# On the Incremental Construction of Deep Rule Theories

Florian Beck[1], Johannes Fürnkranz[1] and Van Quoc Phuong Huynh[1]

[1]Johannes Kepler University Linz, Department of Computer Science, Institute for Application-oriented Knowledge Processing (FAW), Linz, Austria

Abstract

While we have seen considerable progress in learning rule-based theories in the past years, all state-of-the-art rule learners still learn descriptions that directly relate the input features to the target concept. However, the limitation of learning concepts in this shallow disjunctive normal form (DNF) may not necessarily lead to the desired models. In this paper, we investigate a novel search strategy that uses conjunctions and disjunctions of individuals as its elementary operators, which are successively combined to deep rule structures with intermediate concepts. We make use of efficient data structures known from association rule mining, which can efficiently summarize counts of conjunctive expressions, and expand them to handle disjunctive expressions as well. The resulting rule concepts develop over multiple generations and consist of arbitrary, deep combinations of conjunctions and disjunctions. The behavior of this algorithm is evaluated on a benchmark task from the domain of poker. A comparison to other rule learning algorithms shows that, while it is not generally competitive, it has some interesting properties, such as finding more compact and better generalizing models, that cannot be found in conventional rule learning algorithms.

**Keywords**

rule learning, concept learning, learning in logic, local search, genetic algorithms

## 1. Introduction

Traditional rule learning algorithms usually learn models that directly relate input features to the output class. This approach works well for most benchmark datasets, however, there are also datasets where it comes to its limits. One such case is the *poker*-dataset, where the task is to learn to identify the quantitative value (*pair*, *full house*, *straight*, etc.) of a hand of five cards. Every card is defined by a unique combination of a suit (clubs, spades, hearts, diamonds) and a rank (ace, 2, 3, ..., queen, king). For example, the class *one pair* includes hands where two of the five cards have the same rank and the remaining three cards a different one. Already this task of detecting one pair is particularly difficult for a rule learner, because, in order to make a correct classification, it essentially has to enumerate all possible card combinations (card 1 and 2, card 1 and 3, ..., card 4 and 5), for every rank (ace, 2, 3, ..., queen, king). Advanced concepts based on this pair concept, like *two pairs* or *full house*, are even harder to learn, so that on the full *poker*-dataset the state-of-the-art rule learner Ripper only achieves a little more than 50% accuracy, which can already be reached by simply predicting the most frequent class *nothing in hand* that covers 50% of the data as well. Even in an adapted binary version of the *poker*-dataset, where the only classes are *pair* and *no pair*, Ripper gets stuck at 70% accuracy since it is not able to generalize to pairs that it has not seen during training.

In this paper, we make first steps towards tackling this problem by trying to remove the restriction to DNF formulas that is commonly made by conventional rule learning algorithms, and directly learn arbitrary logical concept descriptions. More precisely, we investigate the behavior of a simple algorithm that successively combines input signals with logical operators, thus building up complex logic structures.

The remainder of the paper is organized as follows: Section 2 describes the problem of learning the pair concept in different logical representations and refers to related work. Based on this, we propose a combination of a local search and genetic algorithm in Section 3 and test it in various settings in Section 4. Finally, the results are concluded in Section 5.

## 2. Pair Concept

**Disjunctive Normal Form.** Ripper [1] and most other rule learners learn their models in *disjunctive normal form* (DNF) [2]. This normal form consists of a disjunction of conjunctions of literals whereby each conjunction is called a rule and the whole DNF expression a ruleset. To describe the concept of a pair in DNF for a set of $c$ cards and $r$ ranks, we have to find one rule for each possible combination of two cards for every rank as seen in the following equation:

*ITAT'22: Information technologies – Applications and Theory, September 23–27, 2022, Zuberec, Slovakia*

fbeck@faw.jku.at (F. Beck); juffi@faw.jku.at (J. Fürnkranz); vqphuynh@faw.jku.at (V. Q. P. Huynh)

0000-0003-3183-2953 (F. Beck); 0000-0002-1207-0159 (J. Fürnkranz)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-3226/paper2.pdf

$$\bigvee_{i=1}^{r}\bigvee_{j=1}^{c}\bigvee_{k=j+1}^{c} c_j = i \wedge c_k = i. \tag{1}$$

The total number of rules in this expression is $r \cdot c \cdot \frac{c-1}{2} = r \cdot \binom{c}{2}$ with each of them consisting of two literals, resulting in a total number of $r \cdot c \cdot (c-1)$ literals for the whole ruleset.

**Conjunctive Normal Form.** An alternative representation is the *conjunctive normal form* (CNF), using a conjunction of disjunctions of literals instead. In general, every logical formula can be represented in both DNF and CNF, the CNF form is less popular in rule learning though (notable exceptions include [3, 4]). However, it turned out to be similarly powerful in practical applications as well [4], with slight advantages for CNF learners on some datasets and for DNF learners on others. Looking at the pair concept in Equation (1), the corresponding CNF would be much more complex. By only looking at pairs of a single rank $r_0$, we can find a CNF of a similar complexity though:

$$\bigwedge_{j=1}^{c}\bigvee_{\substack{k=1\\k\neq j}}^{c} c_k = r_0. \tag{2}$$

Each disjunction in this CNF has a length of $c-1$ literals, so that the whole CNF contains $c$ disjunctions, resulting in the same number of $c \cdot (c-1)$ literals as the DNF when only considering a single rank $r_0$. However, in comparison to the DNF, this representation might require a closer look to be comprehensible: Whenever any card $c_k$ has the rank $r_0$, all disjunctions in the CNF besides the $r_0{}^{\text{th}}$ disjunction become true, and the latter is true if any other card has the rank $r_0$ as well to form a pair.

Note, however, that the expressions of type 2 need to be combined disjunctively, so that the resulting formula actually contains three logical layers:

$$\bigvee_{i=1}^{r}\bigwedge_{j=1}^{c}\bigvee_{\substack{k=1\\k\neq j}}^{c} c_k = i. \tag{3}$$

Converting this formula to a pure CNF would be considerably more complex than a DNF. In general, the choice whether CNF or DNF is a more compact representation depends on the problem.

**Other Representations.** In fact, if we do not restrict the logic formula to consist of only one conjunctive and one disjunctive layer, even more compact representations are possible. As an example, consider the pair concept for $c = 4$ and a single rank $r_0$. The corresponding CNF and DNF contain $4 \cdot (4-1) = 12$ literals but other representations allowing nested conjunctions and disjunctions can reduce this number to 8, as, e.g., in the following two cases:

$$\begin{aligned}&[(c_1 = r_0 \vee c_2 = r_0) \wedge (c_3 = r_0 \vee c_4 = r_0)] \vee\\ &(c_1 = r_0 \wedge c_2 = r_0) \vee (c_3 = r_0 \wedge c_4 = r_0),\end{aligned} \tag{4}$$

$$\begin{aligned}&[(c_1 = r_0 \vee c_2 = r_0) \wedge (c_3 = r_0 \vee c_4 = r_0)] \vee\\ &[(c_1 = r_0 \vee c_3 = r_0) \wedge (c_2 = r_0 \vee c_4 = r_0)].\end{aligned} \tag{5}$$

The first line of Equation (4) states that one of the first two cards as well as one of the last two cards has rank $r_0$, and the remaining two possibilities to form a pair within the first two cards or within the last two cards are covered by the conjunctions in the second line. Equation (5) uses the same conjunction in the first line and creates a similar one in the second line, this time looking at the first and third card and the second and fourth card respectively to find the remaining pair combinations.

Note that these constructions can be generalized to an arbitrary number of cards $c$: We first use two disjunctive clauses with $c/2$ terms, which describe that a sought rank occurs in the first half and in the second half of the cards, and then need to recursively encode the cases that a pair of that rank occurs in the first half, as well as that a pair occurs in the second rank. Thus, the total number of terms $N_c$ that we need for encoding all possible pairs for $c$ cards is

$$N_c = \frac{c}{2} + \frac{c}{2} + N_{\frac{c}{2}} + N_{\frac{c}{2}} = c + 2 \cdot N_{\frac{c}{2}}. \tag{6}$$

Flattening out the recursion yields

$$N_c = \log_2 c \cdot c, \tag{7}$$

which is considerably smaller than, for example, the $\binom{c}{2} = O(c^2)$ terms that are needed for learning the DNF representation of the concept.

Because of the more compact representation, deeper models like shown in Equations (4) and (5) could be easier to learn than those restricted to CNF or DNF. Furthermore, we noticed that state-of-the-art rule learners like Ripper cannot generalize their pair concept to all possible combinations and therefore only reached 70% accuracy on the binary *poker*-dataset. It may be assumed that deeper models generalize better even if not all pair combinations occur in the training data. If in the previous example the pair combination of the first and the last card is missing in the training data, an algorithm looking for deep structures might still find and use the conjunction $(c_1 = r_0 \vee c_2 = r_0) \wedge (c_3 = r_0 \vee c_4 = r_0)$ instead of splitting it into two conjunctions of size three $(c_1 = r_0 \vee c_2 = r_0) \wedge c_3 = r_0$ and $c_2 = r_0 \wedge (c_3 = r_0 \vee c_4 = r_0)$ or three conjunctions of size two $c_1 = r_0 \wedge c_3 = r_0$, $c_2 = r_0 \wedge c_3 = r_0$ and $c_2 = r_0 \wedge c_4 = r_0$. In the following, we present a genetic algorithm that is capable to build a model with arbitrary combinations of conjunctions and disjunctions to verify this hypothesis.

## 3. Algorithm

The motivation behind the work reported in this paper was to investigate, whether general logical formulas could be built up using a local search algorithm, which incrementally builds up logical structures.

The key idea is to simply start with the logical input variables, and use the logical operators AND ($\wedge$) and OR ($\vee$) to combine pairs of such values into a more complex expression. Note that such an approach should, in principle, be sufficient for learning, for example, the DNF representation in the *poker pairs* domain. If, e.g., the population size is large enough to form all possible pairs in a first iteration, all subsequent iterations could form pairwise disjunctions of such pairs. Thus, once the $\binom{c}{2}$ conjunctive pairs are formed, we would need $\log_2 \binom{c}{2}$ iterations in order to combine them into a single large disjunct, which makes in total $1 + \log_2 \binom{c}{2}$ iterations.

The approach is quite similar to pattern trees, a data structure designed for and used in fuzzy reasoning [5]. However, while this approach deals with numerical data and fuzzification operators, we remain in strictly binary worlds, where solving an optimization problem is considerably harder than in its linear counterpart [6]. Also, the structures are often built up in a top-down fashion [7], whereas we proceed from the bottom upwards.

The goal of the experiments reported is to see whether a logical formula correctly describing the pair concept can be found at all, which parameter settings would be necessary to find it, whether more compact structures can be found, or maybe even better generalizations can be achieved.

This attempt is realized in the form of a variant of a genetic algorithm, which uses two types of crossovers, one for conjunctively combining and one for disjunctively combining two individuals in the current population, as shown in Algorithm 1.

**Algorithm 1:** `learning()`-method

```
   Input: population_size, metric_type, metric_arg,
          n_generations, n_offspring, max_sim, selectors
   Output: concept
 1 population ← Population(selectors, metric_type,
     metric_arg);
 2 population.print_summary();
 3 for g ← 1 to n_generations do
 4     new_population ← population;
 5     for o ← 1 to n_offspring do
 6         for i_1 ∈ population do
 7             repeat
 8                 i_2 ←
                     tournament_selection(population,
                     tournament_size);
 9             until i_1 ≠ i_2;
10             i_conj ← crossover(i_1, i_2, true);
11             i_disj ← crossover(i_1, i_2, false);
12             sim ← i_conj.support / i_disj.support;
13             if sim ≤ max_sim then
14                 if i_conj ≠ null then
15                     new_population.add(i_conj);
16                 end
17                 if i_disj ≠ null then
18                     new_population.add(i_disj);
19                 end
20             end
21         end
22     end
23     population ←
         new_population.get_n_fittest(population_size,
         metric_type, metric_arg);
24     population.print_summary();
25 end
26 concept ← population.get_n_fittest(1, metric_type,
     metric_arg);
27 return concept
```

The algorithm builds upon data structures created by Lord, a novel rule learner developed in our group [8]. Lord reuses ideas from association rule learning and only needs a single pass through the training data during the learning phase, which makes it very efficient for large datasets. All further operations, e.g., determining how many examples are correctly and incorrectly classified for any given rule expression, can be directly extracted from n-Lists. Note that, apart from the usage of the same data structures, the genetic algorithm presented in this paper and Lord are completely different and separate approaches: While the genetic algorithm aims at finding arbitrary logical formulas bottom upwards, Lord uses the data structures to find the best DNF rule for each training example. Detailed information about n-Lists are given in [9], and about their application in the Lord algorithm in [8].

The learning method of the variant of genetic algorithm is shown in Algorithm 1 and starts by creating a default population consisting of all possible features, also known as selectors in Lord. All single features can already be interpreted and evaluated as a rule for predicting the class *pair* for some arbitrary metric and parameter (e.g., m-Estimate with $m = 10$). A first summary of the population is output to potentially analyze the best selectors.

The evolution begins in line 3 of Algorithm 1 and consists of an outer loop for each generation, which copies the population of the previous generation. It is followed by a loop for possibly generating multiple offspring per individual and two inner loops for selecting individuals for the crossover. The first individual $i_1$ is already preset whereas the second one $i_2$ is selected by a tournament selection, returning the best individual of a fixed-sized subset of the population. Since conjunctions and disjunc-

**Algorithm 2:** `crossover()`-method

**Input:** $i_1$, $i_2$, conjunctive
**Output:** $i_3$

```
1  i1, i2 ← order(i1, i2);
2  if conjunctive then
3  |   nlist ← conj(i1.nlist, i2.nlist);
4  |   i3 ← Individual(nlist, i1.body + "&&" + i2.body)
5  else
6  |   nlist ← disj(i1.nlist, i2.nlist);
7  |   i3 ← Individual(nlist, "(" + i1.body + "||" + i2.body
   |    + ")")
8  end
9  if i3.support = i1.support ∨ i3.support = i2.support ∨
   |  i3.support = 0 then
10 |   return null
11 end
12 return i3
```

tions with itself are not changing the coverage of the expression, we force $i_1$ and $i_2$ to be different individuals before starting with the crossovers. Note that both the conjunction and disjunction are computed successively in lines 10 and 11.

Algorithm 2 describes the crossover procedure. After ordering the two individuals alphabetically, either the conjunctive or disjunctive n-List and condition string are built. If one of the individuals covers a subset of instances of the other one, or both individuals are disjoint from each other, the resulting individual is meaningless and *null* is returned instead. Otherwise, the created crossover is returned to the learning method.

Optionally, in lines 12 and 13 of Algorithm 1, the Jaccard similarity between the two generated crossovers is computed to avoid offspring that is too similar to its conjunction respectively disjunction "sibling" and parents. If this is not the case, the crossovers are added to the new population. Since the population increases this way, at the end of each generation the population is filtered and only the $n$ best individuals are kept (line 23). By printing the summary of the population in line 24, the maximum heuristic value and the ten best individuals are output.

Finally, the best individual of the last generation is returned as the concept, which can then be used to evaluate whether test examples are covered by the concept.

# 4. Experiments

For the experiments, we generated multiple versions of the *poker pairs*-dataset with varying difficulty:

- *pairs2* consisting of 30 card combinations with 12 pairs ($c = 2$ cards, $r = 2$ ranks, $s = 3$ suits)
- *pairs3* consisting of 336 card combinations with 144 pairs ($c = 3$ cards, $r = 4$ ranks, $s = 2$ suits)
- *pairs4* consisting of 11,880 card combinations with 6,120 pairs ($c = 4$ cards, $r = 6$ ranks, $s = 2$ suits)
- *pairs4a* consisting of 6,660 card combinations with 900 pairs ($c = 4$ cards, $r = 6$ ranks, $s = 2$ suits). In comparison to *pairs4*, all pairs besides those with rank $r_0 = 6$ are removed, and additionally those with $c_1 = 6$ and $c_4 = 6$ are retained for evaluation.

All datasets are split into ten folds, whereby just use nine of them are used for training to break symmetries. In all experiments, 10 generations with a population size of 100 and a tournament size of 5 are used, and the rules are evaluated by the m-estimate metric with $m = 10$. The m-estimate value $h_m$ of a rule $r$ predicting class $c$ has been proposed by Cestnik [10] and is calculated as

$$h_m(r) = \frac{r.p + m\frac{P}{P+N}}{r.p + r.n + m}, \tag{8}$$

where

$m$ = a settable parameter in the range $[0, +\infty)$
$r.p$ = the number of true positives of rule $r$
$r.n$ = the number of false positives of rule $r$
$P$ = the number of examples with class $= c$
$N$ = the number of examples with class $\neq c$.

It provides an excellent, tunable trade-off between weighted relative accuracy ($m \to \infty$), which is frequently used in descriptive rule learning, and precision ($m \to 0$), the main target for predictive learning [11].

The results of the experiments are summarized in Tables 1 and 2, the detailed evaluation is split into one paragraph per *poker pairs*-dataset.

**Table 1**
Average percentage of pairs covered by the final concept (number of folds with 100% coverage in parenthesis). The column "full c." denotes the number of generation where all crossovers have been computed.

| full c. \ pairs | 2 | 3 | 4 | 4a |
|---|---|---|---|---|
| 0 | 93.5% (5) | 74.0% (0) | 39.3% (0) | 98.9% (6) |
| 1 | 100.0% (10) | 100.0% (10) | 87.0% (0) | 100.0% (10) |
| 2 | 100.0% (10) | 98.8% (8) | 63.0% (0) | 100.0% (10) |

**Table 2**
Average number of generations needed to find best concept.

| full c. \ pairs | 2 | 3 | 4 | 4a |
|---|---|---|---|---|
| 0 | 4.4 | 5.8 | 6.5 | 5.3 |
| 1 | 3.2 | 5.4 | 7.8 | 5.0 |
| 2 | 2.0 | 5.1 | 7.2 | 3.5 |

**Pairs within two cards.** In this minimalistic dataset, there are only two different types of pairs since only two cards and ranks are available. Both the minimal DNF $c_1 = 1 \wedge c_2 = 1 \vee c_1 = 2 \wedge c_2 = 2$ and the corresponding CNF $(c_1 = 1 \vee c_2 = 2) \wedge (c_1 = 2 \vee c_2 = 1)$ consist of four literals and could be found in two steps.

Using a strict genetic algorithm approach, i.e., without building all possible crossovers in the first generation(s), the learned concept describes all pairs for 5 out of 10 folds, and this concept is found between the third and fifth generation. The 5 remaining folds also diverge to a fixed concept (in different logical expressions) the latest in the sixth generation, however, in 4 of them one or two pairs are not covered respectively, and in the last one a non-pair was covered mistakenly.

If in the first generation all possible crossovers are generated, thus in particular the crossovers $c_1 = 1 \wedge c_2 = 1$ and $c_1 = 2 \wedge c_2 = 2$, the perfect theory can be found in all 10 folds. 6 of theses folds need four generations to do so, the remaining 4 find it already in the second generation. Interestingly, one of them also finds the minimal CNF additionally to the minimal DNF.

Finally, if also in the second generation all crossovers are generated, both the minimal DNF and CNF are found in all folds.

**Pairs within three cards.** The next dataset is already more complex; three cards and four ranks lead to 12 pair combinations, which need a length of at least 20 literals if using an arbitrary nested logical expression and a length of at least 24 literals if using a DNF.

Even with problems of this size, the presented simple genetic algorithm can not find an expression covering all pairs in any fold. The percentage of covered pairs ranges from 56% to 92% and is approximately equally distributed, same holds for the number of generations that lies between 4 and 7.

As discussed in Section 3, already a single generation with all possible crossovers can fix this problem. Even if the population size is too small to keep all crossovers, it is still large enough so that only irrelevant crossovers are removed immediately (those taking the suit attributes into account). In all folds, a concept covering all pairs is learned in the fifth generation or sixth generation.

The average number of generations needed as well as the overall complexity of the found concepts can be slightly decreased by computing all crossovers for a second generation. Surprisingly, this approach leads to incomplete concepts in two folds though, that are missing 1 of the 12 pair combinations. This indicates that an exhaustive search over multiple generations leads to too many similar individuals that prevent other diverse individuals from being included into the population and used for further concepts.

**Pairs within four cards.** In the dataset with four cards we used six ranks, summing up to 36 pair combinations. While a minimal DNF needs at least 72 literals, with deeper logical expressions sketched in Equations 4 and 5 only 48 literals are needed. However, this requires to find suitable disjunctions in the first generation.

The results for the dataset with four cards are similar to the previous one with three cards but with even more significant differences between the three settings. The performance of the pure genetic algorithm setting drops to a pair coverage between 24% and 49% and it converges in an even later generation than in the smaller datasets. This can again be fixed by a single exhaustive search for crossovers and a subsequent genetic algorithm covering between 79% and 93% of the pairs. However, similar to the dataset with three cards, if also in the subsequent generation all crossovers are computed, the convergence of the algorithm is sped up but the pair coverage decreases to percentages between 52% and 77%.

While the best performance is achieved by the algorithm using a single generation with an exhaustive crossover search, all resulting models have in common that they learn DNFs instead of creating intermediate disjunctions before conjuncting them. To investigate further into this, we refine the dataset in the following paragraph.

**Subset of pairs within four cards.** In the last experiment, we want to focus on a different aspect: the capability of the learner to augment the concept to pairs that are not covered in the training data like described in the end of Section 2. We consider pairs with cards of rank 6 and ensure that only five out of six are part of the training data. As a consequence, state-of-the-art DNF rule learners like Ripper are only capable to detect these five pair combinations and conjunct them:

$$\begin{aligned}&(c_1 = 6 \wedge c_2 = 6) \vee (c_1 = 6 \wedge c_3 = 6) \vee \\ &(c_2 = 6 \wedge c_3 = 6) \vee (c_2 = 6 \wedge c_4 = 6) \vee \\ &(c_3 = 6 \wedge c_4 = 6)\end{aligned} \tag{9}$$

In this representation, the remaining pair combination $c_1 = 6 \wedge c_4 = 6$ remains uncovered. However, we have seen in Equations 4 and 5 that even smaller concepts can be learned, and those possibly also cover the missing pair combination. We verify whether similar models can indeed be learned with the suggested approach. Starting with the pure genetic algorithm, the learner does not even reliably find all pair combinations. In four out of ten folds, parts of the concepts additionally use suit attributes in the conjunctions, which led up to 37 of 810 pair instances uncovered.

By generating all crossovers in the first generation, the five pair combinations are found and combined to a

```
Generation:   1 Max fitness: 0,9503
0,9503 (164|0) c1=6 && c2=6 -> class=1
0,9503 (164|0) c1=6 && c3=6 -> class=1
0,9497 (162|0) c2=6 && c3=6 -> class=1
0,9491 (160|0) c4=6 && c2=6 -> class=1
0,9491 (160|0) c4=6 && c3=6 -> class=1
0,3822 (  4|0) c3=5 && c4=5 -> class=1
0,3822 (  4|0) c2=5 && c4=5 -> class=1
0,3822 (  4|0) c4=1 && c1=1 -> class=1
0,3822 (  4|0) c2=1 && c4=1 -> class=1
0,3822 (  4|0) c3=1 && c4=1 -> class=1
```

```
Generation:   2 Max fitness: 0,9869
0,9869 (648|0) (c1=6 || c4=6) && (c2=6 || c3=6) -> class=1
0,9826 (486|0) (c1=6 || c2=6) && (c4=6 || c3=6) -> class=1
0,9826 (486|0) (c1=6 || c3=6) && (c4=6 || c2=6) -> class=1
0,9744 (328|0) (c1=6 && c2=6 || c1=6 && c3=6) -> class=1
0,9744 (328|0) c1=6 && (c2=6 || c3=6) -> class=1
0,9743 (326|0) (c1=6 && c2=6 || c2=6 && c3=6) -> class=1
0,9743 (326|0) (c1=6 && c3=6 || c2=6 && c3=6) -> class=1
0,9743 (326|0) (c1=6 || c3=6) && c2=6 -> class=1
0,9743 (326|0) (c1=6 || c2=6) && c3=6 -> class=1
0,9741 (324|0) (c1=6 && c2=6 || c4=6 && c2=6) -> class=1
```

**Figure 1:** Best ten individuals in the first and second generation respectively.

DNF similar to Equation 9 within five generations, i.e., one iteration more than in the optimal logarithmic case. Therefore, all training instances consisting one of these five pair combinations are covered, but the test instances with the remaining sixth pair combination remain disregarded.

This behavior changes if the second generation computes all crossovers as well. Figure 1 lists the best ten individuals found in the first and second generation. While in the first generation this list only consists of conjunctions that are already able to only cover pairs, in the second generation the best three individuals are conjunctions of disjunctions, i.e., crossovers of individuals that have not been among the best ten in the first generation. In particular, the second and third individual evaluate better than all possible individuals in DNF and at the same time still cover the missing sixth pair combination.

Based on these individuals, a concept covering all pairs of the training set can be found already in the third generation for five of the folds, and in the fourth generation for the remaining five folds. The percentage of individuals covering the additional the missing sixth pair combination ranges from 13% to 31%. While this number is still low, it is worthwhile to mention that in comparison to DNF rule learner it is capable to generate such individuals, and in comparison to the other two versions it also actually finds them.

## 5. Conclusion

In this work, we dealt with the question when conventional DNF rule learners come to their limits. We conducted this on the basis of a *poker pairs*-dataset and showed that deep rule theories can both describe the pair concept in a more concise form and are capable to generalize the concept better. To investigate into this behavior in practical experiments, we created a simple variant of a genetic algorithm, which uses conjunctive and disjunctive crossovers to built deep rule concepts out of single features. We also implemented the possibility to generate all possible crossovers at the beginning of the algorithm, which can then be seen as a local search approach.

While the presented combination of a local search and genetic algorithm is not yet matured to be successfully applied on arbitrary real-world datasets, through the application on different versions of the *poker pairs*-dataset we could take away some interesting properties of the deep rule theories learned this way. An exhaustive search for crossovers in the first generation was needed to reliably find concepts covering all or at least most pairs. However, an additional exhaustive search in the subsequent generation led to too many similar individuals and resulted in a worse performance for the chosen parameters. Finally, the last experiment on a subset of the *poker pairs*-dataset with four cards showed that the presented learner indeed is capable to find the more compact and better generalizing model described at the beginning of this paper, which can not be learned by state-of-the-art rule learners, and highlights the importance of subsequent combinations of disjunctions and conjunctions to do so.

## 6. Future Work

In future work, further refinements of the genetic algorithm are necessary and partly already implemented. Most emphasized should be the handling of diversity, which can, e.g., either be handled by a separate reserve population that focuses on covering a wider variety of examples instead of covering the most examples as possible or by penalizing the evaluation of individuals if their Jaccard similarity is too high. Additionally, a penalty for the rule length might force the algorithm to prefer the most compact representation between those covering the same instances. Last but not least, the idea of mutations might be refined for the rule learning setting. Features on all levels of the logical expression could be removed, replaced or augmented, and conjunctions could be interchanged with disjunctions and vice versa. For all these suggestions, a further in-depth analysis is needed though.

## References


[1] W. W. Cohen, Fast effective rule induction, in: A. Prieditis, S. Russell (Eds.), Proceedings of the 12th International Conference on Machine Learning (ML-95), Morgan Kaufmann, Lake Tahoe, CA, 1995, pp. 115–123.

[2] J. Fürnkranz, D. Gamberger, N. Lavrač, Foundations of Rule Learning, Springer-Verlag, 2012.

[3] A. Dries, L. D. Raedt, S. Nijssen, Mining predictive k-CNF expressions, IEEE Transactions on Knowledge and Data Engineering 22 (2010) 743–748. URL: https://doi.org/10.1109/TKDE.2009.152. doi:10.1109/TKDE.2009.152.

[4] R. J. Mooney, Encouraging experimental results on learning CNF, Machine Learning 19 (1995) 79–92.

[5] E. Hüllermeier, From knowledge-based to data-driven fuzzy modeling – development, criticism, and alternative directions, Informatik Spektrum 38 (2015) 500–509. URL: https://doi.org/10.1007/s00287-015-0931-8. doi:10.1007/s00287-015-0931-8.

[6] A. Schrijver, Theory of linear and integer programming, John Wiley & Sons, 1998.

[7] R. Senge, E. Hüllermeier, Top-down induction of fuzzy pattern trees, IEEE Transanctions on Fuzzy Systems 19 (2011) 241–252. URL: https://doi.org/10.1109/TFUZZ.2010.2093532. doi:10.1109/TFUZZ.2010.2093532.

[8] V. Q. P. Huynh, J. Fürnkranz, F. Beck, Efficient learning of large sets of locally optimal classification rules, Under review for journal publication, 2022.

[9] Z.-H. Deng, S.-L. Lv, Prepost+: An efficient n-lists-based algorithm for mining frequent itemsets via children–parent equivalence pruning, Expert Systems with Applications 42 (2015) 5424–5432.

[10] B. Cestnik, Estimating probabilities: A crucial task in Machine Learning, in: L. Aiello (Ed.), Proceedings of the 9th European Conference on Artificial Intelligence (ECAI-90), Pitman, Stockholm, Sweden, 1990, pp. 147–150.

[11] J. Fürnkranz, P. A. Flach, ROC 'n' rule learning – Towards a better understanding of covering algorithms, Machine Learning 58 (2005) 39–77.