Optimizing the AUC with Rule Learning

Bachelor-Thesis von Julius Stecher
April 2014

TECHNISCHE
UNIVERSITÄT
DARMSTADT

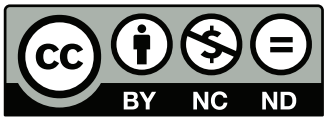
Fachbereich Informatik
Knowledge Engineering

Optimizing the AUC with Rule Learning

Vorgelegte Bachelor-Thesis von Julius Stecher

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Dr. Frederik Janssen

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 14, 2014

(Julius Stecher)

Abstract

In the field of machine learning, one particular research area is *rule learning*, which can be described as modelling some learned experience with the help of conditional rules classifying new unknown data, either in the form of a *decision list* going through an ordered set of rules, applying the first one that fires, or *rule sets* which make use of the entirety of rules learned when classifying a new example. Rule learning as regarded in this thesis is based on a greedy algorithm, which in turn relies heavily on some sort of guidance during the learning process. This guidance is provided by so-called *rule learning heuristics* - functions of certain properties of a rule (which can mostly be derived from coverage and consistency goals) that output a real-valued quality rating of a rule which is used for comparisons with other rule candidates as well as refinements of a particular rule. We will extend the generic *Separate-and-Conquer* [5] algorithm in order to be able to use two different heuristics for rule selection and refinement, and evaluate twelve combinations of heuristics with the means of accuracy and the AUC. The results are partially statistically significant, with the former half of the experiments favoring some heuristics combinations over others. We will conclude with a quick overview of the properties innate to the algorithms evaluated and the theory output they produced.

Contents

1 Introduction

We will start off with a quick introduction to rule learning, followed by an overview of the structure of this thesis.

1.1 Rule Learning: Problem Definition

A general definition of learning is the utilization of previously gathered specific experience to increase performance with respect to future, more general tasks. Extending this to the field of machine learning, we follow the definition by Mitchell [16] stating a task T, a performance measure P and a quantity of experience E with the task T. A program is then said to *learn* if P increases as E does. Note that while a measure P is required for assessing the performance of the learner before and after adding to the experience E, the learning process itself must be guided towards the desired results. Since we assume the learner to be a deterministic machine program, additional measures are required to evaluate the quality of newly gathered experience E in each iteration.

Within this thesis, we will specifically regard a rule learner being trained and evaluated on data available in Attribute- Relation file format (which was developed for the University of Waikato WEKA project [9]). A single instance of attribute-relation formatted data consists of a set of attributes (and possibly associated nominal or numeric values), always including a *class label* attribute determining the *class* the example in question belongs to. Note that without loss of generality, we can reduce the multi-class classifier problem to a binary instance where all labels can be considered as boolean variables; the method used by our classifier will be explained later on. There are more ways to boil down a multiclass problem to one or more singular ones, but these are not relevant for later experiments. In accord with our previous definition, we can then define T as the task of labeling new previously unknown data either *positive* or *negative* and E as the cumulative experience gained by training the classifier on a dataset with known labels. Note that there exist machine learners who regard the set of known pre-labeled data itself as the collected experience (so-called *lazy learners*) like k-nearest-neighbour approaches, which do have the drawbacks of computational load [8] (having to search through the entirety of known labeled data to classify each single testing instance) and necessity [8] of a metric to determine the distance of a point of data to other known points. For other types of rule learners including the separate-and-conquer rule learning algorithms used within later experiments, building up experience E often involves the construction of a more abstract experience model.

In rule learning, this model consists of a set of rules constructed with the help of a pre-classified subset of the cumulative data that can subsequently be applied to new unlabeled examples. To achieve optimal performance, it is of utmost importance not only to measure the performance on the set of pre-labeled training data, but rather to evaluate how well the trained rule learner generalizes to new instances. Since rule learning relies on gathering information from the pre-labeled training data, the method of selecting new rules or adding conditions to existing rules determines the number of rules in a theory, the length of single rules, the *coverage* (meaningful rules should be based on a substantial data foundation) and *consistency* (consistent rules cover only positive examples) of the rules themselves and as such performance on new unknown datasets. Adapting too well to the training data is subject to a number of weaknesses, namely noisy training data (attribute value randomness in a substantial subset of instances) to begin with, which generates unnecessarily complex rules that may generalize insufficiently to new data. This behavior is generally referred to as *overfitting* (on the set of training data).

1.2 Thesis Structure

Starting off with a recap of the base mechanics of machine learning relevant to the SeCo- Framework [15] (currently in development by the Knowledge Engineering Group at TU Darmstadt) which is going to be used as the underlying concrete implementation of the generic *Separate-and-Conquer* algorithm, the optimization method w.r.t the performance measures *cross-validation accuracy per dataset* and *Area under the curve* will be presented. The difference between the standard separate-and-conquer approach and the modified algorithm with is capable of making use of two heuristics for different purposes will be highlighted and we will give a quick overview over existing heuristics and their respective modifications as used in later experiments. The examples chosen for visualization of certain steps are based on a simple and small testing dataset which is not going to be included in the set of datasets used within later experiments. Following the experiments, statistical tests will be conducted and the output of the learning algorithms themselves will be briefly examined.

2 Background

2.1 Terminology

Given a dataset with a fixed number of examples and a rule, the following terminology applies:

- **P**: number of examples with positive label
- **N**: number of examples with negative label
- **p**: number of positive examples predicted positive (*true positives*)
- **n**: number of negative examples predicted positive (*false positives*)
- **P-p**: number of positive examples predicted negative (*false negatives*)
- **N-n**: number of negative examples predicted negative (*true negatives*)

As such, the amount of examples correctly classified amounts to p+N-n (with n+P-p incorrect classifications, respectively). We can display these values in a *confusion matrix* (see table 2.1). It should be noted that all of the heuristics presented require nothing more than the rule's confusion matrix and in some cases the rule predecessor's statistics. The rule learning algorithm used in this thesis belongs to the *Separate-and-Conquer* [5] family of algorithms; a type of rule learner that will be introduced in the following section.

	Classified positive	Classified negative	
+	true positives	false negatives	P
−	false positives	true negatives	N
			P+N

Figure 2.1: A confusion matrix for a binary-class classification problem

2.2 Separate-and-Conquer Rule Learning

In this thesis, we will focus on a subset of rule learning algorithms based on the *covering* or *separate-and-conquer* strategy [5], a type of rule learner adding one rule at a time. This algorithm can be roughly divided into two steps, the *separate-* and the *conquer-* step, where a single rule is first learned (conquer) followed by the removal of any covered examples (separate) [13]. The remaining pieces of data are then used to learn the next rule (return to the conquer step). More specifically, we will regard the *top-down hill-climbing* method of beginning the learning process with the *universal rule* R covering *all* examples, and subsequently adding conditions to this rule based on a *heuristic* h in order to achieve the goals of *consistency* and *coverage*. The former denotes rules that minimize the amount of covered *negative* examples while the latter goal strives to create rules covering as many examples as possible. These goals often have to be traded off against each other. Increasing consistency usually causes a decrease in coverage and vice versa [13].

In contrast to algorithms producing an unordered ruleset, the experience E in our specific implementation is modelled as a *decision list* made up of an ordered list of rules (including the default rule which unconditionally applies the majority class label to any example in the absence of any conditions in the body of the rule). This ordered list is then checked from top to bottom for any evaluated example, applying the class label present in the rule's *head* to the example in question for the first rule in the list with a condition set that is satisfied by that example.

For *top-down hill-climbing* (the method used in our implementation of the *Separate-and-Conquer* Algorithm), we see that by starting off with the universal rule when the learning of a new or additional rule for the theory being built has been deemed appropriate, we begin with maximum coverage (R covering *all* examples, regardless of label) and minimum consistency (R not excluding *any* negative example). By adding conditions to this rule, the amount of covered examples will decrease with each iteration, lowering coverage while increasing consistency. How much consistency is gained (coverage will inevitably decrease while adding conditions to this rule) depends on the choice of conditions added to the rule in each iteration. As such, the top-down hill-climbing learning algorithm will try to find the *best* condition to add with respect to the desired tradeoff between coverage and consistency in each iteration. This choice depends on the *heuristic* function applied to all possible rule refinements (added conditions to a base rule), choosing the refinement that scores best after applying the heuristic to all refinements. It is easy to see that the importance of a good heuristic is vital for learning a theory w.r.t. consistency and coverage as it is the only type of guidance the rule learner can make use of during the training process.

The basic separate-and-conquer algorithm is shown below. The basic algorithm displayed in table 2.2 as well as the subroutine shown in table 2.3 used to find one best rule for inclusion with the theory can be found in the rule learning heuristics article written by Fürnkranz et al. [14] and is included here for the sake of a quick introduction to the rule learner used throughout this thesis, as the difference between the generic algorithm and the modification made to allow later experiments can be highlighted well this way. Note how this algorithm classifies as greedy, e.g. choices made in one iteration are not revisited.

Given the implementation of our *Separate-and-Conquer* main loop (see figure 2.2) and a set of training data with known class labels, we create a new theory T to hold the decision list we aim to create. In our implementation, the main loop will run until no examples with positive class labels are left in the training data set; termination is guaranteed because anytime a rule is added to T, we remove examples covered by this rule from the set of training data and use the remainder the next time we run through the loop. The actual learning process for a single rule is described in detail in the *findBestRule()* subroutine found in figure 2.3, with the main steps being the creation of a set of possible refinements and the subsequent pairwise comparison of the refined base rule with the currently best known rule, again making use of a heuristic function.

```
1 Separate-And-Conquer(TrainingData)
2         Start with empty theory T
3         WHILE (positive examples left in TrainingData)
4                 Rule r = findBestRule(TrainingData)
5                 IF (positiveCovered(r) <= negativeCovered(r) BREAK
6                 ADD r to T
7                 REMOVE all covered examples from TrainingData
8         return T
```

Figure 2.2: Basic separate-and-conquer algorithm as presented in [14]

```
findBestRule(TrainingData)
        Rule rBest = best rule
        bestValue = heuristic value of rBest
        DO
                get possible refinements
                evaluate all refinements
                Rule rRef = best refined rule
                if (heuristic(rRef) >= bestValue)
                        update rBest
        UNTIL (no refinements left)
        return rBest
```

Figure 2.3: Subroutine of the separate-and-conquer algorithm as presented in [14]

2.3 Coverage Space

For our implementation of *Separate-and-Conquer*, it is easy to see that frequent comparisons of both entire rules and different refinements of the same rule are vital for success. We have already mentioned heuristic functions tackling this particular objective (with a more formal definition following later). For the sake of comprehension we will now describe a method of visualizing coverage.

Given the aforementioned values (see figure 2.1) in a dataset of fixed size, Fürnkranz and Flach [6] have created a means of visualization with *coverage space* [6] schematics as shown in 2.4. In order to obtain coverage space, we plot the number of positive examples covered over the number of negative examples covered, resulting in a rectangular plot with the x-axis consiting of the values $\{0, 1, ..., P\}$ (and $\{0, 1, ..., N\}$ for the y-axis). This schematic can then be used to both plot entire theories consisting of an ordered rule list (the decision list) as well as single rules.

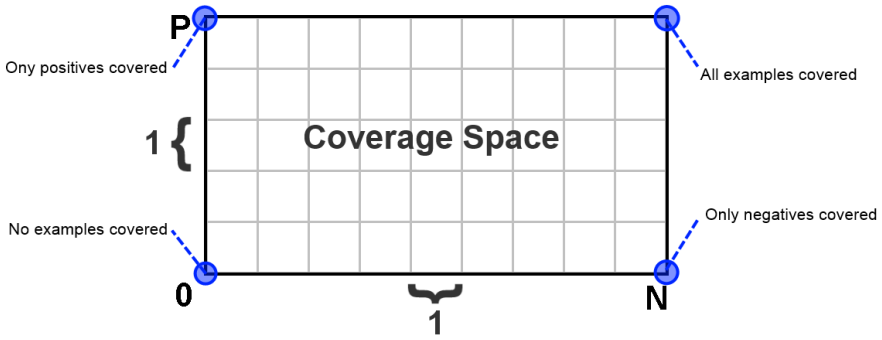


Figure 2.4: Coverage space visualization with P total positive examples and N total negative examples

The following points are of special interest:

- (0,0) is the *empty theory*. It does not cover any examples, neither positive nor negative ones. A *bottom-up* learning algorithm would start at this point and successively add rules.
- (0,P) is the *perfect theory* covering all positive, but no negative examples.
- (N,0) is the *opposite theory* covering all negative, but no positive examples.
- (N,P) is the *universal theory*. It covers all examples regardless of label.

We have already stated that coverage space is useful for both theory and rule plots. Regarding the latter in *top-down hill-climbing*, when attempting to learn a rule, neither p nor n ever increase. This invariant holds because the algorithm started off with the universal rule, maximizing both p and n to achieve values of $p = P$ and $n = N$, and subsequently decreasing coverage by adding conditions (excluding examples can never increase the value of p and n for a specific rule). Adding conditions, as is the case with this particular strategy of rule learning in each iteration, specializes the rule, excluding some examples that were previously covered. Compare this to *covering algorithms* that start off with the empty theory, successively adding rules to cover more examples. Note that the point in time where a rule is considered to be specialized enough (top-down) or generalized enough (bottom-up) can be determined by exchangeable stopping criterions; for all purposes of this paper all experiments will be conducted using a top-down algorithm, stopping the process of rule refining when the amount of true positives taken from the rule's stats is lowered below that of false positives.

ROC Space

Given the coverage space, the ROC (*Receiver Operating Characteristic*) space can be deducted by normalizing the p- and n-axis. The result is a plot of the true positive rate $tpr := \frac{p}{P}$ over the false positive rate $tpr := \frac{n}{N}$. As such, the resolution of the plot changes to $\frac{1}{N}$ for the x-axis (and $\frac{1}{P}$ for the y-axis, respectively). The result is displayed in figure 2.5.

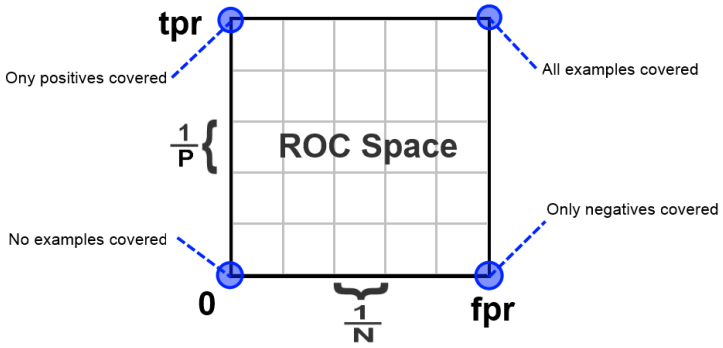


Figure 2.5: ROC space visualization with P total positive examples and N total negative examples

2.4 Rule Learning Heuristics

Any *separate-and-conquer* rule learning algorithm relies on some sort of measure to determine the quality of a rule on-the-fly; this is done with the help of a *heuristic function*. Heuristics can be classified as either *value heuristics* or *gain heuristics*, where the former determine a value (with larger values denoting better rules) to assert a rule and the latter taking into account the amount of information gained compared to a predecessor rule. In this thesis, we will regard the former variant. The heuristics as regarded in this paper disregard misclassification costs, favoring rules that cover as many positive examples as possible (optimizing coverage) while keeping the amount of negative examples covered small (optimizing consistency). In accord with the previous definition, it can thus be concluded that this type of heuristic depends mostly on p (positive examples covered) and n (negative examples covered) [13]. Since for some of the examined heuristics (e.g. the m-Estimate as well as the modifications suggested later) the values of P (total positive examples) and N (total negative examples) must be known as well as the statistics of the predecessor rule (to take into account nested coverage spaces), we generalize our definition of a rule learning heuristic within the scope of this work as follows, analog to the more abstract characterization found in [14]:

h: (p,n,P,N,predecessor(rule)) → val

Note that the rule's predecessor (and thus the predecessor rule's confusion matrix) are not needed for the base heuristics examined in this thesis; however we will require access to this information for the modified variants presented later.

Examined base heuristics

For the experiments we will choose three common base heuristics with slightly different properties. If not otherwise specified, P and N are assumed to be constant. Figures 2.6, 2.7 and 2.8 show a visualization of the *isometrics* of the heuristics. Isometrics are a means of visualizing the properties of a heuristic, with every line consisting of points (n, p) in coverage space with the same value $h(p, n)$ assuming constant values for P and N and the same predecessor rule [6].

- ***Precision***: $h_{prec}(p, n) = \frac{p}{p+n}$ Precision is a simple heuristic with trivial intention: A rule r_1 is preferable to another rule r_2 if r_1 covers a larger *percentage* of positive examples. Note that this does not take into account coverage - a rule covering one positive and zero negative examples will score the highest possible value, while a rule covering all positive and one negative example will score slightly lower. We will see that a theory learned with the help of the precision heuristic is likely to be training-data orientated, with a bad performance when generalizing to new or noisy data. Figure 2.6 shows the isometrics for the precision heuristic.

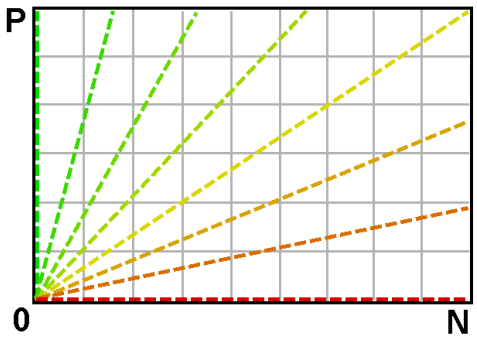


Figure 2.6: Visualization of the precision isometrics (points in coverage space with the same value)

- ***Laplace***: $h_{lap}(p, n) = \frac{p+1}{p+n+2}$ The laplace heuristic reduces some of the overfitting drawbacks (bad generalization) of precision while following the same general intent of maximizing (mostly) consistency. Starting the p and n count at one instead of zero, the origin of the isometrics shifts to $(-1, -1)$. The effects of this change include rules on the P and N axis not sharing the same value anymore. The change in isometrics is visualized above. Applied to the exemplary rules r_1 and r_2 described above, r_1 would not necessarily score better: if both r_1 and r_2 cover zero negative examples, but r_1 covers two positives while r_2 only covers one, the resulting values are $h(r_1) = 0.75$ and $h(r_2) = 0.66$, while evaluating both rules with precision would have yielded $h(r_1) = h(r_2) = 1.0$. Figure 2.7 shows the isometrics for the Laplace heuristic.

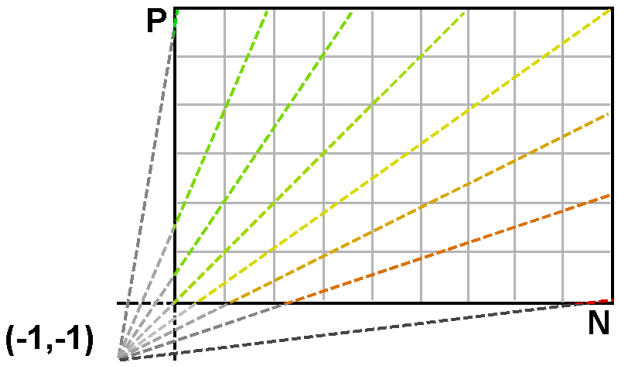


Figure 2.7: Visualization of the laplace isometrics

- ***m-Estimate***: $h_{mest}(p,n) = \frac{p+m\cdot\frac{P}{P+N}}{p+n+m}$ The m-estimate belongs to the family of configurable heuristics. The additional parameter m can be used to alter the isometrics. It should be noted that in the special case $m = 0$, the m-Estimate equals precision, and with $m \to \infty$, the lines connecting points with the same value become parallel with a slope of $\frac{P}{P+N}$ (the apriori distribution), causing the isometrics to equal that of *weighted relative accuracy* ($h_{WRA} = \frac{p+n}{P+N}(\frac{p}{p+n} - \frac{P}{P+N})$, which equals $\frac{p}{P} - \frac{n}{N}$ for constant values of P and N). The optimal value for the parameter m has been empirically determined by Fürnkranz et al. [14] to be $m = 22.466$ and will be used as such within the later experiments. Figure 2.8 shows the isometrics for the m-Estimate heuristic.

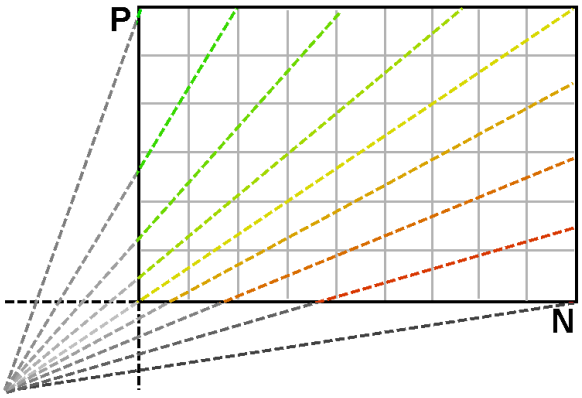


Figure 2.8: Visualization of the m-estimate isometrics

2.5 Metrics used for validation performance

We have already highlighted the problem of classifiers overfitting to the training data provided. There are several ways to use dedicated testing data; a trivial approach would be to split a dataset into two disjunct parts and train the classifier on the first part before validating it making use of the second part. However, this is still prone to bias and requires a very large amount of data to be moderately effective. In order to make the most of a single dataset provided, we will be using cross-validation throughout this thesis. To execute *k-fold cross-validation*, the dataset is split into k disjunct *folds*. Every fold will serve as the training data once, with the remaining k-1 folds being used for training the classifier beforehand. The results of all k validations is then averaged to obtain a performance estimate which is less prone to bias. In this thesis we will make use of two methods to get a performance estimate from testing a trained classifier (which happens k times during our k-fold cross-validation).

Accuracy per Dataset

A means of validation is to calculate the percentage of correctly classified instances in the testing data. This value is more straightforward than the AUC and as we will see later, less prone to error on small datasets evaluated with the help of cross-validation in the case of an algorithm with *decision list*-type theory output. The baseline implementation of the separate-and-conquer algorithm in the SeCo- Framework already offers the ability to calcuate this value.

Area under the curve

Recall that a visualization of coverage space is a rectangular plot where both the x- and y-axis share a resolution of 1 as shown in figure 2.4. We have seen that this representation can be normalized to feature *axis lengths* of 1 (see figure 2.5), changing the resolution of the x-axis to 1/N and of the y-axis to 1/P. As such, the axis labels change from N and P to the *false positive rate* and *true positive rate*, respectively. Since we are using *decision lists* as output of our classifiers, the approach to plot a ROC curve (and thus obtain the area underneath) is as follows (figure 2.9):

- Group examples covered by the first rule in the decision list and plot coverage in ROC space.
- Remove examples covered by the first rule (because we are using a decision list classifier), then proceed with the next rule until reaching the default rule covering all leftover examples.
- Calculate the AUC based on the points in ROC space determined in the above steps.

Figure 2.9: Calculation of the AUC in our binary (non-scoring) classifier

Given an example ROC plot, we can easily visually identify the quality of class separation with the help of the ROC. Less concavities correspond to better class separation and vice versa. As such, we can measure the quality by regarding the *area under the ROC curve* (referred to as AUC). Figure 2.10 shows an example using a top-down classifier that produced an unspecified decision list with two rules followed by the default rule after being trained on some data. The colored area under the curve represents the AUC. If the values plotted below were obtained from a real classifier, the concave shape would denote performance above mere guessing. In order to be able to calculate the AUC, modifications of the SeCo-implementation were necessary to keep track of required data. Figure 2.10 (right) shows an example ROC plot with corresponding AUC based on a decision list plotted in coverage space (left) with $N = 10$ and $P = 5$. Out of these, rule 1 covers 3 positive and 2 negative examples. Then the covered examples are removed; rule 2 subsequently covers 1 positive and 2 negative examples out of the 10 left examples.

Finally, again after removing the examples newly covered, the default rule covers the leftover 1 positive and 6 negative examples. To obtain the ROC (figure 2.10, right) we normalize the coverage space. The AUC can then be calculated as the *area under the curve*, e.g. the integral.

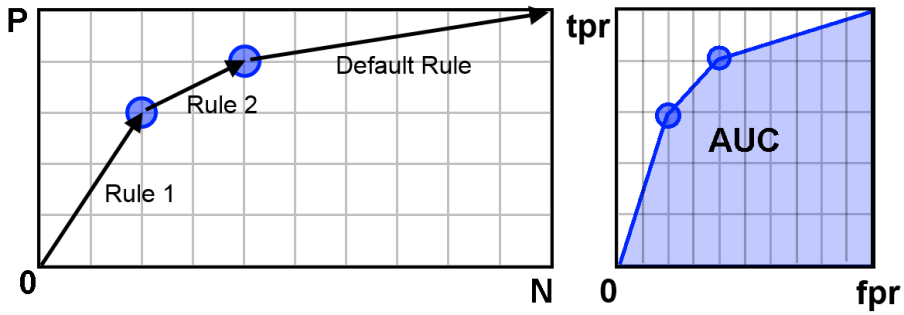


Figure 2.10: From decision lists to the Area under the curve

3 Derived method

The traditional generic Separate-and-Conquer rule learning algorithm uses exactly one measure to both develop new rules from a base rule by adding conditions as well as select the best rule at the end of each conquer step to add to the theory. We will first explain the learning of a theory on a simple example dataset in detail, followed by an explanation of the intent of the optimization approach as well as a step-by-step comparison, making use of *weather.nominal* dataset provided by the UCI repository [1].

3.1 Baseline implementation: an example

The data contained in the example dataset outlined in table 3.1 describes a set of weather conditions (temperature, humidity and outlook) as well as a binary class label determining whether a fictional game will take place under these conditions or not. It was chosen for our example because of the relative simplicity of the theory learned and the steps taken to aquire such a theory.

Example	**Outlook**	**Temperature**	**Humidity**	**Windy**?	Play?
p_1	sunny	hot	high	false	NO
p_2	sunny	hot	high	true	NO
p_3	rainy	cool	normal	TRUE	NO
p_4	sunny	mild	high	FALSE	NO
p_5	rainy	mild	high	TRUE	NO
n_1	overcast	hot	high	FALSE	YES
n_2	rainy	mild	high	FALSE	YES
n_3	rainy	cool	normal	FALSE	YES
n_4	overcast	cool	normal	TRUE	YES
n_5	sunny	cool	normal	FALSE	YES
n_6	rainy	mild	normal	FALSE	YES
n_7	sunny	mild	normal	TRUE	YES
n_8	overcast	mild	high	TRUE	YES
n_9	overcast	hot	normal	FALSE	YES

Table 3.1: Overview of the 14 instances of data. Possible class values are yes and no (right column)

We will now train a classifier on the entire dataset. Since we are using an algorithm that produces a decision list defaulting to the majority class, the default rule is play = yes. All other rules will predict play = no, and if none of these does trigger, the default rule will predict the majority class (yes). As such, the values for P and N in the above dataset are 5 and 9, respectively. Rules will be denoted by the format $head$:- $condition_1$, $condition_2$, ..., $condition_n$ with $head$ denoting the predicted class label for an instance that matches all attribute conditions. The heuristic used for both selecting the optimal refinement and comparing whole rules is h_{prec}.
As our algorithm is a top-down learner, we start off with an empty theory (table 2.2, line 2) and begin to refine the base rule $play = no$. To achieve this, we evaluate all possible refinements for this rule with the help of the subroutine shown in figure 2.3. A top-down rule refinement is one condition added to the body of the rule. Given our base rule $play = no$, the refined rule candidates and their corresponding heuristic values as calculated with precision are shown in figure 3.2. r_1 is kept as refinement because it scored the highest value among refinements and thus always won the comparison (figure 2.3, line 8).

r_i	**Refined rule** head :- body	**Coverage** (p, n)	**Heuristic** h_{prec}
r_1	play = no :- **outlook = sunny**	(3,2)	0.6
r_2	play = no :- outlook = rainy	(2,3)	0.4
r_3	play = no :- temperature = hot	(2,2)	0.5
r_4	play = no :- temperature = mild	(2,4)	0.33
r_5	play = no :- temperature = cool	(1,3)	0.25
r_6	play = no :- humidity = high	(4,3)	0.57
r_7	play = no :- humidity = normal	(1,6)	0.14
r_8	play = no :- windy = TRUE	(3,3)	0.5
r_9	play = no :- windy = FALSE	(2,6)	0.25

Table 3.2: Coverage of possible refinements of the rule play = no.

Note that the refinement *outlook = overcast* is excluded as there exists no positive example meeting this condition. The set of refinements and their evaluation results are shown in figure 3.1.

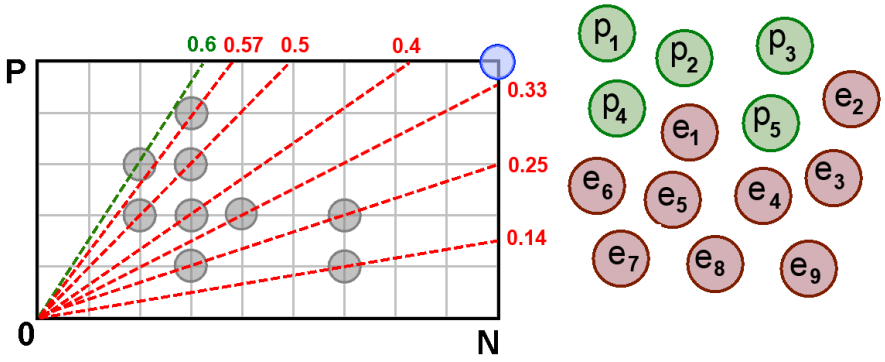


Figure 3.1: Refinements of the rule play = no in coverage space

The best refinement as chosen by precision is the condition *outlook = sunny*, which will thus be added to the body of the base rule. Our new base rule is play = no :- outlook = sunny. We now have to rate our rule to compare it with the previous *best* rule found. Since we use the same heuristic for rating the rule as we have used for determining the best refinement, the value of our rule is $h_{prec} = 0.6$, as it consists of the added condition only. Since it scores higher than our previous best rule (which labeled *all* examples as positive), our new best rule is *play = no :- outlook = sunny* as displayed in figure 3.2.

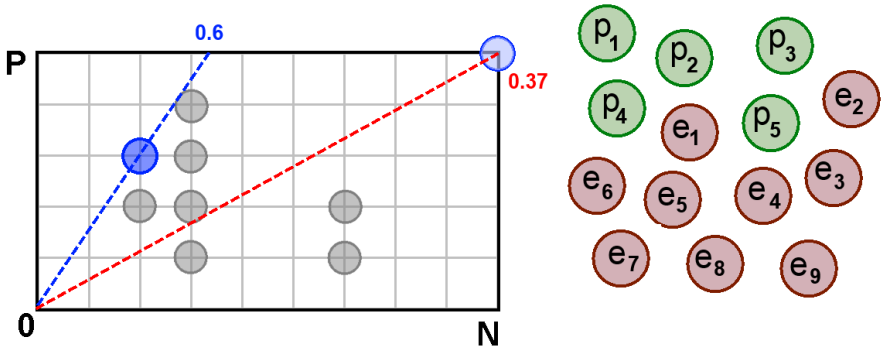


Figure 3.2: Comparison of previous best and newly refined rule

We will now attempt to refine this rule as we still have options (figure 2.3, line 10), starting with making a list of applicable refinements again (figure 2.3, line 5; table 3.3). The plot is shown in figure 3.3.

r_i	**Refined rule** head :- body	**Coverage** (p, n)	**Heuristic** h_{prec}
r_{10}	play = no :- outlook = sunny, temperature = hot	(2,0)	1.0
r_{11}	play = no :- outlook = sunny, temperature = mild	(1,1)	0.5
r_{12}	play = no :- outlook = sunny, **humidity = high**	(3,0)	1.0
r_{13}	play = no :- outlook = sunny, windy = TRUE	(2,1)	0.5
r_{14}	play = no :- outlook = sunny, windy = FALSE	(2,1)	0.66

Table 3.3: Coverage of possible refinements of the rule play = no :- outlook = sunny.

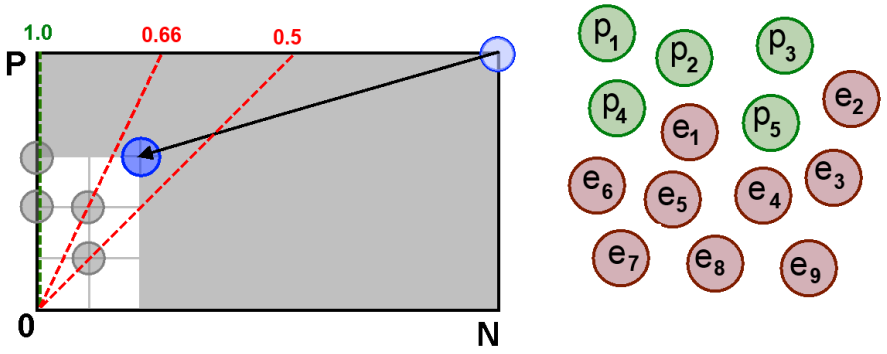


Figure 3.3: Refinements of the rule play = no :- outlook = sunny in coverage space

Note that there are two refinements that score the same value. In this case the refinement selection depends on the implementation; here we select *humidity = high* because of a larger value for p and thus have a new best rule *play = no :- outlook = sunny, humidity = high* scoring a value of 1.0 with three covered positive examples and no covered negative (compared to our previous best rule rated 0.6). We now stop since this rule cannot be improved anymore by refinements (as neither p or n ever increase, as we have established, and n is already zero, we could only decrease the value of p), and add it to the theory (figure 2.2, line 6). We are now finished with the first *conquer* step (figure 3.4). Following the *separate* step, the covered examples are removed (figure 2.2, line 7) and since there are still examples left, we start off with the base rule play := no again, attempting to learn a second rule for our theory (*conquer* step again), this time only regarding the examples not covered by the first rule we have chosen to be part of our theory.

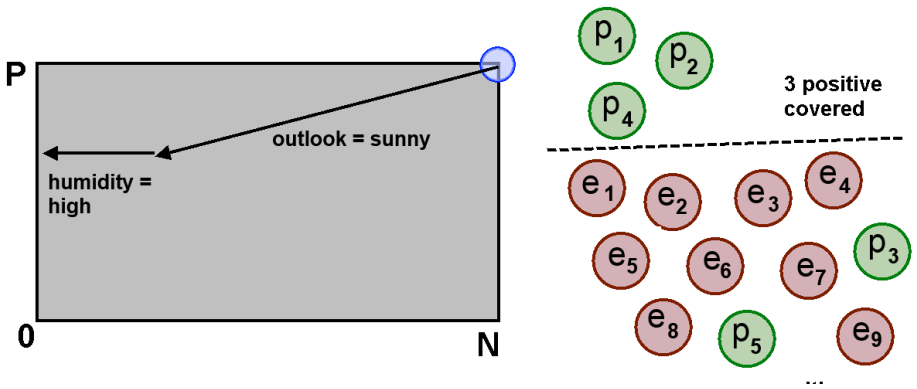


Figure 3.4: A rule covering three positive and zero negative examples has been learned

3.2 Optimization via modified heuristics for rule refinement

In the standard *Separate-and-Conquer* implementation, we use the same heuristic function everytime we want to evaluate an entire rule or a refinement of a rule to determine the current best rule and the best refinement w.r.t. the goals of the heuristic (usually coverage and consistency). The approach highlighted in this paper modifies this standard algorithm to use different heuristics for *rule selection* and *rule refinement*. We will again use the three value heuristics described above for *rule selection*, but use modified variants of these heuristics for *rule refinement*. Since the goal of this paper is to optimize the AUC, the choice of these secondary heuristics must reflect that.

Note that the three base heuristics (h_{prec}, h_{lap} and h_{mest}) all share similar isometrics, with the only difference being the origin (in the latter case, the location of the origin can be configured via the parameter m). We will want to preserve this attribute, but shift the origin to the top right corner of the coverage space. The intend of this is that in our case the *rule refiner* follows the top-down strategy (starting off with the most general rule and successively adding conditions). We have to take into account that the values of P and N are not constant this time w.r.t. the heuristic function, but depend on the predecessor of the rule. This is because for our approach to work, we will want the origin of the isometrics to be placed at the point in coverage space corresponding to the base rule we want to refine, which will produce *nested* coverage spaces, and subsequently evaluate the refinements within the base rule's nested coverage space. Figure 3.5 visualizes the intend of geometrically optimizing the area under the curve during the *rule refinement* stage.

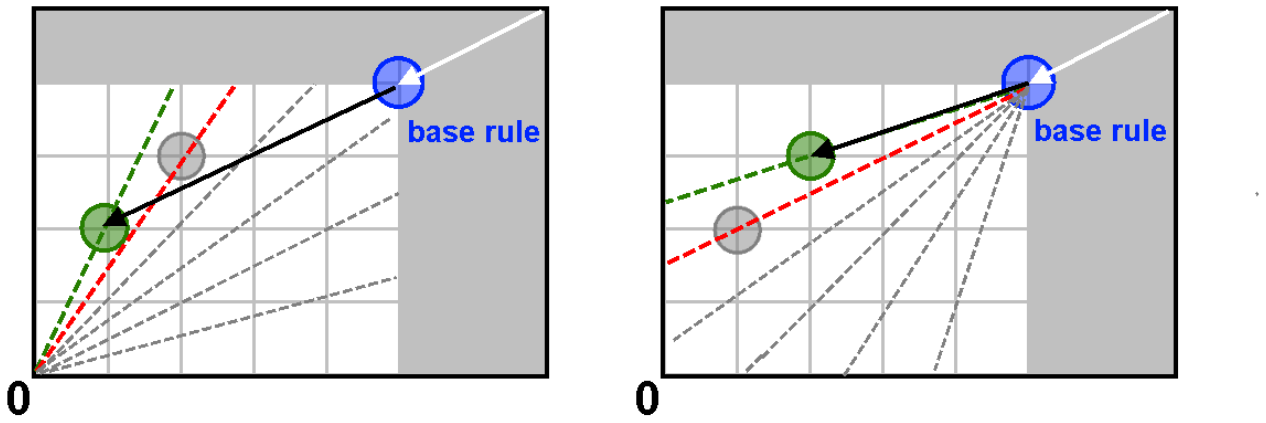


Figure 3.5: Isometrics of the rule refinement heuristic evaluation process

It becomes apparent that if the top-down refinement of a rule is driven by the modified heuristic (isometrics visualized in blue) and the choice of rules for our theory depends on the original heuristic (isometrics visualized in red), the resulting ROC should be of a more convex shape given our geometric optimization approach. We will attempt to validate this claim with the help of the later experiments. The modified heuristics regarded here are as follows, with one variant for each base heuristic.

- ***Precision'***: $h'_{prec}(p,n,P,N) = \frac{N-n}{(P+N)-(p+n)}$

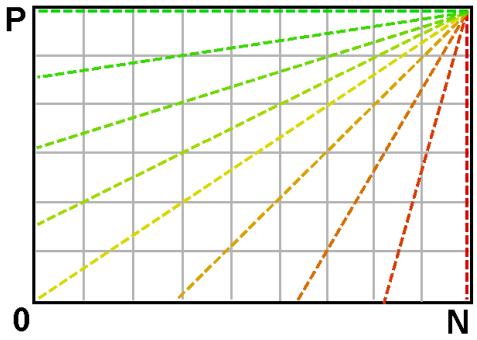


Figure 3.6: Visualization of the modified precision isometrics (points in coverage space with the same value)

- ***Laplace'***: $h'_{lap}(p,n,P,N) = \frac{N-n+1}{(P+N)-(p+n-2)}$

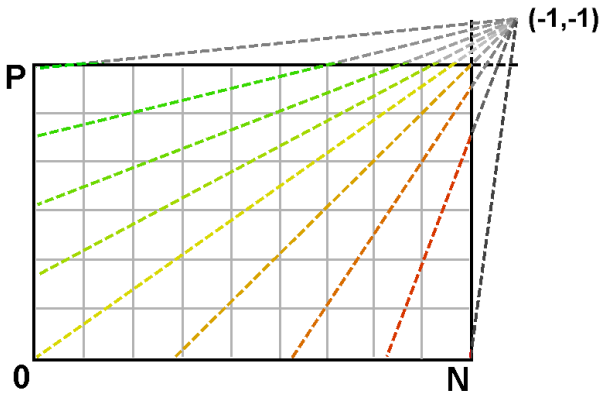


Figure 3.7: Visualization of the modified laplace isometrics

- ***m-Estimate'***: $h'_{mest}(p,n,P,N) = \frac{N-n+m\cdot\frac{P}{P+N}}{(P+N)-(p+n-m)}$

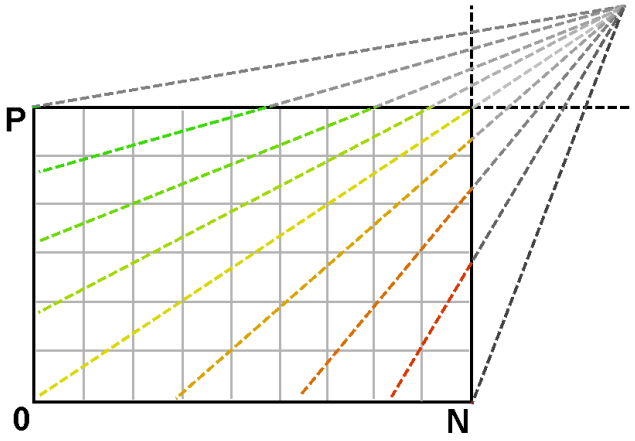


Figure 3.8: Visualization of the modified m-estimate isometrics

While in theory the modified variants could be used for rule selection too, it is easy to see that this would not lead to the desired results, especially with h'_{prec}, since all rules covering all positive examples share the same (maximal) heuristic value of 1.0, even if covering different amounts of negative examples. In the case of h'_{prec} and to a lesser extend h'_{lap} being used for *rule selection*, while rules with high coverage are still preferred by these heuristics, the rule learning process is not steered towards a consistent theory very well, especially in the case of h'_{prec}, which would assign the maximum score to any rule that covers only positive examples within the nested coverage space (examples already covered not taken into account) while completely disregarding consistency. In preceding experiments training a classifier with modified heuristics being used for *rule selection as well as rule refinement*, the resulting classifiers were sometimes unable to label *any* new testing example correctly. This first approach has as such been dismissed early on.. Note that h'_{prec} is even undefined for p=P and n=N as this would lead to a division by zero. An interesting property of h_{mest} and h'_{mest} is that with m increasing, their isometrics converge towards those of a cost metric taking into account the apriori distribution of examples. With our parameter setting of *m=22.446*, the difference between h_{mest} and h'_{mest} is rather marginal in practice. Recall the *findBestRule*-subroutine of the generic *Separate-and-Conquer* algorithm (figure 2.3). In order to apply the modified heuristics to rule *refinements* specifically, we have to alter the subroutine as shown in figure 3.9; the updated parts are lines 3, 7 and 8.

```
1  findBestRule(TrainingData)
2          Rule rBest = best rule
3          bestValue = selection_heuristic(rBest)
4          DO
5                  get possible refinements
6                  evaluate all refinements
7                  Rule rRef = best refimenent w.r.t. refinement_heuristic(rRef)
8                  if (selection_heuristic(rRef) >= bestValue)
9                          update rBest
10         UNTIL (no refinements left)
11 return rBest
```

Figure 3.9: Altered subroutine of the separate-and-conquer generic algorithm as used in later experiments

Analogous to the previous example, we will again describe the rule learning and refinement process for one step making use of the *weather.nominal* UCI [1] dataset (see table 3.1. For this we will use $h_{prec} = \frac{p}{p+n}$ as the rule selection heuristic again, but this time evaluate the refinements with the help of $h'_{prec} = \frac{N-n}{(P+N)-(p+n)}$.

Again we start off with the universal rule *play = no* and look at the possible refinements. The value of this universal rule according to the rule selection heuristic is $h_{prec} = \frac{p}{p+n} = 0.357$. Note that for the following refinements (table 3.4), the values are calculated by the *modified* precision variant.

r_i	**Refined rule** head :- body	**Coverage** (p, n)	**Heuristic** h'_{prec}
r_1	play = no :- outlook = sunny	(3,2)	0.77
r_2	play = no :- outlook = rainy	(2,3)	0.66
r_3	play = no :- temperature = hot	(2,2)	0.7
r_4	play = no :- temperature = mild	(2,4)	0.625
r_5	play = no :- temperature = cool	(1,3)	0.6
r_6	play = no :- **humidity = high**	(4,3)	0.857
r_7	play = no :- humidity = normal	(1,6)	0.428
r_8	play = no :- windy = TRUE	(3,3)	0.75
r_9	play = no :- windy = FALSE	(2,6)	0.5

Table 3.4: Coverage of possible refinements of the rule play = no.

Even though our example was a simple one, the new approach has taken a different path at this point already. Instead of refining the base rule with the condition *outlook = sunny*, the refinement *humidity = high* now scores higher. Figure 3.10 shows the chosen refinement and it's position in coverage space w.r.t the base rule.

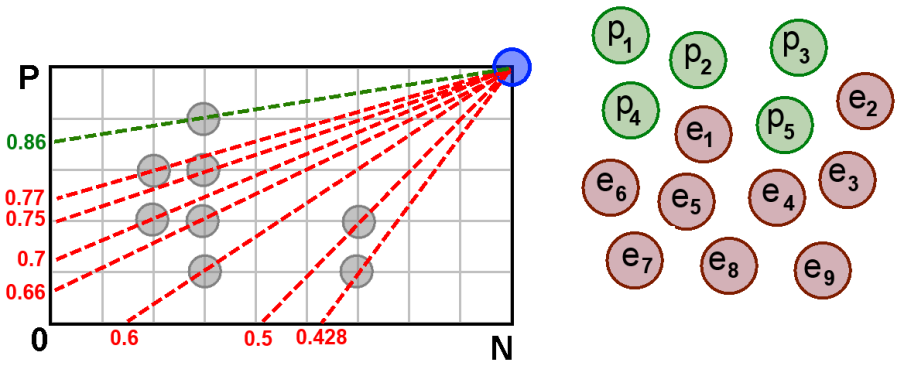


Figure 3.10: Refinements of the rule play = no with the modified *rule refinement* heuristic

Recall that the value of the base rule w.r.t the *rule selection* heuristic (ordinary precision) was 0.357. We now calculate the value of the rule obtained via refining the base rule with the condition that scored best w.r.t. the *rule refinement* heuristic with the help of the *rule selection* heuristic, obtaining the value $h_{prec} = 0,571$ for the rule *play = no :- humidity = high* (which covers 4 positive and 3 negative examples), which thus is an improvement of the base rule according to h_{prec}.

Again removing the examples covered by this rule (recall that our algorithm produces an ordered decision list), we proceed to evaluate the possible refinements of *play = no :- humidity = high*.

r_i	**Refined rule** head :- body	**Coverage** (p, n)	**Heuristic** h'_{prec}
r_{10}	play = no :- humidity = high, **outlook = sunny**	(3,0)	0.75
r_{11}	play = no :- humidity = high, outlook = rainy	(1,1)	0.4
r_{12}	play = no :- humidity = high, temperature = hot	(2,1)	0.5
r_{13}	play = no :- humidity = high, temperature = mild	(2,2)	0.33
r_{14}	play = no :- humidity = high, windy = TRUE	(2,1)	0.5
r_{15}	play = no :- humidity = high, windy = FALSE	(2,2)	0.33

Table 3.5: Coverage of possible refinements of the rule play = no :- humidity = high.

Given these values, the refinement outlook = sunny is chosen by the *rule refinement* heuristic. Figure 3.11 visualizes the nested coverage space including the isometrics of the h'_{prec} heuristic as well as the possible refinements.

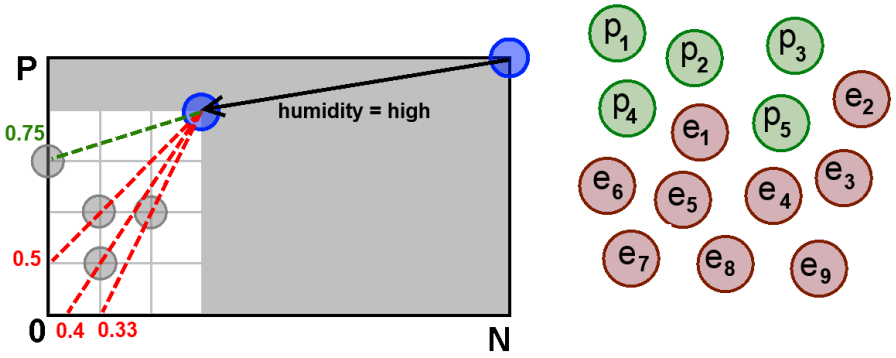


Figure 3.11: Refinements of the rule play = no :- humidity = high with the modified *rule refinement* heuristic

At this point we would stop learning the rule again, add it to the theory and attempt to learn a new rule on the leftover data. Figure 3.12 gives a quick overview over the difference in the learning process of a single rule in our example. The assumption is that a theory consisting of rules refined via the modified approach will reflect in the performance of the resulting classifier measured by accuracy and the AUC in a positive way. The goal of the optimization method is to learn different rules (possibly in a different order in the decision list) in order to achieve a better classifier.

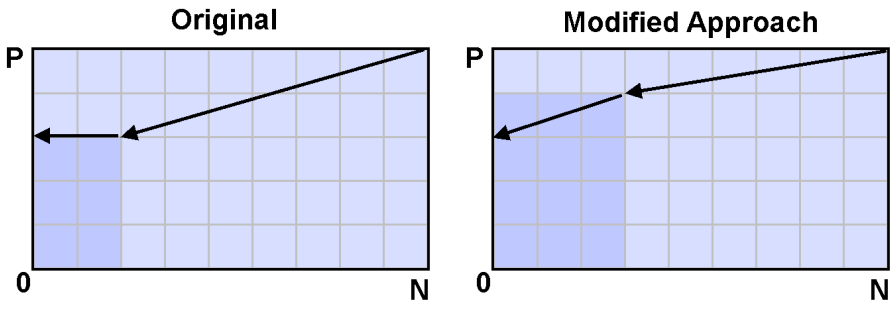


Figure 3.12: Comparison of the learning process of a single rule in the previous example.

4 Experiments

Note that for all of the experiments conducted below, no additional procedures (like pruning) to avoid overfitting on the training data have been used following the same strategy as Fürnkranz and Janssen [14], focusing on the heuristics used to achieve the goal of a classifier that generalizes well to new data.

4.1 Examined algorithms

For the following experiments, the Separate-and-Conquer algorithm implementation remains constant, with the only changes being made to the heuristics used for rule selection and refinement. We can thus denote an algorithm by a pair $(h_{selection}, h_{refinement})$. The following twelve combinations will be tested and compared:

- (h_{prec}, h_{prec})
  (standard algorithm using the precision heuristic for all purposes)
- (h_{lap}, h_{lap})
  (standard algorithm using the Laplace heuristic for all purposes)
- (h_{mest}, h_{mest})
  (standard algorithm using the m-Estimate heuristic for all purposes)
- $(h_{prec}, h'_{prec}), (h_{prec}, h'_{lap})$ and (h_{prec}, h'_{mest})
  (modified algorithm with precision-based rule selection and a new refinement heuristic)
- $(h_{lap}, h'_{prec}), (h_{lap}, h'_{lap})$ and (h_{lap}, h'_{mest})
  (modified algorithm with laplace-based rule selection and a new refinement heuristic)
- $(h_{mest}, h'_{prec}), (h_{mest}, h'_{lap})$ and (h_{mest}, h'_{mest})
  (modified algorithm with m-Estimate-based rule selection and a new refinement heuristic)

4.2 Examined datasets

We will evaluate the twelve combinations listed above on 20 binary- and multiclass datasets by the means of *average accuracy*. The evalutation method is ten-fold cross validation to reduce bias and increase the quality of the resulting performance estimate.

4.2.1 Arbitrary number of classes

The following table gives a quick overview of the datasets used in the first part of the experiments and their most important properties.

Dataset	Classes	Instances	Attributes	Notes
breast-cancer	2	286	10	-
car	4	1728	7	-
contact-lenses	3	24	5	-
futebol	2	14	5	-
glass	7	214	10	-
hepatitis	2	155	20	-
hypothyroid	2	3163	26	-
horse-colic	2	368	23	-
idh	3	29	5	-
iris	3	150	5	-
ionosphere	2	351	35	-
labor	2	57	17	-
lymphography	4	148	19	-
mushroom	2	8124	23	low noise
monk3	2	122	7	-
primary-tumor	22	339	18	large number of classes
soybean	19	683	36	large number of classes
tic-tac-toe	2	958	10	-
vote	2	435	17	-
zoo	7	101	18	-

Table 4.1: Overview of the binary- and multiclass datasets used in the comparisons based on average accuracy. Attribute counts include the class label.

4.2.2 Binary-class datasets

We will now repeat the same experiment on 9 binary-class datasets by the means of *Area-under-the-Curve*, as this is the performance measure we are primarily interested in optimizing. We have limited the amount of classes to two for simplicity, because the baseline implementation did not provide support for calculating the AUC and the implemented solution shown in figure 2.9 assumes a class count of two. Table 4.2 again gives a quick overview of the datasets used in the second part of the experiments and their most important properties.

Dataset	Classes	Instances	Attributes	Notes
breast-cancer	2	286	10	-
hepatitis	2	155	20	-
horse-colic	2	368	23	-
kr-vs-kp	2	3196	37	-
monk3	2	122	7	-
tic-tac-toe	2	958	10	-
vote	2	435	17	-

Table 4.2: Overview of the binary-class datasets used in the comparisons based on the AUC. Attribute counts include the class label.

Recall the method of calculating the AUC, slightly altered to accommodate ten-pass ten-fold cross-validation, shown in figure 4.1.

- Group examples covered by the first rule in the decision list and plot coverage in ROC space.
- Remove examples covered by the first rule (because we are using a decision list classifier), then proceed with the next rule until reaching the default rule covering all leftover examples.
- Calculate the AUC for the current fold based on above ROC plot.
- Average the per-fold AUC results to obtain an AUC estimate for one pass of a ten-fold cross-validation.

Figure 4.1: Approach to calculate the AUC after cross-validation

Note that for the second experiment 10x10 cross-validation will be used (repeating the above procedure for a total of 10 passes and again averaging the results). Because of the comparably small size of the binary class datasets in question, a lack of data in the respective testing fold may cause significant bias; this problem is reduced considerably by increasing the number of passes. It should also be noted that the algorithm implementation does not permit a more reliable way of calculating the AUC, given that for any two examples e_1 and e_2, if both e_1 and e_2 are covered by a rule r, the algorithm will assign the same confidence value to both e_1 and e_2, limiting the number of points in the ROC plot to the number of rules in the decision list (including the default rule) plus one (as the point $(0, 0)$ is always included).

4.3 Comparison of average accuracies (19 datasets)

The results of the experiments on the first set of datasets can be found in tables 4.3 and 4.4. Note that the dataset *mushroom* is an useful measure to check basic functionality of modified algorithms as the data is easy to learn. For the sake of easier readability, the resulting table has been split into two successive tables. Note that for calculating the ranks of the respective algorithms later, both tables are being taken into account.

Dataset	h_{prec}, h_{prec}	h_{prec}, h'_{prec}	h_{prec}, h'_{lap}	h_{prec}, h'_{mest}	h_{lap}, h_{lap}	h_{lap}, h'_{prec}
breast-cancer	68.53	72.38	72.03	73.43	69.58	70.63
car	90.10	90.34	90.51	88.66	90.45	91.20
contact-lenses	79.17	87.50	87.50	83.33	79.17	87.50
futebol	28.57	64.29	57.14	42.88	28.57	64.29
glass	56.54	65.89	68.69	62.15	61.22	65.89
hepatitis	78.07	79.36	80.00	76.77	78.71	79.36
hypothyroid	98.23	98.61	98.74	98.83	98.39	98.61
horse-colic	72.01	79.35	79.35	77.99	70.65	79.35
idh	62.07	82.76	75.86	75.86	62.07	82.76
iris	92.67	93.33	95.33	94.67	94.00	93.33
ionosphere	95.16	82.62	83.19	89.46	94.87	82.62
labor	91.23	80.70	82.46	89.47	91.23	80.70
lymphography	83.78	77.70	84.46	83.11	85.14	77.70
mushroom	100.0	100.0	100.0	100.0	100.0	100.0
monk3	87.71	82.79	82.79	84.43	88.53	85.25
primary-tumor	33.63	39.23	35.10	30.97	32.45	39.23
soybean	90.04	91.51	92.24	91.36	90.34	91.80
tic-tac-toe	97.39	98.02	97.60	97.81	97.60	98.02
vote	94.94	93.56	94.25	94.48	95.40	94.25
zoo	84.16	88.12	92.08	90.01	86.14	88.12

Table 4.3: Average accuracies obtained via ten-fold cross-validation on 20 datasets

Dataset	h_{lap}, h'_{lap}	h_{lap}, h'_{mest}	h_{mest}, h_{mest}	h_{mest}, h'_{prec}	h_{mest}, h'_{lap}	h_{mest}, h'_{mest}
breast-cancer	71.33	72.73	71.33	72.03	72.38	73.78
car	91.73	91.20	89.64	90.45	90.28	87.91
contact-lenses	87.50	83.33	87.50	87.50	87.50	83.33
futebol	57.14	42.88	50.00	64.29	57.14	42.86
glass	68.69	62.15	69.16	67.29	71.50	63.55
hepatitis	80.00	76.74	78.07	79.36	80.00	76.77
hypothyroid	98.74	98.83	98.80	98.61	98.74	98.83
horse-colic	80.16	77.99	77.45	79.35	78.80	77.99
idh	75.86	75.86	68.97	82.76	75.86	75.86
iris	95.33	94.67	94.00	93.33	95.33	94.67
ionosphere	93.19	89.46	91.74	82.91	83.19	91.17
labor	82.46	89.47	85.97	80.70	82.46	89.47
lymphography	84.46	83.11	75.00	76.35	81.08	83.78
mushroom	100.0	100.0	100.0	100.0	100.0	100.0
monk3	84.43	86.89	81.15	79.51	81.15	82.79
primary-tumor	35.99	30.38	33.92	37.76	34.51	30.68
soybean	92.39	90.63	91.51	90.92	90.48	91.36
tic-tac-toe	97.60	97.91	98.12	98.02	97.60	97.81
vote	94.25	94.94	93.33	93.56	94.71	96.09
zoo	92.08	90.10	89.11	88.12	92.08	90.10

Table 4.4: (cont'd.) Average accuracies obtained via ten-fold cross-validation on 20 datasets

Looking at tables 4.3 and 4.4, we notice that the algorithm (h_{lap}, h'_{lap}) outperforms other combinations on a whole 7 datasets (namely *car, contact-lenses, hepatitis, horse-colic, iris, soybean* and *zoo*). As such, this combination in particular becomes interesting for further validation. We will later conduct statistical tests to try and prove the assumption that the combination (h_{lap}, h'_{lap}) is superior w.r.t. accuracy.

Algorithm	Average rank
(h_{prec}, h_{prec})	8.82
(h_{prec}, h'_{prec})	6.18
(h_{prec}, h'_{lap})	5.21
(h_{prec}, h'_{mest})	6.76
(h_{lap}, h_{lap})	7.76
(h_{lap}, h'_{prec})	6.18
(h_{lap}, h'_{lap})	4.37
(h_{lap}, h'_{mest})	6.37
(h_{mest}, h_{mest})	7.13
(h_{mest}, h'_{prec})	6.76
(h_{mest}, h'_{lap})	5.92
(h_{mest}, h'_{mest})	6.53

Table 4.5: Average ranks for all twelve algorithms based on above experiments

Table 4.5 shows the average ranks for the twelve combinations taking into account 19 datasets (*mushroom* is omitted). We will need these values for further testing later. Our most promising candidate (h_{lap}, h'_{lap}) scored a notable rank of 4.37 (the average expected rank for any algorithm would be 6.5).

4.4 Comparison of the AUC (7 binary-class datasets)

The results of the experiments on the second set of datasets can be found in tables 4.6 and 4.7. Note that as previously mentioned, 10x10 cross-validation has been used as a method for evaluation.

Dataset	h_{prec}, h_{prec}	h_{prec}, h'_{prec}	h_{prec}, h'_{lap}	h_{prec}, h'_{mest}	h_{lap}, h_{lap}	h_{lap}, h'_{prec}
breast-cancer	0,605	0,617	0,626	0,639	0,601	0,617
hepatitis	0,685	0,670	0,668	0,639	0,704	0,670
tic-tac-toe	0,981	0,980	0,982	0,976	0,978	0,980
vote	0,949	0,937	0,938	0,948	0,955	0,941
horse-colic	0,747	0,782	0,783	0,796	0,737	0,782
monk3	0,886	0,847	0,850	0,862	0,893	0,846
kr-vs-kp	0,995	0,990	0,993	0,993	0,996	0,989

Table 4.6: AUC obtained via ten-pass ten-fold cross-validation on 7 datasets

Dataset	h_{lap}, h'_{lap}	h_{lap}, h'_{mest}	h_{mest}, h_{mest}	h_{mest}, h'_{prec}	h_{mest}, h'_{lap}	h_{mest}, h'_{mest}
breast-cancer	0,619	0,634	0,606	0,611	0,620	0,635
hepatitis	0,668	0,639	0,685	0,670	0,668	0,639
tic-tac-toe	0,982	0,976	0,984	0,980	0,982	0,976
vote	0,939	0,947	0,940	0,934	0,943	0,955
horse-colic	0,783	0,796	0,785	0,783	0,789	0,797
monk3	0,848	0,856	0,793	0,785	0,795	0,807
kr-vs-kp	0,993	0,994	0,997	0,990	0,993	0,993

Table 4.7: (cont'd) AUC obtained via ten-pass ten-fold cross-validation on 7 datasets

Tables 4.6 and 4.7 show mixed results. This is also shown in table 4.8 listing the average ranks for the twelve combinations. We will later try to explain why the results do not show a notable correlation with the experiments based on accuracy; this is partially due to implementation constraints.

Algorithm	**Average rank**
(h_{prec}, h_{prec})	5,36
(h_{prec}, h'_{prec})	8,21
(h_{prec}, h'_{lap})	6,29
(h_{prec}, h'_{mest})	5,64
(h_{lap}, h_{lap})	5,50
(h_{lap}, h'_{prec})	8,00
(h_{lap}, h'_{lap})	6,57
(h_{lap}, h'_{mest})	5,79
(h_{mest}, h_{mest})	5,50
(h_{mest}, h'_{prec})	8,93
(h_{mest}, h'_{lap})	6,14
(h_{mest}, h'_{mest})	6,07

Table 4.8: Average ranks for all twelve algorithms based on the second experiment

After obtaining results for all planned experiments, we will proceed with testing for significant differences in the entire group as well as comparisons of the individual algorithms with each other. For the former the Friedman test [3] will be used with the less conservative [3] [11] statistic $F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2}$ [11]. If the F_F statistic is greater than or equal to the critical value for a significance level of 0.05 (based on the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom [3]) we will reject the null hypothesis of no significant differences within the set of algorithms and proceed with a comparison of single algorithms based on the Post-hoc Nemenyi Test [3] using studentized range tables [10]. In the case of failure of the Friedman test, potential reasons for the experiment group in question will be identified. We will also take a look at the actual output of the individual algorithms regarding average amount of rules per decision list, average number of conditions (*rule length*) and notable phenomenons when validating certain algorithms on specific datasets to give a quick overview on possible further experiments that are not included in the scope of this thesis.

4.5 Results on 20 datasets based on accuracy

Recall the average ranks of the twelve algorithms obtained during the first experiment. To allow for further analysis, a Friedman test must first be conducted to test the entire set of results for statistical relevance.

4.5.1 Validity of the set of results

Using N=19 datasets with k=12 algorithms, we obtain a chi-square value of 20,834 and a corresponding F_F statistic of 1,993. The corresponding critical value based on a significance level of 0.05 as well as 11 and 198 degrees of freedom is 1,837, resulting in a passed Friedman test (failure at level 0.01). With a probability of at least 95 percent, the results obtained include significant differences, as such allowing further research to be done.

4.5.2 Comparison of individual algorithms

Following a passed Friedman test, the individual algorithms were compared to each other making use of the Post-hoc Nemenyi test as described above. The ranks of the algorithms as well as the critical distance are shown in figure 4.2 with comparable algorithms (no significant difference according to the Nemenyi test) connected. The significance level is 0.1 (no interesting results show for lower levels).

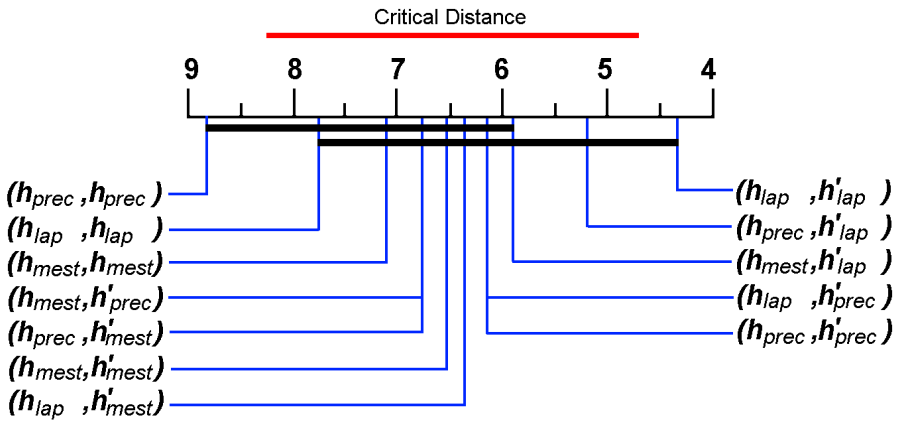


Figure 4.2: Nemenyi Test with a significance level of 0.1

The Nemenyi test supports the theory of the modified algorithm variants performing significantly better than an unaltered algorithm using pure precision for both rule selection and refinement. It can be noted that with the exception of the algorithm making use of the m-Estimate heuristic in both cases, the approach of using two different heuristics for selection and refinement in general seems to perform better, with the three algorithms making use of the modified laplace heuristic achieving best results.

In subsequent sections, we will try to uncover reasons for this behavior as well as the notable drop in performance when switching to the modified precision or m-Estimate heuristic for rule refinement. For this to work, more data concerning the properties of the learned theories is required. We will regard the m-Estimate being used for rule selection in combination with the unmodified m-Estimate as well as all three modified heuristics for rule refining; note that all of these four algorithms have been grouped by the Nemenyi test, yet they exhibit properties that change drastically depending on the heuristic used for rule refinement.

Number of rules and conditions

Dataset	(h_{mest}, h_{mest})	(h_{mest}, h'_{prec})	(h_{mest}, h'_{lap})	(h_{mest}, h'_{mest})
breast-cancer	34/158	33/189	39/179	20/66
car	161/846	161/833	162/834	165/845
contact-lenses	3/8	3/9	3/8	4/13
futebol	2/4	2/9	2/5	4/7
glass	17/55	15/241	15/90	28/84
hepatitis	8/30	6/60	7/46	6/24
hypothyroid	10/52	11/285	9/69	15/80
horse-colic	23/114	18/163	19/111	31/111
idh	3/4	2/9	2/5	2/5
iris	5/15	5/28	5/17	6/15
ionosphere	9/21	7/111	8/42	12/40
labor	3/4	3/22	3/12	3/5
lymphography	13/46	10/97	10/49	16/49
mushroom	11/13	7/44	7/35	7/29
monk3	14/44	14/50	14/45	14/40
primary-tumor	77/521	81/1001	79/563	74/298
soybean	46/151	43/516	44/192	53/163
tic-tac-toe	15/64	16/74	16/69	25/93
vote	12/63	12/69	12/59	7/25
zoo	11/15	6/48	6/14	12/14

Table 4.9: Number of rules and conditions for four different algorithms sharing the same rule selection heuristic

Based on table 4.9, the following observations can be stated, including a possible explanation for the behavior in question.

Overfitting on the training data with modified precision

Looking at above results it becomes apparent that the average number of conditions added to a rule (especially when using modified precision) for rule selection slightly supersedes the number of conditions found in other algorithm's output rules. At the same time, the amount of rules remains relatively constant or decreases slightly. A unusually large number of conditions means that the *separate-and-*

conquer learner has chosen the rule refinements in question based on only a small set of examples in each step, as the number of steps required to trigger the stopping criterion rises when coverage decreases only slowly during the refinement process. Recalling the isometrics of the modified precision variant, the reason for this behavior becomes apparent. Given an exemplary rule r, a refined rule covering only one positive and no negative examples would score the highest possible value (1.0) when evaluated with modified precision.

However, a rule constructed this way is prone to low coverage, and rules consisting entirely of conditions that only cover a small percentage of examples each are subject to noise in the training data to such an extent that the performance measured with the help of cross-validation (which aims to provide an unbiased estimate of the performance of the algorithm on new unknown testing data) drops notably. Note that basic precision already suffers from this problem, even though it usually applies to the number of positive examples while with our modification, the problem shifts to disregard the number of negative ones. This is especially true in cases of noisy datasets with a large number of attributes compared to the number of instances (and as such, alot of opportunities for the rule refinement process to overfit on the training data provided). In the case of the *hypothyroid* UCI [1] dataset, the average number of conditions in each rule is more than three times as high with modified precision being used for rule refinement than the number of conditions found in rules learned by the other three variants. This observation might explain the performance drop of modified precision being used compared to modified laplace, which is less prone to overfitting, since rule refinements with no negative examples but different amounts of positive examples covered will no longer score the same value, and better coverage will result in a higher heuristic value even if consistency stays the same.

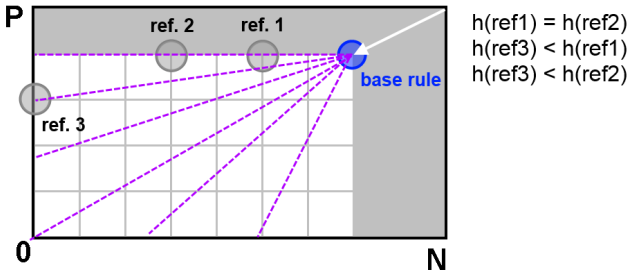


Figure 4.3: Visualization of the usage of modified precision to refine a base rule

Looking at figure 4.3, refinements 1, 2 score the exact same heuristic value, while refinement 3 scores slightly lower. Yet refinement 1 is trivially a worse refinement than refinement 2, and not necessarily better than refinement 3. Plus, if the rule is constructed out of a large amount of refinements that only denote a small step in coverage space, the problem of training data overfitting becomes apparent, resulting in lower quality rules that may still be selected by the algorithm later on based on their statistics. Note that the small shift of the isometrics origin caused by the modified laplace heuristic considerably reduces this problem during the refinement stage. I must be noted though that intelligent *tie-breaking* may be able to help resolve this issue partially; the tie-breaking mechanism (default behavior is to prefer a higher number of covered positives in case two rules score the same heuristic value) has not been altered from the baseline algorithm w.r.t. this thesis.

Larger amount of rules within decision lists with modified m-Estimate

While smaller and more symbolic datasets being used for validation do not result in any notable differences between the modified m-Estimate rule refinement heuristic compared to the other variants, a slight increase in the amount of rules learned (and as such, complexity of the theory learned) can be

noted on certain other, generally larger datasets (notably *horse-colic*, *soybean* and *hypothyroid*). Since in the case of *decision lists* used within these experiments, decision list length and order do matter w.r.t. performance of the algorithm, this may be a possible explanation of why the modified laplace heuristic, which does not feature this property, performs slightly, although not significantly, better. While not directly related to the family of algorithms that are subject to this thesis, it should be noted that simple theories consisting of fewer rules with large support seem to be preferable, which is the underlying motivation for algorithms such as ROCCER [17] that aim to reduce the size of the output theory via ROC analysis.

One reason for the slightly increased theory size when using the m-Estimate with the parameter setting of $m = 22,446$ to refine rules may be the increasing similarity to the weighted relative accuracy heuristic with large values of m, which follows a different intent than the precision-family heuristics and tends to over-generalize [12]. While the learning of a single rule will usually incorporate only a limited amount of conditions, the above results show that in some cases, this will result in more rules required to cover all positive examples (which is when the algorithm will stop adding rules to the theory and terminate with a result).

4.6 Results on 7 binary-class datasets based on the AUC

Executing a Friedman test based on the results obtained in the second half of the experiments, the null hypothesis of no significant differences between algorithms cannot be refuted at a significant level. Therefore a Nemenyi test is not admissible. Since these results differ from those obtained earlier for *accuracy* and generally seem to be of a more mixed nature, possible reasons will be discussed, focusing mostly on the method of obtaining the AUC measure for a given algorithm and dataset.

Influence of the lack of testing data within one fold of a cross-validation

Recall that conducting a ten-fold cross-validation will produce testing datasets of roughly 10 percent of the original dataset's size. This is especially relevant due to the following factors which apply to the second half of the experiments:

- The binary-class datasets already have low instance counts on average.
- The amount of datasets used may be too low to uncover statistically significant differences.
- The method used to calculate the AUC per-fold is potentially prone to bias, especially if the dataset is too sparse to reliably obtain well-stratified testing folds.

Particularly due to the reasons listed above, the results obtained show no similarity to those obtained in the previous experiment. Raising the amount of passes to ten (resulting in 10x10 cross-validation) did decrease the amount of randomness but did not cause any significant differences between algorithms as determined by the Friedman test. Recall that any two examples covered by the same rule cannot be differentiated w.r.t. the method of calculating the AUC in the underlying implementation of our *decision-list* based *Separate-and-Conquer* algorithm.

5 Related Work

We will now give a quick overview over related work in the field of AUC optimization, namely PRIE [4], ROCCER [17] and Boström's work [2].

5.1 PRIE

PRIE is a separate-and-conquer rule learner that, like the classifier previously examined in this thesis, produces a theory in the form of a single ordered rule list that is sequentially checked for the first rule to match a given example [4, p. 3]. We will first describe the method used in more detail before discussing differences with our approach.

5.1.1 Underlying basic algorithm

```
1 Initialize empty rule list RL
2 FOREACH class c: initialize ROC hull of c with ((0,0),(1,1))
3 Create initial rules for discrete and set-valued attributes
4 Create initial rules for continuous attributes
5 WHILE (<criterion>)
6         FOREACH class c: rules_for_ROC(c)
7         Rule r = best rule
8         Add r to the end of RL
```

Figure 5.1: Initialization and outer loop of PRIE. See [4, p. 8]

Figure 5.1 shows the outer loop of PRIE.The algorithm starts off with an initially empty rule list (line 1) that will form the output theory after termination. PRIE maintains a ROC space and the corresponding *convex hull* (referred to as ROCCH [4]) for each class $c_i \in C$ under consideration by setting the positive class to c and the negative class to $\{c_j \in C : j \neq i\}$. This hull is initialized with the endpoints $(0,0)$ and $(1,1)$, resulting in an initial default value of 0.5 for the AUC (line 2). Lines 3 to 4 describe the creation of *singleton rules* for every attribute a_j with value v_k as shown in table 5.1.

Attribute	Attribute type	Value	Rule(s) generated
a_j	discrete-valued	v_k	c_i :- $(a_j = v_k)$
a_j	set-valued	v_k	c_i :- $(v_k \in a_j)$
a_j	continuous-valued	v_k	c_i :- $(a_j < v_k)$ and c_i :- $(a_j >= v_k)$

Table 5.1: PRIE: Initial creation of rules

After creation of these rules, their statistics are calculated (TP, FP, tpr, fpr) as well as their respective position in ROC(c_i). PRIE now enters its outer loop as shown in figure 5.1, line 5. For every class, it will now develop rules for ROC(c_i) (line 6) as described in subsection 5.1.2, and subsequently extract the *best* rule (line 7) of *all* convex hulls. The best rule in this context can be found by choosing the rule in the segment with minimal false positive rate of the corresponding ROCCH(c_i), preferring a higher true positive rate in case of a tie. This rule is then added to the end of the rule list (line 8), following a recomputation of the ROC(c_i)- spaces and the convex hulls. The stopping criterion for the loop (line 5) is *exhaustion* of all ROC spaces, reducing them to a single line with the endpoints $(0,0)$ and $(1,1)$.

PRIE generates rules by combining existing ones, joining their conditions to form a new rule. The rule generation process is done for each class (and thus, each ROC space maintained), forming the inner loop of PRIE further elaborated in subsection 5.1.3. In order to combat the problem of NP-complete enumeration of all possible combinations, PRIE (as any separate-and-conquer rule learner) must use some sort of heuristic measure to exclude some possible combinations. PRIE does this by eliminating rules that are unlikely to extend the convex hull of each ROC space, as such attempting to optimize the resulting AUC directly. For this to work, it needs to have access to certain estimates of the performance of a rule $r_{combined}$ consisting of the conjoined conditions of two rules r_1 and r_2. The relevant statistics (tpr and fpr) of r_1 and r_2 can be assumed to be known as they have been previously calculated.

Assuming conditionally independent rules

Fawcett has shown that if the above assumption holds, given tpr_1, tpr_2, fpr_1 and fpr_2, the combined rule's position in ROC space is $(fpr_1 \cdot fpr_2, tpr_1 \cdot tpr_2)$ [4, p. 9 ff.]. Figure 5.2 shows the resulting position in ROC space for $r_{independent}$, assuming conditional independence.

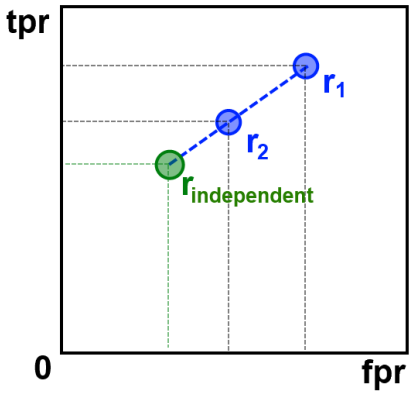


Figure 5.2: ROC position of combined rule if rules 1 and 2 are conditionally independent

Assuming the best possible result w.r.t. ROC performance

Given two rules r_1 and r_2, the best possible combination result would obviously maximize the true positive rate, while at the same time minimizing the false positive rate. Because rules are combined by conjoining their conditions, an example matching the resulting rule would have to match both r_1 and r_2. As such the upper bound for the true positive rate of the result would be the minimum of tpr_1 and tpr_2. Fawcett further derived the lower bound for the false positive rate of the resulting rule to be zero if $fpr_1 + fpr_2 = 0$ and $fpr_1 + fpr_2 - 1$ otherwise as the false positives of r_1 and r_2 might have examples in common, leading to an optimal rule located at $(max(fpr_1 + fpr_2 - 1, 0), min(tpr_1, tpr_2))$ in ROC space [4, p. 10]. The result is shown in figure 5.3.

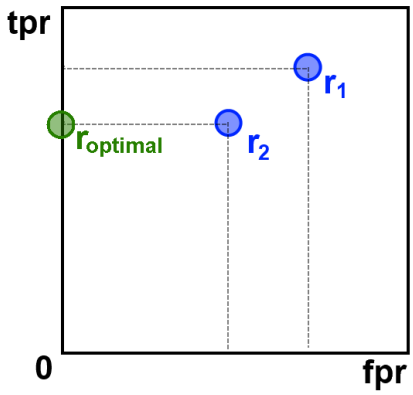


Figure 5.3: ROC position of best possible combined rule according to Fawcett

Optimism parameter and interpolation

After calculating $r_{independent}$ and $r_{optimal}$, the *optimism* parameter can be used to execute linear interpolation between both points in ROC space. This parameter ranges from zero to one, resulting in $r_optimal$ for $optimism = 1$, $r_{independent}$ for $optimism = 0$ and a linear interpolation for any value in between. The result of using a value of $optimism = 0.5$ is shown in figure 5.4. Using these assumptions, rules that can be expected not to extend the convex hull of the ROC space in question can be eliminated [4, p. 10 ff.].

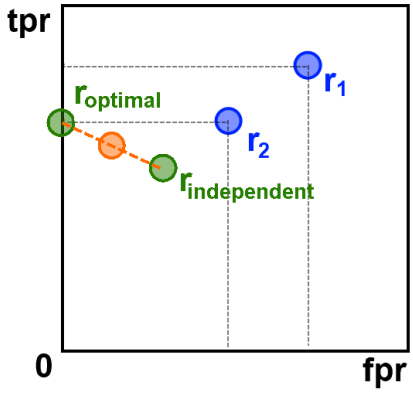


Figure 5.4: Result of using a value of 0.5 for the optimism parameter

The constraint line concept

Fawcett showed the concept of constraint lines for the independence assumption to further eliminate rule pairs that are unlikely to produce new quality rules w.r.t. ROC performance, and stating that this can be extended towards the optimistic and interpolated assumptions [4, p. 11f.]. Starting off with a rule r_1 and the set of hull segments H, a set of boundary segments B is created such as that for a rule r_2, if r_2 lies below B then the conjunction of r_1 and r_2 will not lie above H and as such, r_2 does not have to be considered for combination with r_1.

A segment $h_i \in H$ is a line expressed by the equation $y = m_{hull} \cdot x + b_{hull}$; given true and false positive rates tpr_1 and fpr_1 of r_1, Fawcett concluded that a rule r_2 can only be combined with r_1 resulting in a rule above the hull line if r_2 satisfies the inequality $tpr_2 > \frac{fpr_1 \cdot m_{hull}}{tpr_1} \cdot fpr_2 + \frac{b_{hull}}{tpr_1}$ [4, p. 11].

5.1.3 PRIE inner loop

The inner loop is where the combining of conditions to form new rules happens. This is done for each maintained ROC space making use of the techniques outlined in [4, p. 4-12]. For every hull segment examined, rule pairs that can be expected to extend the hull (based on the assumptions listed in subsection 5.1.2) are considered.
There may also be cases where the combination of two rules will result in a rule with zero coverage. This trivially is the case with two rules r_1 and r_2 where r_1 :- $(a_i = v_1)$ and r_2 :- $(a_i = v_2)$ for discrete-valued attributes a_i, as well as rules r_1 :- $(a_i < v)$ and r_2 :- $(a_i >= v)$ checking a continuous attribute a_i. In addition, duplicates in the set of conditions can be removed immediately.

5.1.4 Comparison with our derived method

PRIE, being a separate-and-conquer rule learner with a decision list-type output shares some characteristics with the algorithm described in chapter 3. Most notably, the greedy method of adding one rule per iteration to the end of a rule list remains unchanged. However, the approach to generate rules both at initialization and in the rule development process differs greatly. PRIE naturally works with multiple classes [4, p. 3], while our approach has to learn a theory for every class starting with the least frequent one, subsequently adding the resulting class-specific theories to the main one.
The main differences start when we begin to learn rules. The approach of combining rules by conjoining their conditions may seem similar to the rule refinement process of our approach described in section 3.2 as rules will grow w.r.t. the size of the condition set starting from trivial rules, however PRIE combines *rules*, while our approach combines a *rule* and exactly *one condition* in every refinement step. As such, for the rule refinement process to produce quality rules, we had to make use of rule learning heuristics as described in section 2.4 and 3.2. Our approach was to use heuristics to geometrically optimize the resulting AUC by guiding the refinement process along a preferably convex curve, but as the results in chapter 4 have shown this method might be better suited for optimizing precision, but not necessarily the AUC in all cases. PRIE can do without such a measure by working directly on the AUC; the criterion for dropping candidates in the case of PRIE is based on estimates entirely relying on AUC analysis, making use of upper- and lower bound estimates, a method of weighting these and an inequality for dismissing candidates purely based on predicted AUC performance (see constraint line concept in subsection 5.1.2).

5.2 Boström's experiments

The experiments conducted in this thesis are all based on a *decision list* theory output. Boström has compared this approach to rule learning with the method of generating order-independent *rule sets* and described the impact on the AUC in his work [2, p. 1]. We are going to summarize the key results in this section.

Problems with unordered rule sets

Unordered rule sets require a more sophisticated classification procedure than decision lists, since the contents of the latter can be checked one by one and only the first matching rule must be considered. For rule sets however, this does not apply, and more sophisticated methods must be executed. The two main problems encountered are as follows:

- For a given example, no rule at all might apply.
- For a given example, multiple rules might apply.

Using decision lists, both of these problems are avoided, since only one rule (the first matching rule) is considered for classification, and the default rule will apply to any example the previous rules failed to classify. In Boström's work, the former problem is solved by assigning the majority class to examples with no matching rules [2]; this is equal to the default rule in the *decision list*- approach. Two possible solutions for the latter problem are discussed. By imposing an order to the rules in the rule set based on class probability, this can be solved. Another approach would be to use Naive Bayes to compine the class distributions of all applicable rules.

The algorithms used in Boström's work to generate decision lists and rule sets are variants of the IREP (*Incremental Reduced Error Pruning* [7]) technique, namely IREP-O for ordered decision lists and IREP-U for unordered rule sets. IREP uses pruning [7, p. 1] to avoid the generation of heavily specialized rules; several common pruning techniques used in IREP maximize precision [2, p. 29].

Employed methods and results

Boström has conducted experiments based on IREP-O and IREP-U, using accuracy and lift ($\frac{\frac{p}{p+n}}{\frac{P}{P+N}}$) as exclusion criteria for the former, and evaluating each method both with and without post-processing. For classifying based on rule sets, both solutions to the problem of multiple applicable rules mentioned above were applied. The pruning criterion is always precision, splitting the training data into a growing and pruning set with sizes $\frac{2}{3}$ and $\frac{1}{3}$, respectively [2, p. 30].

The author has come to the conclusion that the benefits of decision lists come at the cost of a negative impact on the AUC. The explanation given by Boström centers around the problem of a default rule prohibiting the differentiation of examples belonging to the default class. Using all applicable rules and a method of combining the results resulted in a higher value for the AUC than using only the first applicable rule [2, p. 31ff.].

5.3 ROCCER

ROCCER [17] differs from standard covering algorithms (such as the algorithms examined in this thesis) as it relies on ROC analysis for selecting rules instead of employing a greedy method of adding rules based on certain heuristics. Rules are created with the help of the Apriori association algorithm and selected based on the ROC curve itself. ROCCER keeps rules in an ordered decision list separately for each class and employs a similar method to PRIE [4] by regarding the class rules are currently being selected for as positive and the union of all other classes as negative. Every rule list is initialized with a default rule predicting the positive class cite[p. 2]roccer, forming a ROC convex hull described by the points $(0,0)$ and $(1,1)$ in ROC space.

Consider for example a rule r_1 that did indeed extend the ROC convex hull. Given another rule r_2, ROCCER will now attempt to find the best position *within the rule list* for insertion of r_2. This is done by first checking if the rule's statistics (namely false and true positive rate) position the rule outside of the ROC convex hull, in which case r_2 is inserted before the first rule in the rule list. If this is not the case, ROCCER will remove all examples covered by the rule to compare r_2 with (this would be r_1 in this case), update the false and true positive rates of the rule and apply the same procedure, this time comparing r_2 with the next rule in the decision list. If no rule is left to compare with, and the position of r_2 in ROC space never extended the convex hull, r_2 is discarded; as such, concavities in the ROC curve are avoided.

Classification with ROCCER

Since ROCCER selects rules separately for each class, it maintains multiple ROC convex hulls (comparable to PRIE [4]). Given an example instance, each class is thus considered separately and the first rule that fires is determined. ROCCER is capable of generating both a ranking output and a classification output. For the former, the probability is calculated from the posterior odds (recall that each rule has an associated point (fpr, tpr) in ROC space, yielding a likelihood ratio); alternatively, ROCCER selects the class with maximum posterior odds to achieve the latter [17, p. 3].

AUC Performance of ROCCER

ROCCER has been compared with CN2 (both ordered and unordered), C4.5 (both with and without pruning) and other approaches in [17]. The results obtained showed comparable values for the AUC while at the same time generating more meaningful rule sets based on fewer rules with larger support and weighted relative accuracy, which led Prati et al. to the conclusion that the rules generated by ROCCER are more expressive by themselves compared to the output of other approaches including unpruned decision trees [17, p. 5].

6 Conclusion and Future Work

Based on the experiments on 19 datasets, we can see that the choice of heuristics has a notable influence on the performance of the respective algorithm, with one of the modified approaches, namely (h_{lapl}, h'_{lapl}), performing best. We can observe significantly better performance with the modified approaches compared to using the baseline algorithm with h_{prec}, backed by a Nemenyi test. The average ranks obtained seem to suggest further research of *Separate-and-Conquer* making use of the modified laplace or m-Estimate heuristic for rule refinement. We have determined possible reasons for the lack of performance of certain algorithms, such as Separate-and-Conquer with precision (both the original variant for rule selection and the modified one for rule refinement) tending to overfit on training data and the m-Estimate heuristic with a parameter setting of $m = 22.446$ possibly being too similar to weighted relative accuracy in order for the basic intend of mirroring the isometrics to net a profit during the rule refinement process.

The average length of the rules produced by the algorithms is a good indicator of the degree of generalization (or specialization) when compared to a baseline experiment (notably usage of Separate-and-Conquer without a distinction between rule selection and rule refinement). Given the slight performance peak when moving the origin of the modified precision isometrics to take the form of the modified laplace heuristic, which drops off after further origin shifting (recall that the m-Estimate in cases of P=N is basically a configurable laplace heuristic), it might be worth evaluating different parameter settings for m in the context of this optimization approach.

The experiments on the AUC did not provide any significant results; this may be partially due to the size and amount of datasets used as well as the method of calculating the AUC based on a *decision-list* output. The solution of using ten-pass ten-fold cross-validation did not relieve this issue; using larger datasets with ten passes of a ten-fold cross-validation was not possible at the time of writing due to memory constraints when running the implementation of Separate-and-Conquer found in the SeCo framework with modifications to accomodate for calculating the AUC on a per-fold basis.

The results of the experiments on 19 datasets seem to indicate that the optimal distance of the rule refinement heuristics origin is larger than zero (h'_{prec}), but not as large as when using h'_{mest} with the parameter setting of $m = 22.446$. This is because we observe a performance peak when using h'_{lapl} compared to the other two choices, suggesting that the optimal value for the h'_{mest}- heuristic is likely to be found within the interval $[1, 22.446]$. As such, future experiments may attempt to determine an optimal value for m in the case of the modified m-Estimate analog to Fürnkranz and Janssen [14].
Given larger binary-class datasets the experiments based on the AUC can be repeated and checked for compatibility with the results found in the first block of experiments conducted in this paper. In addition, we have disregarded *tie-breaking* in the case of multiple refinements scoring the same value, but (h'_{prec}) could benefit from this to some extent (the default tie-breaking mechanism used has not been optimized for the modified heuristics approach).

Bibliography

[1] K. Bache and M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

[2] Henrik Boström. Maximizing the area under the roc curve with decision lists and rule sets. In *SDM*. SIAM, 2007. URL http://epubs.siam.org/doi/pdf/10.1137/1.9781611972771.3.

[3] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, December 2006. ISSN 1532-4435. URL http://jmlr.org/papers/volume7/demsar06a/demsar06a.pdf.

[4] Tom Fawcett. PRIE: a system for generating rulelists to maximize ROC performance. *Data Mining and Knowledge Discovery*, 17(2):207–224, October 2008. doi: 10.1007/s10618-008-0089-y. URL http://dx.doi.org/10.1007/s10618-008-0089-y.

[5] Johannes Fürnkranz. Separate-and-conquer rule learning. *Artif. Intell. Rev.*, 13(1):3–54, February 1999. ISSN 0269-2821. doi: 10.1023/A:1006524209794. URL http://dx.doi.org/10.1023/A:1006524209794.

[6] Johannes Fürnkranz and Peter A. Flach. Roc 'n' rule learning - towards a better understanding of covering algorithms. *Mach. Learn.*, 58(1):39–77, January 2005. ISSN 0885-6125. doi: 10.1007/s10994-005-5011-x. URL http://dx.doi.org/10.1007/s10994-005-5011-x.

[7] Johannes Fürnkranz and Gerhard Widmer. Incremental reduced error pruning. In *ICML*, pages 70–77, 1994. URL http://www.ke.informatik.tu-darmstadt.de/~juffi/publications/ml-94.ps.gz.

[8] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. *Advances in Neural Information Processing Systems*, 17:513–520, 2005.

[9] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009. ISSN 1931-0145. doi: 10.1145/1656274.1656278. URL http://dx.doi.org/10.1145/1656274.1656278.

[10] H. Leon Harter. Tables of range and studentized range. *The Annals of Mathematical Statistics*, 31:1122–1147, 1960. doi: 10.1214/aoms/1177705684. URL http://projecteuclid.org/download/pdf_1/euclid.aoms/1177705684.

[11] Ronald L. Iman and James M. Davenport. Approximations of the critical region of the fbietkan statistic. *Communications in Statistics - Theory and Methods*, 9(6):571–595, 1980. doi: 10.1080/03610928008827904. URL http://www.tandfonline.com/doi/abs/10.1080/03610928008827904.

[12] Frederik Janssen. *Heuristic Rule Learning*. PhD thesis, TU Darmstadt, Knowledge Engineering Group, October 2012. URL http://tuprints.ulb.tu-darmstadt.de/3135/.

[13] Frederik Janssen and Johannes Fürnkranz. An empirical investigation of the trade-off between consistency and coverage in rule learning heuristics. In Jean-François Boulicaut, Michael R. Berthold, and T. Horváth, editors, *Proceedings of the 11th International Conference on Discovery Science (DS-08)*, pages 40–51, Budapest, Hungary, 2008. Springer-Verlag. doi: 10.1007/978-3-540-88411-8_7. URL http://www.ke.tu-darmstadt.de/publications/papers/DS08.pdf.

[14] Frederik Janssen and Johannes Fürnkranz. On the quest for optimal rule learning heuristics. *Machine Learning*, 78(3):343–379, March 2010. doi: 10.1007/s10994-009-5162-2. URL `http://www.ke.tu-darmstadt.de/publications/papers/ML2010.pdf`.

[15] Frederik Janssen and Markus Zopf. The seco-framework for rule learning. In *Proceedings of the German Workshop on Lernen, Wissen, Adaptivität - LWA2012*, 2012.

[16] Tom Mitchell. The discipline of machine learning. Technical Report CMU ML-06 108, 2006.

[17] Ronaldo C. Prati and Peter A. Flach. Roccer: An algorithm for rule learning based on roc analysis. In *IJCAI*, pages 823–828, 2005.