Right of Inference: Nearest Rectangle Learning Revisited

Byron J. Gao and Martin Ester

School of Computing Science, Simon Fraser University, Canada
{bgao, ester}@cs.sfu.ca

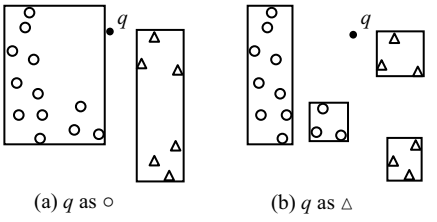
Abstract. In Nearest Rectangle (*NR*) learning, training instances are generalized into hyperrectangles and a query is classified according to the class of its nearest rectangle. The method has not received much attention since its introduction mainly because, as a hybrid learner, it does not gain accuracy advantage while sacrificing classification time comparing to some other interpretable eager learners such as decision trees. In this paper, we seek for accuracy improvement of *NR* learning through controlling the generation of rectangles, so that each of them has the *right of inference*. Rectangles having the right of inference are compact, conservative, and good for making local decisions. Experiments on benchmark datasets validate the effectiveness of the proposed approach.

1 Introduction

Nearest Rectangle (*NR*) learning [9] is a hybrid inductive learning approach, in which training instances are generalized into axis-parallel hyperrectangles, and a query is classified according to its nearest rectangle. If a query falls inside a rectangle, its distance to that rectangle is zero; if the query lies outside a rectangle, the distance is the (weighted) Euclidean distance from the query to that rectangle. If the query is equidistant to several rectangles, the smallest of which is chosen. The rectangles we mention in this paper are isothetic bounding boxes of the instances they contain, unless otherwise specified.

NR learners belongs to the class of hybrid lazy-eager learning algorithms. Lazy algorithms such as k-Nearest Neighbor (*kNN*) classifiers are instance-based and non-parametric, where the training data are simply stored in memory and the inductive process is deferred until a query is given. In contrast, eager algorithms such as decision trees, neural networks, and naive Bayes classifiers are model-based and parametric, where the training data are greedily compiled into a concise hypothesis (model) and then completely discarded. Obviously, lazy algorithms incur lower computational costs during training but much higher costs in answering queries also with greater storage requirements, not scaling well to large datasets. They do not generate interpretable models as some eager algorithms, in particular, decision trees can be directly inspected to understand the decision surfaces embedded in data even for non-technical end-users. This ease of comprehension is very appealing in decision support related data mining activities, where insight and explanations are of critical importance [2].

J. Fürnkranz, T. Scheffer, and M. Spiliopoulou (Eds.): ECML 2006, LNAI 4212, pp. 638–645, 2006.



In (a), the rectangles make "wild", inappropriate inferences and the query q is classified as ○;

In (b), the rectangles generalize the same training instances in a compact and conservative fashion, having the right of inference, and q is more appropriately classified as △.

Fig. 1. Right of inference

However, in terms of accuracy, lazy methods can be more advantageous. They do not lose information since all the training data are retained. They have additional information to utilize, the query instances, so that local and adaptive decisions can be made for predictions. On the other hand, eager methods try to make predictions that are good on average using a single global model.

To compromise on some of the distinguishing characteristics of purely lazy or eager methods, hybrid lazy-eager algorithms are studied. As an example, *NR* learning partially processes the training instances and generalizes them into hyperrectangles; these intermediate results are retained and used to answer queries. Nonetheless, the *NR* method has not received much attention mainly because it is considered not accurate enough. The original *NR* learning algorithm as well as several improved versions were experimentally compared with *kNN* [10,11], and it was concluded that the *NR* approach performed well in domains with axis-parallel decision boundaries; while in other occasions it was significantly inferior to *kNN* in terms of accuracy. Comparing to axis-parallel decision trees, which are essentially rectangle-based, the rectangles induced by *NR* learners also offer a level of intuitive interpretability. However, as a hybrid approach, *NR* is slower in answering queries; then with similar accuracy, it has no advantage over decision trees and this line of research discontinued soon after its introduction.

We revisit *NR* learning, and propose that the major reason accounting for its loss of accuracy in previous endeavors was that, the generalized rectangles were not given the *right of inference* that guarantees the appropriateness of rectangles in making inferences. In general, rectangles having the right of inference should be compact, conservative, and good for making local decisions, as illustrated in Fig. 1. By imposing the right of inference on rectangles, *NR* classifiers can potentially be intuitively explanatory, fast, scalable, yet highly accurate, having many combined appealing properties from decision trees and *kNN* classifiers.

1.1 Related Work

Decision trees [6] are typical eager learners while *kNN* classifiers [4] exemplify the simplest form of lazy learners. [1] identified the distinguishing characteristics of eager and lazy learners. Both types of learners have their own desirable properties. To obtain good trade-offs, varied hybrid approaches were proposed, e.g., [7] introduced a method combining instance-based and model-based learning.

As a hybrid approach, nearest rectangle learning was first introduced in [9] under the name of Nested Generalized Exemplar (*NGE*) theory. In *NGE*, an exemplar can be a generalized axis-parallel hyperrectangle or a single training instance, which is a degenerate (trivial) rectangle. Arbitrary overlapping and nesting of rectangles of different classes are allowed. [10,11] challenged the accuracy performance of *NGE* and made several improvement attempts such as disallowing nesting and/or overlapping, modifying the rectangle construction heuristic, and weighting features by mutual information. It was concluded that the major reason leading to the loss of accuracy of *NGE* was the overlapping of rectangles of different classes, yet the best improved version was still significantly inferior to *kNN* in most of the tested datasets. We notice that, all the above attempts did not pay much attention to the quality of the generated rectangles. They allowed rectangles to make wild and inappropriate inferences, which would significantly deteriorate the accuracy performance as illustrated in Fig. 1.

[5] studied cluster description formats, problems and algorithms, which also involved discriminative summarization of labeled data using hyperrectangles. But they considered only a two-class problem concerning objects in or not in the cluster. Moreover, their focus was on description (generalization) instead of classification (inference); the appropriateness of inference of rectangles was not an issue, but the conciseness of descriptions, i.e., the number of rectangles.

In the remaining of the paper, Section 2 discusses the concept of right of inference and its enforcement. Section 3 proposes LearnCovers, an *NR* learning heuristic. Section 4 presents empirical results and Section 5 concludes the paper.

2 Right of Inference and Its Enforcement

2.1 Right of Inference

Rectangle-based classifiers can provide certain degree of insight and understanding into data and the instance space. In fact, consider a closed rectangular instance space, the leaf nodes of an axis-parallel decision tree correspond to a set of isothetic rectangles (not bounding boxes) forming a partition of the instance space. The induction of decision trees generalizes the training data and makes inferences to the entire instance space simultaneously with the disjointness constraint, striving to achieve good-on-average predictions. Intuitively, if we separate generalization and inference into two serial phases and allow same-class rectangles to overlap, we should be able to build classifiers that are more flexible, adaptive and accurate, with the capacity to make local decisions.

Potentially, *NR* learning can induce such explanatory, adaptive and accurate classifiers. However, if in the generalization phase, the rectangles are not constructed in a conservative and compact fashion, they would make wild and improper inferences, similar to the case of decision trees, as demonstrated in Fig. 1 (a). It can be inspected that decision trees would make the same decision for the query in the figure. On the contrary, Fig. 1 (b) illustrates some compact rectangles for the same training data that are good for making local decisions, having the so-called right of inference.

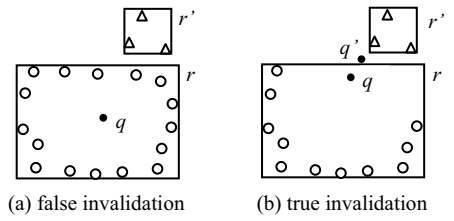


Fig. 2. For Definition 1

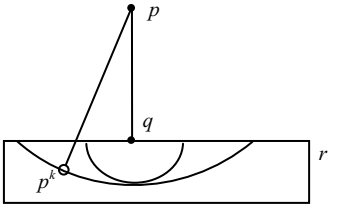


Fig. 3. For Theorem 1

The *right of inference* of a rectangle can be conceptually defined as the privilege that the rectangle has to make sound and local inferences. As we have seen, rectangles having the right of inference should appear compact and saturated. Then, how to define right of inference in a quantitative manner?

Definition 1. (*right of inference*) *A rectangle r has the right of inference if and only if for any query q outside of r, $dist(q, q^k) - dist(q, r) \leq \delta$, where $dist(q, q^k)$ is the Euclidean distance from q to its kth nearest instance in r, $dist(q, r)$ is the Euclidean distance from q to r, and δ is the distance threshold.*

The distance from q to r is equivalent to the line dropped perpendicularly from q to the nearest face, edge, or vertex of r, which is formally defined as follows, without considering rectangle weighting and feature weighting. Let q_{f_i} be the value of q on the ith feature, where $1 \leq i \leq m$; let r_{lower,f_i} and r_{upper,f_i} be the lower and upper end values of r on the ith feature, then:

$$dist(q, r) = \sqrt{\sum_{i=1}^{m} dif_i^2} \quad \text{where} \quad dif_i = \begin{cases} q_{f_i} - r_{\text{upper},f_i} & \text{when} \quad q_{f_i} > r_{\text{upper},f_i} \\ r_{\text{lower},f_i} - q_{f_i} & \text{when} \quad q_{f_i} < r_{\text{lower},f_i} \\ 0 & \text{otherwise} \end{cases}$$

What is the rationale behind the right of inference thus-defined? Note that, we always have $dist(q, r) \leq dist(q, q^k)$. If $dist(q, q^k) - dist(q, r)$ is too large, the inference on the class of q from r might be inappropriate, since $dist(q, r)$ would alter (bring closer) the locality of the instances in r with respect to q in an intolerable manner; e.g., in Fig. 1 (a), q is very close to the left rectangle, but far away from the instances in it. In contrast, if $dist(q, q^k) - dist(q, r)$ is reasonably small, *NR* classifiers would behave similarly to *kNN*, as shown in Fig. 1 (b).

In Definition 1, we only consider queries lying outside of the rectangle r. This is because some inside query may falsely invalidate a "good" r. In Fig. 2 (a), even if $dist(q, q^k) - dist(q, r)$ is rather large, r is good because q would not be closer to other instances/rectangles of different classes, say r'. Recall that overlapping of rectangles is allowed only if they have the same class label. On the other hand, for a "bad" r as shown in Fig. 2 (b), not considering q or other queries in r would not falsely validate r since if q should invalidate r (closer to r'), there would be another q' outside of r that also invalidates r. We can easily find such q', say, somewhere close to r and on the line joining q and r'.

In Definition 1, it is also reasonable to use the average of distances from q to its k nearest instances in r for $dist(q, q^k)$. k is limited by the number of

instances in r, and the choice of k can be a legitimate research issue just as in the case of *kNN* classification. The distance threshold δ has a direct impact on how closely *NR* classifiers would behave to *kNN*. If δ is too large, the rectangles tend to be very large as well making unconstrained inferences. If δ is too small, *NR* learning would induce too many rectangles and become "lazy", losing the desirable properties as a hybrid learner. In the extreme case of $\delta = 0$, *NR* learning would lose the generalization capacity completely and essentially become 1-*NN*.

2.2 Enforcing the Right of Inference

It is not straightforward to enforce the right of inference defined in Definition 1 since there are potentially infinite number of queries to examine. In the following, we discuss some inspiring observations and practical implications.

Theorem 1. *If for any query q that is on the surface of a rectangle r, $dist(q, q^k) \leq \delta$, where q^k is the kth nearest instance of q in r, then r has the right of inference defined in Definition 1 with respect to δ.*

Proof. Let p be any query outside of r. Let p^k be the *kth* nearest instance of p in r and q the projected p on the nearest face of r, as depicted in Fig. 3. According to the definition of point-to-rectangle distance, $dist(p, q) = dist(p, r)$. We use arc_p to denote the intersection of r and the sphere with radius $dist(p, p^k)$ centered at p, and arc_q to denote the intersection of r and the sphere with radius $dist(p, p^k) - dist(p, q)$ centered at q. Clearly, $arc_q \subseteq arc_p$.

Since p^k is the *kth* nearest instance of p in r, the number of instances in arc_p is less or equal to k if not considering ties. Since arc_p and arc_q intersect on only one point, the number of instances in arc_q is less or equal to k even considering ties. That is to say, q^k, the *kth* nearest instance of q in r, lies outside or on the surface of arc_q. In other words, $dist(q, q^k) \geq dist(p, p^k) - dist(p, q) = dist(p, p^k) - dist(p, r)$. Therefore, $\delta \geq dist(q, q^k) \Rightarrow \delta \geq dist(p, p^k) - dist(p, r)$, and the conclusion of Theorem 1 follows.

The implication of Theorem 1 is that, we only need to consider queries on the surface of r to test its right of inference. In the prototype *NR* learner LearnCovers, to be proposed shortly, a simple recursive testing and bisecting enforcement heuristic is embedded. For testing, the query pool consists of a constant number of queries generated according to a ranking scheme that gives high ranks to queries with high probability of invalidating r. Generally, highly ranked queries include vertices that are far away from the mean of the instances in r. Certain positions (say, centers) on long edges or large faces have the next priority to be inserted in the query pool, and then uncertain (random) positions on the surface of r. r is validated (passes the test) if it is not invalidated by any query in the query pool. If r is invalidated, the k-means clustering algorithm with $k = 2$ is applied to bisect the instances in r, and the newly generated rectangles (bounding boxes of the two sections) are tested separately. This recursive testing and bisecting process terminates until all the rectangles pass the test.

3 LearnCovers: Learning the "Right" Rectangles

LearnCovers heuristically constructs a set of rectangles with minimized cardinality and enforced right of inference. The rectangles generalize the given training instances with 100% accuracy, and same-class rectangles are allowed to overlap.

Algorithm 1. LearnCovers

```
1.  R = ∅; //R: the set of generated rectangles
2.  sort T; //T: the given training set
3.  for each t ∈ T { //process each training instance t in the sorted order
4.      for each r ∈ R {
5.          calculate cost(r,t); //r with smaller cost(r,t) is favored in covering t
6.          if (r.class != t.class && cost(r,t) == 0)
7.              validateToclose(r); //r can be closed only after validation
8.          if (r.class == t.class && r is not closed && cost(r,t) == 0)
9.              extend r to cover t and continue to process the next t; } //back to line 3
10.     for each r ∈ R { //t was not covered; fetch r in ascending order of cost(r,t)
11.         if (r.class == t.class && r is not closed && violationCheck(r,R) == no)
12.             expand r to cover t and continue to process the next t; } //back to line 3
13.     insert(R, r_new) }; //t cannot be covered; insert the trivial rectangle r_new to R
14. enforce(R);
```

The pseudocode is presented in Algorithm 1. R, the rectangle set, is initialized to be empty (line 1). Instances in the given training set T are sorted along a selected feature (line 2) and processed in the sorted order. For each training instance t (line 3), we search through R (line 4) for the best rectangle to accommodate it. Expanding rectangles would incur *covering violations*, i.e., overlapping of rectangles of different classes, which are not allowed. The best rectangle to cover t is the one with the smallest *covering cost* with respect to t, which is defined such that the number of generated rectangles can be minimized.

In line 5, the cost of r in covering t, $cost(r,t)$, is calculated. $cost(r,t) = 0$ only if t lies straightly under r, i.e., by simply extending r along the selected sorting feature, t will be covered by r. If $cost(r,t) = 0$ and r and t are of different classes (line 6), r is closed on condition that it can be validated; otherwise, r is bisected and the two propagated rectangles are inserted into R (line 7). Closed rectangles will not be considered in the remaining procedures, since they cannot be used to cover any further instances without causing violations.

If a non-closed r has the same class label as t with $cost(r,t) = 0$ (line 8), r is an optimal rectangle to cover t. We can simply stop searching and continue to process the next instance (line 9). Note that in this case, violation checking is unnecessary since instances in T are sorted and we only need to extend r along the sorting feature to cover t.

If t has not been covered by such an optimal r (line 10), we need to search through R for the best r with the smallest $cost(r,t)$. The rectangles in R will be considered in the ascending order of $cost(r,t)$, the first available one (line 11) will be used to cover t and we can continue to process the next instance (line 12).

If there is no such $r \in R$ that can cover t without incurring a violation, a trivial rectangle r_{new} for t will be constructed and inserted into R (line 13).

Upon reaching line 14, all the training instances in T have been processed and generalized. The enforcement heuristic discussed previously is applied to all non-closed rectangles in R (closed ones must have been validated), and all the recursively propagated rectangles will be inserted into R after validation. In the actual implementation, we have chosen $k = 1$ for testing the right of inference, that is, any query q on the surface of r with $dist(q, q^1) > \delta$ will invalidate r.

As for the choice of δ, we randomly sample a series of queries. For each query q, we record $dif_q = dist(q, t_q) - dist(q, \overline{t_q})$, where t_q is the nearest training instance of q and $\overline{t_q}$ is the nearest training instance of q that has a different class label from t_q. If we set $\delta = dif_q$, the resulting *NR* classifier behaves the same as 1-*NN* on q and q will not be assigned a class label other than the one of t_q. To see why, let r_{t_q} and $r_{\overline{t_q}}$ be any two rectangles covering t_q and $\overline{t_q}$ respectively, then $dist(q, r_{t_q}) \leq dist(q, t_q)$ and $dist(q, \overline{t_q}) - dist(q, r_{\overline{t_q}}) \leq \delta$ if $r_{\overline{t_q}}$ is enforced the right of inference, from which $dist(q, r_{t_q}) \leq dist(q, r_{\overline{t_q}})$ follows. Intuitively, since the right of inference is enforced on $r_{\overline{t_q}}$, $\overline{t_q}$ will not be brought close enough by the rectangular generalization to challenge the locality of t_q with respect to q. Note that t_q is also brought closer to q by r_{t_q}. After obtaining a series of dif_q's, we use the average value as δ. While how to decide δ deserves further investigations, a more practical situation would be, selecting δ so as to meet a given constraint on the maximum number of rectangles allowed.

The proposed *NR* learner LearnCovers is an extension of Learn2Cover [5], a discriminative summarization heuristic for labeled data, from 2 class to multi-class and with the right of inference enforcement mechanism embedded. Some related issues, such as selecting the sorting feature, handling ties, defining $cost(r, t)$ and so on, are discussed in [5] with more details.

4 Empirical Results

A series of experiments were conducted to evaluate the accuracy performance of the proposed *NR* learner LearnCovers. The notion of rectangle can be extended to tolerate categorical features but not in this prototype version; thus 20 numerical benchmark datasets without missing values from the UCI repository [3] were used to run C4.5 [8], 1-*NN*, *kNN* and LearnCovers. For *kNN*, the highest accuracy was recorded. The datasets were normalized on each feature. For each of the datasets where cross-validation was needed, the averaged result over 3 runs of stratified 10-fold cross-validation was taken.

In Table 1, "Att", "Ins" and "Cla" indicate the numbers of attributes, instances and classes respectively for the datasets. The results show that, LearnCovers outperforms C4.5 in 19, 1-*NN* in 12, and *kNN* in 8 of the 20 datasets. It has the averaged accuracy of 0.857, significantly higher than C4.5 (0.817), better than 1-*NN* (0.843) and comparable to *kNN* (0.864). Recall that, without considering the right of inference, but assisted by some other sophisticated techniques such as rectangle weighting and feature weighting using mutual information, the

Table 1. Accuracy: C4.5, 1-*NN*, *kNN*, and LearnCovers (LC)

Dataset	Att	Ins	Cla	C4.5	1-*NN*	*kNN*	LC	Dataset	Att	Ins	Cla	C4.5	1-*NN*	*kNN*	LC
balance	4	625	3	0.758	0.790	0.900	0.828	pima	8	768	2	0.737	0.701	0.738	0.752
bupa	6	345	2	0.655	0.632	0.652	0.672	satimage	36	6435	6	0.850	0.894	0.906	0.878
car	6	1728	4	0.917	0.917	0.951	0.938	segment	19	2310	7	0.960	0.974	0.974	0.966
ecoli	7	336	8	0.841	0.806	0.871	0.869	sonar	60	208	2	0.702	0.865	0.865	0.794
glass	10	214	6	0.687	0.701	0.712	0.739	spambase	57	4601	2	0.895	0.908	0.908	0.897
iono	34	351	2	0.900	0.869	0.869	0.937	vehicle	18	846	4	0.734	0.696	0.725	0.709
iris	4	150	3	0.953	0.953	0.967	0.973	vowel	10	990	11	0.788	0.989	0.989	0.973
letter	16	20000	26	0.868	0.955	0.955	0.925	waveform	21	5000	3	0.781	0.809	0.833	0.808
new-thyr	5	215	3	0.916	0.968	0.968	0.953	wine	13	178	3	0.936	0.949	0.972	0.977
page-blo	10	5473	5	0.965	0.957	0.959	0.971	yeast	8	1484	10	0.494	0.526	0.574	0.581
								Average				0.817	0.843	0.864	0.857

remedies proposed in [10,11] only achieved moderate accuracy improvement on the original *NR* learner, remaining "significantly inferior to *kNN*".

5 Conclusion

In this paper, we revisited *NR* learning, seeking for its accuracy improvement through imposing the right of reference on rectangles. Experiments on benchmark datasets demonstrated the effectiveness of the proposed approach. For future work, more effective and efficient testing and enforcement mechanisms should be investigated. Grounded on the right of inference of rectangles, there are several interesting directions to further extend *NR* learning. One is to consider k nearest (weighted) rectangles in classification; another is to consider creating a classifier ensemble with multiple rectangle sets that, for example, can be obtained from LearnCovers by varying the sorting feature.

References

1. D. Aha. Lazy learning. *Artificial Intelligence Review*, 11:7-10, 1997.
2. C. Apte and S. Weiss. Data mining with decision trees and decision rules. *Future Generation Computer Systems*, 1997.
3. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.
4. B.V. Dasarathy. Nearest Neighbor (NN) norms: NN pattern classification techniques. *IEEE Computer Society Press*, 1991.
5. B.J. Gao and M. Ester. Cluster description formats, problems, and algorithms. In *SIAM International Conference on Data Mining*, 2006.
6. S.K. Murthy. Automatic construction of decision trees from data: a multidisciplinary survey. *Data Mining and Knowledge Discovery*, 2(4):345-389, 1998.
7. J.R. Quinlan. Combining instance-based and model-based learning. In *ICML*, 1993.
8. J.R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.
9. S. Salzberg. A nearest hyperrectanhgle learning method. *Machine Learning*, 6:251-276, 1991.
10. D. Wettschereck. A hybrid nearest-neighbor and nearest-hyperrectangle algorithm. In *ECML*, 1994.
11. D. Wettschereck and T.G. Dietterich. An experimental comparison of the nearest-neighbor and nearest-hyperrectangle algorithms. *Machine Learning*, 19:5-27, 1995.