GPU-Accelerated Parameter Optimization for Classification Rule Learning

Greg Harris
Department of Computer Science
University of Southern California
Los Angeles, California
gfharris@usc.edu

Anand Panangadan
Department of Computer Science
California State University
Fullerton, California
apanangadan@fullerton.edu

Viktor K. Prasanna
Ming-Hsieh Department
of Electrical Engineering
University of Southern California
Los Angeles, California
prasanna@usc.edu

Abstract

While some studies comparing rule-based classifiers enumerate a parameter over several values, most use all default values, presumably due to the high computational cost of jointly tuning multiple parameters. We show that thorough, joint optimization of search parameters on individual datasets gives higher out-of-sample precision than fixed baselines. We test on 1,000 relatively large synthetic datasets with widely-varying properties. We optimize heuristic beam search with the m-estimate interestingness measure. We jointly tune m, the beam size, and the maximum rule length. The beam size controls the extent of search, where over-searching can find spurious rules. m controls the bias toward higher-frequency rules, with the optimal value depending on the amount of noise in the dataset. We assert that such hyper-parameters affecting the frequency bias and extent of search should be optimized simultaneously, since both directly affect the false-discovery rate. While our method based on grid search and cross-validation is computationally intensive, we show that it can be massively parallelized, with our GPU implementation providing up to 28x speedup over a comparable multi-threaded CPU implementation.

Introduction

Classification rule finding is a well-studied data mining technique for learning “If-Then” concepts from data. The search for high-precision rules involves testing many candidates against training data. This repeated use of the training data is called *multiple testing* and increases the false-discovery rate. Several authors have shown that searching too hard for good rules can lead to over-fitting, particularly if complex rules are allowed (Quinlan and Cameron-Jones 1995; Jensen and Cohen 2000; Možina et al. 2006; Janssen and Fürnkranz 2009). This is a trade-off, because some searching must certainly be done.

Another trade-off is the balance of preference given to precision and frequency. Higher-frequency rules are more statistically significant and robust to over-fitting. A rule with many examples in the training data may be preferable to a higher-precision rule with fewer examples.

These two trade-offs depend on each other and should be decided jointly. Both affect the false-discovery rate. They also both depend on the amount of noise in the domain. For example, if performing an extensive search through a noisy dataset, one may need to increase the frequency bias in order to reduce the chances of selecting a spurious rule. These trade-offs are controlled through algorithm-specific hyper-parameters.

Despite the sensitivity of rule learning accuracy to search parameters, rarely are they tuned for individual datasets. Studies which compare algorithm precision often use default parameter values. A review of 100 experimental studies on Inductive Logic Programming found that only 38 mention values assigned to some parameters, and only 17 describe an enumerative search over a small set of possible values to optimize typically one parameter (Srinivasan and Ramakrishnan 2011). Our study shows that thorough, joint parameter optimization significantly improves out-of-sample precision.

Presumably, the reason joint optimization of multiple hyper-parameters is not done in rule learning is due to the high computational cost. To show this approach is feasible, we show that it can be massively parallelized. We describe a GPU implementation that provides up to 28x speedup over a comparable multi-threaded CPU implementation.

Background

Classification Rules

The classification rules we consider take the form: “If A, then B.” In binary classification, B is always the same: $(class = Positive)$. A consists of a conjunction of logical conditions placed on input attributes, called *features*.

We use the same notational conventions as Fürnkranz and Flach (2005): The number of positive samples covered by the rule is p. The total number of positive samples is P. The number of negative samples covered by the rule is n. The total number of negative samples is N.

Interestingness Measures

Rule-finding is a combinatorial optimization problem with the objective of maximizing an *interestingness measure*. Many interestingness measures have been proposed. Surveys on the topic compare and contrast the proposed mea-

	Covered by rule	Not covered by rule	
Pos samples	p	$P - p$	P
Neg samples	n	$N - n$	N
	$p + n$	$P + N - p - n$	$P + N$

Table 1: Notational conventions for a confusion matrix of positive and negative samples covered or not covered by a classification rule.

sures (Tan, Kumar, and Srivastava 2002; Geng and Hamilton 2006; Lenca et al. 2008; Janssen and Fürnkranz 2010). Interestingness measures are calculated from values found in a 2×2 contingency table, as seen in Table 1. Each measure has a unique way of trading-off precision for frequency (i.e. coverage or support).

Beam Search

Counter-intuitively, searching more extensively for good rules can lead to worse out-of-sample performance. Searching involves generating hypotheses, or candidate rules, and evaluating them on the training data.

Quinlan and Cameron-Jones (1995) speculate that any training dataset contains “fluke theories” that fit the data well but have poor predictive accuracy. The more extensive the search is, the higher the probability of discovering the flukes. Greedy heuristic search algorithms, such as beam search, work well in this regard. They tend to find good rules quickly, while covering only a small fraction of the search space. By contrast, complete search algorithms such as Apriori (Agrawal et al. 1996) or FP-growth (Han, Pei, and Yin 2000) are guaranteed to discover all flukes as well as good rules.

Top-down heuristic beam search has emerged as the most commonly used search algorithm in rule finding. It is similar to hill-climbing, although less myopic. Hill-climbing starts with an empty rule that covers all samples and greedily refines it by adding conditions. Beam search considers more refinements or alternatives than hill-climbing throughout the search. It maintains a beam of the best b rules of each rule length. To find rules of length l, each feature is added as a refinement to each rule in the beam for length $l - 1$. The refined candidate rules are evaluated, and the the top b rules are then stored in the beam for length l. After reaching some stopping criterion, the best rule from all lengths is selected.

Limiting the beam size limits the extent of the search. Setting $b = 1$ makes the algorithm equivalent to hill-climbing. Setting $b = \infty$ makes it equivalent to exhaustive search.

Related Work

For related work, we consider research on methods of tuning search parameters for each dataset with the goal of finding high-precision rules.

Cestnik (1990) proposed the m-estimate, a parameterized interestingness measure for estimating conditional probabilities. Dzeroski, Cestnik, and Petrovski (1993) show an improvement over the Laplace estimate by trying many values of m on each dataset. They mention cross-validation as a possibility for choosing m.

Quinlan and Cameron-Jones (1995) use the Laplace estimate for an interestingness measure. They iteratively increase the beam size until two successive iterations fail to find a better rule. They call this *layered search*, and report an improvement over a more extensive search. This is related, because they adapt the beam size to the dataset.

Cohen (1995) propose RIPPER, which is still a popular and competitive algorithm. RIPPER learns one rule at a time using hill-climbing in conjunction with FOIL’s information gain and pre-pruning. Each rule is learned on only 2/3 of the training data. Each is then post-pruned by removing any final sequence of conditions from the rule to optimize precision on the remaining 1/3 of the training data. The stopping criterion for adding new rules is based on total description length. Finally, there is a post-processing phase for optimizing the rule set. This is related work, because using held-out data for post-pruning is a method of adapting the rules to the dataset.

Možina et al. (2006) remove the multiple testing-induced optimism in rule evaluation measures by performing permutation tests. They choose the rule that is most statistically significant based on permutation tests for each rule length.

Srinivasan and Ramakrishnan (2011) study parameter optimization for Inductive Logic Programming (ILP) systems. They propose first using a regression model to find the most relevant factors. This is followed by response surface optimization on the relevant parameters. Their optimization method is based on gradient ascent and is not guaranteed to find the global maximum. They do not use cross-validation for parameter optimization. In their experiment, they consider four parameters: maximum rule length, maximum nodes explored in the search, minimum precision, and minimum support. They validate their method on 8 biochemistry datasets. Our method uses grid search instead of gradient ascent, because we believe some variables, such as extent of search and frequency bias, interact in a way that is difficult for greedy algorithms to optimize. We use cross-validation to reduce the chances of over-fitting the hyper-parameters. We also test more extensively, using 1,000 datasets.

We include here two related GPU search implementations. Fang et al. (2009) showed some speedup when implementing Apriori on a GPU versus a CPU. Langdon, Yoo, and Harman (2011) implemented beam search on a GPU for the purpose of Formal Concept Analysis, although they did not achieve speeds as fast as an optimized CPU algorithm.

Approach

Our approach to rule learning is to thoroughly optimize search parameters for each dataset. Our experiments evaluate doing this through grid search with cross-validation. Grid search works even when optimizing variables that interact. Cross-validation helps reduce the chances of over-fitting the search parameters.

We assert that frequency bias and extent of search should be optimized jointly. Both affect statistical significance. Increasing the extent of search increases the number of false discoveries. This lowers the statistical strength of the search, making it difficult to recognize good low-frequency rules among all the spurious ones. Only higher frequency rules can still possibly be found statistically significant as the false discovery rate rises. Therefore, the optimal frequency bias depends on the extent of search. In noisy datasets with little structure, dropping unnecessary hypothesis tests will enable a reduction in frequency bias, potentially leading to an increase in out-of-sample precision.

Baseline Method

We begin by choosing a baseline search method to optimize; namely, beam search with the m-estimate, with parameter values recommended in the literature. This baseline represents arguably the best general-purpose algorithm for finding high-precision rules.

Search Algorithm. Top-down beam search is a greedy heuristic search method which is used by the vast majority of rule learning algorithms (Fürnkranz, Gamberger, and Lavrač 2012). It is used by systems built on the AQ algorithm (Clark and Niblett), CN2 (Clark and Niblett 1989), and SeCo (Fürnkranz 1999). Beam search generalizes hill-climbing, which is used in FOIL (Quinlan 1990) and RIPPER (Cohen 1995).

The parameter b for beam size varies the extent of search. For the baseline beam size, we use $b = 5$. This value has been commonly used and is the default value used in CN2 (Clark and Niblett 1989; Možina et al. 2006). Low values of b have been shown to often out-perform large values when tested out-of-sample (Quinlan and Cameron-Jones 1995; Janssen and Fürnkranz 2009).

Interestingness Measure. The m-estimate (Cestnik 1990) in conjunction with beam search is commonly used in rule learning, with accuracy comparable to other state-of-the-art algorithms. The m-estimate generalizes the Laplace estimate, replacing the assumption of a uniform initial distribution with a more flexible probability density function:

$$f(x) = \frac{1}{B(a,b)} x^{a-1}(1-x)^{b-1}$$

where $a > 0$, $b > 0$, and $B(a, b)$ is the Beta function. The values for a and b are determined from an estimate of the class proportion and a parameter m, which is chosen based on the amount of noise in the domain. The formula for the m-estimate is:

$$h_m = \frac{p + m \cdot \frac{p}{P+N}}{p+n+m}$$

Janssen and Fürnkranz (2010) included the m-estimate in a study comparing five parameterized interestingness measures. They optimized the parameter value for each measure with respect to ruleset accuracy across 27 datasets and validated the results using that parameter on 30 datasets. The optimal general-purpose value of m they reported is $m = 22.466$, which we use for m in the baseline.

Rule Length. Shorter rules are easier to interpret and easier to find. In addition, shorter rules are believed by some to generalize better to unseen data. The number of features in a ruleset is a measure of model complexity. Fürnkranz, Gamberger, and Lavrač (2012) have proven that when learning from noisy datasets, the size of an over-fitting model grows at least linearly with the number of samples. They say that biasing a learner toward shorter rules is a typical way to counter the growth and avoid over-fitting. The bias is generally not explicit, but implemented through pre- or post-pruning.

One way to ensure rules are short is simply to constrain the maximum rule length, an approach often taken in association rule mining. The baseline method does not have a restrictive maximum rule length. However, we include it as a search parameter to be optimized, because either it or another stopping criterion must be specified. Setting a maximum rule length adds little extra complexity to the grid search, because it entails simply storing intermediate results during the search for longer rules.

In summary, the baseline we compare against uses:

1. Beam search ($b = 5$)
2. m-Estimate ($m = 22.466$)
3. Maximum rule length ($l = \infty$)

Experiments

We test the hypothesis that grid search with cross-validation can reliably identify parameter combinations that lead to higher-precision rules than the baseline parameters. We focus on the covering algorithm subroutine that finds the single best rule for the uncovered training samples. For stability in the precision estimation, we combine as many of the best rules as needed until they cover at least 100 samples in the test data.

3-Fold Cross-Validation

We use k-fold cross-validation with $k = 3$. Our prior experiments have shown the results are not sensitive to the value of k. We choose $k = 3$ primarily on the basis of lowered computational complexity.

Synthetic Data

In order to report statistically significant results, we test on many datasets. Using synthetic data allows us generate unlimited datasets. It also allows us to experiment on data with a wide variety of attributes.

We have identified important attributes used to describe datasets. Before generating each dataset, we first pick reasonable values at random for each attribute (see Table 2). We then generate random binary samples and labels, constrained to meet the chosen attribute values. We also create random rules with reasonable attributes, and we incorporate them into the dataset without affecting the other attributes. This gives the otherwise random data some amount of predictable structure. We follow this process to create two datasets, one for training, and one for testing. Each has identical attributes and contains the same structure from the generated rules.

Attribute	Value Distribution
Sample count (M)	$M \sim \mathcal{U}\{50000, 500000\}$
Feature count (N)	$N \sim \mathcal{U}\{50, 5000\}$
Positive class proportion (μ)	$\mu \sim \mathcal{U}(0.1, 0.9)$
Mean noise features per sample (λ)	$\lambda \sim \mathcal{U}\{10, \lfloor N/4 \rfloor\}$
Noise features for sample m (f_m)	$f_m \sim Poisson(\lambda) \mid f_m \geq 1$
Rule count (R)	$R \sim \mathcal{U}\{1, 20\}$
Max rule lift (ϕ)	$\phi \sim \mathcal{U}(0, 0.95 - \mu)$
Max rule support (θ)	$\theta = \lceil M \times \mu/10 \rceil$
Length for rule r (l_r)	$l_r \sim \mathcal{U}\{2, 6\}$
Precision for rule r (p_r)	$p_r \sim \mathcal{U}(\mu, \mu + \phi)$
Support for rule r (s_r)	$s_r \sim \mathcal{U}\{5, p_r \times \theta\}$

Table 2: Synthetic data attribute values are drawn from the given distributions, and samples and rules are randomly generated such that the attribute conditions are satisfied.

Experiment Plan

We use 3-fold cross-validation in a three-dimensional grid search to find the best values for m in the m-estimate, b in beam search, and l, the maximum rule length. We test all combinations of the following values:

- $m \in \{1, 2, 5, 10, 22.466, 50, 100, 1000\}$
- $b \in \{1, 2, 3, 5, 10, 20, 50, 100, 1000\}$
- $l \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

Cross-validation evaluates each parameter combination on folds of the training data. The best combination is then used to search the full training data for rules which are evaluated on the test data. This is done for 1,000 datasets.

Three-dimensional grid search seems computationally cumbersome, but the third dimension, maximum rule length, doesn't add much complexity. This is because longer rules build on shorter rules when using top-down beam search. If the intermediate results are stored, a single beam search for long rules collects the same information as separate beam searches for smaller values of l. The baseline recommendation is having no maximum rule length at all, meaning long rules need to be searched anyway.

In addition to testing cross-validation for all three parameters, we also evaluate cross-validation for individual parameters (one-dimensional grid search) and pairs of parameters (two-dimensional grid search), with left-out parameters being set to recommended values from the baseline approach.

For comparison, we test beam search ($b = 5$) with a large selection of non-parameterized interestingness measures. We also test five parameterized interestingness measures, using parameter values recommended by Janssen and Fürnkranz (2010). One of these tests, the m-estimate ($m = 22.466$), is the baseline method all other experiments are compared against. These tests should ideally be done with no maximum rule length, but for practical purposes, we set the maximum rule length to $l = 10$. This limit should not generally be restrictive, because the maximum true rule length in our synthetic data is $l = 6$.

We test an implementation of RIPPER modified for our use case. Our implementation only finds enough rules to cover 100 test samples. We do not generate the full rule list or perform ruleset pruning, but we still post-prune the individual rules we do find. We only search for rules predicting the positive class. We call this, "Modified RIPPER."

Results

Experiment results are listed in Table 3. The first noticeable feature is that most of the p-values are essentially 1, meaning the method is unlikely to be higher precision than the baseline, given the win-loss-tie record. These results confirm the results of Janssen and Fürnkranz (2010), finding that the m-estimate ($m = 22.466$) outperforms all non-parameterized interestingness measures. Both our study and theirs also confirm that RIPPER and the m-estimate perform equally well. Our results differ, however, in that Relative Cost ($c_r = 0.342$) performed better in their study than ours.

The experiment results confirm our hypothesis that grid search with cross-validation can be used to improve on the baseline method with respect to out-of-sample precision. The best result, CV (m, b), has a win-loss-tie record of 511-416-73 against the baseline.

While cross-validation with all three parameters still outperforms the baseline with statistical significance, it is not the best result. The results show worse performance in every case where the maximum rule length (l) is included in the cross-validation. Evidently, either enforcing a maximum rule length is counter-productive, or else the optimal value of l is difficult to determine using cross-validation.

Figure 1 confirms our assertion that extent of search and frequency bias should be optimized jointly. It shows that large beam sizes are optimal in the training data more often when m can be varied to compensate.

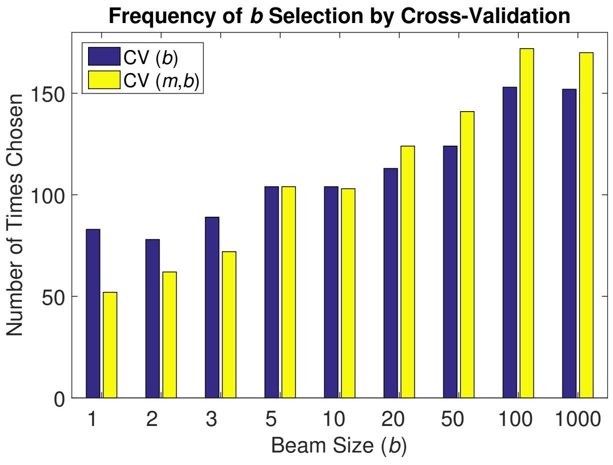

Figure 1: The number of datasets where cross-validation chose each beam size.

Method	Win - Loss - Tie	p-Value
CV (m, b)	511 - 416 - 73	0.0007
CV (m)	420 - 344 - 236	0.0016
CV (m, b, l)	494 - 438 - 68	0.031
CV (m, l)	436 - 390 - 174	0.13
CV (b)	419 - 403 - 178	0.21
Modified RIPPER	497 - 497 - 6	0.51
CV (b, l)	444 - 438 - 118	0.64
CV (l)	275 - 391 - 334	~1
J-Measure	408 - 582 - 10	~1
Klösgen ($\omega = 0.4323$)	403 - 553 - 44	~1
Two-Way Support Variation	402 - 587 - 11	~1
Klösgen ($\omega = 0.5$)	395 - 580 - 25	~1
Correlation	392 - 584 - 24	~1
Linear Correlation	391 - 585 - 24	~1
Example and Counterexample Rate	375 - 622 - 3	~1
Odds Ratio	372 - 580 - 48	~1
Odd Multiplier	371 - 582 - 47	~1
Added Value	370 - 626 - 4	~1
Conviction	369 - 585 - 46	~1
Lift/Interest	369 - 626 - 5	~1
Certainty Factor	366 - 626 - 8	~1
Sebag-Schoenauer	366 - 587 - 47	~1
Yule's Y	365 - 630 - 5	~1
Information Gain	364 - 631 - 5	~1
Yule's Q	361 - 634 - 5	~1
One-Way Support	357 - 636 - 7	~1
Zhang	354 - 642 - 4	~1
Relative Risk	352 - 589 - 59	~1
Rel Cost ($c_r = 0.342$)	339 - 634 - 27	~1
Laplace Correction	325 - 575 - 100	~1
Leverage	316 - 680 - 4	~1
Two-Way Support	282 - 716 - 2	~1
Accuracy	259 - 731 - 10	~1
Piatetsky-Shapiro	259 - 741 - 0	~1
Cost ($c = 0.437$)	255 - 733 - 12	~1
Least Contradiction	254 - 735 - 11	~1
Gini Index	244 - 756 - 0	~1
Collective Strength	242 - 758 - 0	~1
Loevinger	237 - 761 - 2	~1
Cosine	217 - 783 - 0	~1
Jaccard	217 - 783 - 0	~1
F-score ($\beta = 0.5$)	217 - 783 - 0	~1

Table 3: Result of comparing each rule-finding method against the baseline method, m-estimate ($m = 22.466$) with beam search ($b = 5$). "CV (m, b, l)" means cross-validation was performed to find the best values of m, beam size, and max rule length. p-Values are from a paired-sample sign test.

GPU Acceleration

Our large-scale experiment is made feasible through the use of commodity GPUs, which provide considerable processing power and high memory bandwidth. To take advantage of the power, an algorithm must be structured to fit into a single instruction, multiple data (SIMD) paradigm. Each thread in a GPU performs the exact same operations, and essentially only the thread ID is unique. The thread ID is used to index into different parts of an array in global memory, so that each thread can be made to see different data.

For simplicity, we parallelize the algorithm by having each thread evaluate one rule at a time. This choice works well for medium and large problems, but does not fully utilize the GPU threads for small problems. As a rough rule of thumb, if the number of candidate rules for each rule length (number of features times beam size) is greater than 1,000, the GPU outperforms a pure CPU implementation.

Algorithm 1 shows a high-level overview of our approach to beam search on a GPU. The CPU loads data, generates candidate rules, and manages the beam. The GPU performs the computationally intensive task of comparing each candidate rule against each sample in the training data.

Algorithm 1 Simple GPU Beam Search

```
1: procedure BeamSearch
2:     Load $data$ into GPU global memory
3:     $beam[0] \leftarrow \{\}$                ▷ initialize with empty rule
4:     for $i = 1$ to $l$ do                ▷ $l$ = maximum rule length
5:         Prepare $rules$ by refining $beam[i-1]$
6:         EvaluateCandidates($rules$)
7:         $beam[i] \leftarrow$ Best($rules$, $b$)        ▷ $b$ = beam size
8:     end for
9: end procedure

10: function EvaluateCandidates($rules$)
11:     Copy $rules$ from host to GPU global memory
12:     for $rule \in rules$ do                ▷ one thread per rule
13:         $p = 0$                ▷ initialize covered positives
14:         $n = 0$                ▷ initialize covered negatives
15:         for each sample $s \in data$ do
16:             if $rule$ covers $s$ then
17:                 if $s$ is positive class then
18:                     $p \leftarrow p + 1$
19:                 else
20:                     $n \leftarrow n + 1$
21:                 end if
22:             end if
23:         end for
24:     end for
25:     Copy all $p$ and $n$ back to host
26:     Calculate interestingness measure for each rule
27: end function
```

The GPU code consists of copying all candidate rules of the same length to the GPU, evaluating the rules, and copying back the counts of n and p to the host. We benchmark the execution time of the GPU code compared to functionally equivalent multi-threaded CPU code. The GPU code runs 20 times faster on our largest datasets. These tests were run on a desktop computer with an Intel Core i7-3820 CPU and an NVIDIA GeForce GTX 980 Ti GPU.

Two GPU code optimizations further increase the speedup to 28x. The first is caching the rules under evaluation into shared memory, which has higher bandwidth than global memory. The second optimization is having a single thread per block load each sample and label into shared memory visible to the other threads. This reduces the number of threads reading from global memory.

Conclusion

We have shown that rule finding precision can be increased by jointly optimizing the search parameters for each individual dataset using grid search with cross-validation. We found the most statistically significant improvement over the baseline method when simultaneously choosing values for both m in the m-estimate and the beam size in beam search.

Our results did not show a benefit for tuning the maximum rule length. Instead, we got better results by leaving rule length practically unrestricted.

We showed that beam search can be accelerated using a commodity GPU, with our implementation achieving a 28x speedup over a multi-threaded CPU implementation.

Our work adapts traditional rule learning techniques to take advantage of dramatically increased modern processing power. Recall that in 1990, when Cestnik proposed the m-estimate, the fastest Intel CPUs were single core with clock speeds of only 50 MHz.[1]. In contrast, the GPU we tested has 2,816 cores, each clocked at 1,190 MHz. The näive way of taking advantage of the extra processing power would be to simply expand the extent of search with other parameters unchanged, which would likely lead to more false discoveries and lower out-of-sample precision. Our method uses the extra processing power to recognize and avoid settings that lead to over-fitting, thereby improving precision.

Acknowledgment

This work is supported by Chevron USA, Inc. under the joint project Center for Interactive Smart Oilfield Technologies (CiSoft), at the University of Southern California.

References

Agrawal, R.; Mannila, H.; Srikant, R.; Toivonen, H.; Verkamo, A. I.; et al. 1996. Fast discovery of association rules. *Advances in Knowledge Discovery and Data Mining* 12(1):307–328.

Cestnik, B. 1990. Estimating probabilities: A crucial task in machine learning. In *ECAI*, volume 90, 147–149.

Clark, P., and Niblett, T. Induction in noisy domains. Citeseer.

Clark, P., and Niblett, T. 1989. The CN2 induction algorithm. *Machine Learning* 3(4):261–283.

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

Dzeroski, S.; Cestnik, B.; and Petrovski, I. 1993. Using the m-estimate in rule induction. *Journal of computing and information technology* 1(1):37–46.

Fang, W.; Lu, M.; Xiao, X.; He, B.; and Luo, Q. 2009. Frequent itemset mining on graphics processors. In *Proceedings of the fifth international workshop on data management on new hardware*, 34–42. ACM.

Fürnkranz, J., and Flach, P. A. 2005. ROC nrule learningtowards a better understanding of covering algorithms. *Machine Learning* 58(1):39–77.

Fürnkranz, J.; Gamberger, D.; and Lavrač, N. 2012. *Foundations of Rule Learning*. Springer Science & Business Media.

Fürnkranz, J. 1999. Separate-and-conquer rule learning. *Artificial Intelligence Review* 13(1):3–54.

Geng, L., and Hamilton, H. J. 2006. Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)* 38(3):9.

Han, J.; Pei, J.; and Yin, Y. 2000. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, 1–12. ACM.

Janssen, F., and Fürnkranz, J. 2009. A re-evaluation of the over-searching phenomenon in inductive rule learning. In *SDM*, 329–340. SIAM.

Janssen, F., and Fürnkranz, J. 2010. On the quest for optimal rule learning heuristics. *Machine Learning* 78(3):343–379.

Jensen, D. D., and Cohen, P. R. 2000. Multiple comparisons in induction algorithms. *Machine Learning* 38(3):309–338.

Langdon, W.; Yoo, S.; and Harman, M. 2011. Non-recursive beam search on gpu for formal concept analysis. *RN* 11(18):1.

Lenca, P.; Meyer, P.; Vaillant, B.; and Lallich, S. 2008. On selecting interestingness measures for association rules: User oriented description and multiple criteria decision aid. *European Journal of Operational Research* 184(2):610–626.

Možina, M.; Demšar, J.; Žabkar, J.; and Bratko, I. 2006. *Why is rule learning optimistic and how to correct it*. Springer.

Quinlan, J., and Cameron-Jones, R. 1995. Oversearching and layered search in empirical learning. *breast cancer* 286:2–7.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine learning* 5(3):239–266.

Srinivasan, A., and Ramakrishnan, G. 2011. Parameter screening and optimisation for ilp using designed experiments. *The Journal of Machine Learning Research* 12:627–662.

Tan, P.-N.; Kumar, V.; and Srivastava, J. 2002. Selecting the right interestingness measure for association patterns. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 32–41. ACM.

[1] `http://www.intel.com/pressroom/kits/quickrefyr.htm#1990`