P. and Boswell, R. (1991). Rule induction with CN2: some recent improvements. In *Machine Learning - EWSL-91, European Working Session on Learning, Porto, Portugal, March 6-8, 1991, Proceedings*, pages 151–163.

[Clark and Niblett, 1989] Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3:261–283.

[Cohen, 1995] Cohen, W. W. (1995). Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann.

[Demsar, 2006] Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30.

[Dietterich, 1998] Dietterich, T. G. (1998). Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923.

[Fayyad and Irani, 1993] Fayyad, U. M. and Irani, K. B. (1993). Multi-interval discretization of continuousvalued attributes for classification learning. In *Thirteenth International Joint Conference on Articial Intelligence*, volume 2, pages 1022–1027. Morgan Kaufmann Publishers.

[Frank and Witten, 1998] Frank, E. and Witten, I. H. (1998). Generating accurate rule sets without global optimization. In Shavlik, J., editor, *Fifteenth International Conference on Machine Learning*, pages 144–151. Morgan Kaufmann.

[Fürnkranz, 1999] Fürnkranz, J. (1999). Separate-and-conquer rule learning. *Artif. Intell. Rev.*, 13(1):3–54.

[Fürnkranz and Kliegr, 2015] Fürnkranz, J. and Kliegr, T. (2015). A brief overview of rule learning. In *Rule Technologies: Foundations, Tools, and Applications - 9th International Symposium, RuleML 2015, Berlin, Germany, August 2-5, 2015, Proceedings*, pages 54–69.

[Han et al., 2000] Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12.

[Otero, 2017] Otero, F. (2017). MYRA: A Java Ant Colony Optimization Framework for Classification Algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '17 Companion)*, pages 1247–1254. ACM Press.

[Otero et al., 2008] Otero, F., Freitas, A., and Johnson, C. (2008). cAnt-Miner: an ant colony classification algorithm to cope with continuous attributes. In Dorigo, M., Birattari, M., Blum, C., Clerc, M., Stützle, T., and Winfield, A., editors, *Proceedings of the 6th International Conference on Swarm Intelligence (ANTS 2008), Lecture Notes in Computer Science 5217*, pages 48–59. Springer-Verlag.

[Otero et al., 2013] Otero, F., Freitas, A., and Johnson, C. (2013). A new sequential covering strategy for inducing classification rules with ant colony algorithms. *IEEE Transactions on Evolutionary Computation*, 17(1):64–74.

[Parpinelli et al., 2002] Parpinelli, R., Lopes, H., and Freitas, A. (2002). Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4):321–332.

[Santafé et al., 2015] Santafé, G., Inza, I., and Lozano, J. A. (2015). Dealing with the evaluation of supervised classification algorithms. *Artif. Intell. Rev.*, 44(4):467–508.

[Wang and Rudin, 2015] Wang, F. and Rudin, C. (2015). Falling rule lists. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*.

[Witten et al., 2011] Witten, I. H., Frank, E., and Hall, M. A. (2011). *Data mining: practical machine learning tools and techniques, 3rd Edition*. Morgan Kaufmann, Elsevier.