Linf: An L-infinity Classifier

Leland Wilkinson leland.wilkinson@systat.com
SYSTAT Inc.
225 W Washington St.
Chicago, IL 60606, USA

Anushka Anand aanand2@lac.uic.edu
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60606, USA

Dang Nhon Tuan tdang@cs.uic.edu
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60606, USA

Editor:

Abstract

We introduce a classifier based on the L^∞ norm. This classifier, called Linf, is a composition of four stages (transforming, projecting, binning, and covering) that are designed to deal with both the curse of dimensionality and computational complexity. Linf is not a hybrid or modification of existing classifiers; it employs a new covering algorithm. The accuracy of Linf on widely-used benchmark datasets is comparable to the accuracy of competitive classifiers and, in some important cases, exceeds the accuracy of competitors. Its computational complexity is sub-linear in number of instances and number of variables and quadratic in number of classes.

Keywords: Supervised classifiers, Decision trees, Support vector machines, Decision lists

1. Introduction

Linf is a classifier that was designed to address the curse of dimensionality and polynomial complexity by using projection, binning, and covering in a sequential framework. For class-labeled points in high-dimensional space, Linf employs computationally-efficient methods to construct 2D projections and sets of rectangular regions on those projections that contain points from only one class. Linf organizes these sets of projections and regions into a decision list for scoring new data points.

Linf is based on what we call Composite Hypercube Description Regions (CHDR). These composites (unions of weighted hypercubes called *rectangles*) can be used to define local and large-scale structures.

1.1 Composite Hypercube Description Regions (CHDRs)

While the union of open spherical balls is used to define a basis for the L^2 Euclidean metric topology, we can alternatively use balls based on other L^p metrics. For Linf, we employ the L^∞ or *sup* metric:

$$||x||_\infty = sup(|x_1|, |x_2|, \dots |x_n|)$$

when we search for neighbors. In this search, we are looking for all neighbors of a point at the center of a hypercube of fixed size in a vector space. Because we are concerned with finite-dimensional vector spaces in practice, we will use *max*() instead of *sup*() from now on.

A *hypercube description region* (HDR) is the set of points less than a fixed distance from a single point (called the *center*) using the L^∞ norm. A *weighted hypercube description region* is an HDR that uses the weighted L^∞ norm:

$$||x||_\infty = max(w_1|x_1|, w_2|x_2|, \dots w_n|x_n|).$$

We will assume the term HDR refers to this more general case from now on. And we will call these regions *rectangles*. We usually define weights locally, so that different points in a high-dimensional space can have different weights defining their hypercubes. This approach is similar to locally-weighted statistical models (e.g., adaptive kernels) that specify different variances in different regions of space.

By a *local structure*, we mean the points defined by a single hypercube (an L^∞ ball); these points are members of an HDR. By a *large-scale structure*, we mean the points defined by the union of two or more hypercubes. We call this large-scale structure a *composite hypercube description region* (CHDR).

For our application, CHDRs have the following useful properties:

- CHDRs are closed under union.
- Dense regions of point sets can be well-covered by CHDRs. This assertion is formalized and argued in Gao and Ester (2006).
- CHDRs are computationally simple and efficient.

CHDRs are defined for any number of dimensions. For scalability, we have limited them to two dimensions. Each Linf CHDR is defined on a 2D subspace of the training space. In order to classify a high-dimensional dataset, we must assemble a set of 2D projections and compute CHDRs on these projections. Figure 1 shows an example of a CHDR covering a 2D projection of the Orange10 dataset used in our tests later in this article.

1.2 Linf Constructs a List of CHDRs

Each CHDR constructed by Linf is based on a different random 2D projection. The algorithm is a one-against-all classifier (Dietterich and Bakiri, 1995). For each class C_k in a training set, we (a) compute a 2D projection, (b) bin the projected data values in a 2D rectangular segmentation, and (c) cover pure bins containing only instances of C_k with a CHDR. We iterate over classes until we are unable to find pure bins to classify remaining instances in the training set.

The result of this process is a list of CHDRs that can be used to score new data points. A point is assigned to the first CHDR in the list that contains it. If no CHDR contains the point, it is assigned to the closest CHDR in the list (using smallest point-to-rectangle L^∞ distance).

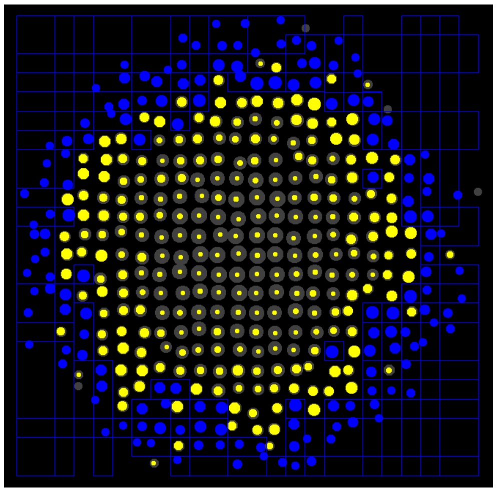

Figure 1: A CHDR covering class instances at the periphery of a projected spherical distribution. Data are the Orange10 dataset from Hastie et al. (2001). The data have been binned into a 24×24 grid with the size of each dot proportional to the count of instances in each bin. Blue dots represent pure class instances covered by the CHDR. Pure yellow dots represent pure class instances missed by the CHDR. Gray dots with yellow centers represent mixed-class bins.

2. Linf Training

The Linf training algorithm is summarized in Appendix A. We will describe its stages in detail in this section. We begin by reading n rows and p columns of a training dataset X. We code numerical values as double precision numbers and string variables as integers. We call the former continuous variables and the latter categorical variables. We delete any variable having only one value and decrement p by the number of deletions.

2.1 Transforming

Next, we transform our continuous variables with a nonlinear transformation. We compute skewness and kurtosis on each raw variable in order to decide whether to transform that variable. We standardize these skewness and kurtosis statistics using the square roots of their asymptotic variances ($6/n$ and $24/n$, respectively). Then we sort the two lists of standardized skewness and kurtosis values. We sequentially test each element in these lists by adjusting the 99th percent critical value drawn from the standard normal distribution in order to control the false discovery rate (Benjamini and Hochberg, 1995). If a standardized skewness or kurtosis value exceeds this adjusted critical value, we apply our transformation.

Our transformation is a folded square root:

$$t(x) = sgn(x)sqrt(abs(x))$$

This flexible transformation accommodates instances such as non-negative, positively-skewed data based on log-normal, Poisson, gamma, or exponential processes (counts, incomes, etc.), as well as long-tailed data from double-exponential, Cauchy, and similar distributions. Because our goal in transforming is simply to improve our chances of discovering margins between classes in dense regions, we do not need to concern ourselves with optimality at this stage, as in the Box-Cox transformation (Box and Cox, 1964). After transforming, we rescale continuous variables to the unit interval.

The next three stages of the Linf algorithm, described in the following subsections, are iterated over classes until we are unable to classify remaining data. Each iteration is a one-against-all classification step involving the current class vs. other classes.

2.2 Projecting

We will be binning 2D projections of variables in the hope of locating dense and well-separated class distributions. To do this, we generate a candidate list of random projections and then pick the best projections (based on a separation measure for the current class) for binning. Before projecting, we need to scale categorical variables (to project them into the same subspace as continuous variables) and we need to select variables (if the number of variables is large).

2.2.1 Scaling Categorical Variables

First, we scale categorical variables to numerical values. We use a strategy derived from the latent class model (Lazarsfeld and Henry, 1968). For a given categorical variable, we count the unclassified instances of the current class in each category. We divide this count by the total count of unclassified instances in each category. Finally, we replace integer category values with the corresponding proportions based on these two counts.

2.2.2 Selecting Variables

If p is large ($p > 50$), we select a subset of the variables for projection. This is a common preprocessing stage for classifier algorithms that must deal with high-dimensional data (Guyon and Elisseef, 2003). Our approach differs in two important respects, however. First, we rank our variables on *each* iteration (i.e., for each new current class). Second, we use a peculiar separation statistic to rank our variables.

Our separation statistic is the distance of the current-class mean from the closest other-class mean:

$$S = \min_{k \neq c}(d_{\bar{x}_c, \bar{x}_k})$$

We considered alternative separation statistics. The most obvious is the Fisher F-statistic based on the mean-square between classes and the mean-square within classes. We also considered a t-statistic measuring separation of the current class from all other classes (Welch, 1947). And we investigated a statistic measuring the overlap of the range of the current class values and the range of other-classes values (as a simple margin criterion). None worked as well as the one we selected. Furthermore, our separation statistic requires only the computation of group means; it is fast and economical in storage resources.

Our choice of the 50-variable limit was empirical. We varied this value across real and artificial datasets and found performance declined in general for other values. For specific datasets, of course, this rule could be adjusted to improve performance. We did not do so in this article.

2.2.3 Generating Projections

Next, we generate 25 random 2D projections (as with the 50-variable selection rule, this value was determined empirically). Each is constructed from a random 1D projection for x and for y. For each 1D projection, we generate p projection weights, of which r are zero. We evaluate this projection using the same separation statistic S that we used for selecting variables. This involves computing a candidate projection on the class means and then computing S on the projection. We do this three times (for $r = p/4$, $r = p/2$, and $r = 3p/4$) and choose the projection having the largest value of S.

Some of our projection weights are negative. We determine how many weights, and which weights, are negative by running a few iterations of simulated annealing (Kirkpatrick et al., 1983). We discovered that using this stratified algorithm (weights based on 3 values of r and a few annealing iterations on each) yields higher values on S than spending comparable time to generate p weights randomly over many iterations to find a best set.

We unit-weight our projections. This means we generate three-valued (-1, 0, 1) projection matrices. This approach would seem to follow a recent finding that using "database-friendly" unit weights instead of Gaussian weights does not result in a substantial loss of accuracy in approximating distances (Achlioptas, 2001). Li et al. (2006) extend this finding by proving that unit random weights for most purposes can safely be made "very sparse" with the following probabilities:

$$w_j = \begin{cases} 1 & \text{with probability} \quad \frac{1}{2\sqrt{p}} \\ 0 & \text{with probability} \quad 1 - \frac{1}{\sqrt{p}} \\ -1 & \text{with probability} \quad \frac{1}{2\sqrt{p}} \end{cases}$$

Efficiency in accessing a database is not our primary goal, however. Instead, our purpose in using unit weighting is to improve robustness in scoring new samples. This rests on a proof by Howard Wainer that draws on a result of Wilks (1938). Wainer proved that, under very general circumstances, a prediction model based on replacing OLS regression coefficients with unit weights will result in smaller expected loss in new samples and greater robustness against outliers than a model based on the regression coefficients themselves (Wainer, 1976).

The assumption underlying Wainer's proof is that the variables are unit-scaled and that the population values of the regression coefficients have a relatively restricted range. This fits our application because we work only with data standardized in the unit interval. In effect, Wainer is saying "eliminate small coefficients and set the rest to 1 (or -1 if the relationship with the criterion is negative)."

2.3 Binning

The next step in the process is to bin currently unclassified instances into a 2D bin matrix for each of our 25 chosen 2D projections. We base the number of bins on a formula in

Sturges (1926). Given n instances, we compute the marginal number of bins b using

$$b = 2\ log_2(n)$$

This formula produces a few more bins than optimal statistical estimates for binning normal and mildly skewed distributions (Scott, 1979; Wand, 1997). Traditional methods assume a homogeneous distribution, which is clearly not the case in classification.

Next, we rank our $b \times b$ 2D bin matrices on a purity measure. For a given target class C_k, our purity measure is

$$P_k = \sum_{i=1}^{b} \sum_{j=1}^{b} n_{i,j} I_{i,j}(C_k)$$

where

$$I_{i,j}(C_k) = \begin{cases} 1 & n_{i,j} = n_{i,j,k} \\ 0 & otherwise \end{cases}$$

In other words, we sum the counts across all bins whose total counts of points falling in them ($n_{i,j}$) are due only to class C_k counts ($n_{i,j,k}$). We want our purity measure to count only pure bins, because our fitting method will be especially greedy. The more pure bins we can eliminate early in the process, the better chance we have of seeing well-separated other classes later.

To recapitulate our current status: we have generated 25 2D projections based on our separation measure and we have chosen the top 5 of these 25 based on our bin purity measure. We now will cover these top 5 with rectangles and pick the cover that most improves our training-set classification.

2.4 Covering

The last stage in each iteration involves covering pure bins in order to define a classification region for a given class C_k. Our cover is a CHDR, which is a list of HDRs. Each CHDR is uniquely associated with a class label.

2.4.1 Growing a CHDR

We grow a CHDR on a given 2D bin matrix with a recursive algorithm. Figure 2 shows how this process works. For a given pure bin element $b_{i,j}$, we grow an HDR covering the bin and its pure neighbors by expanding upward ($b_{\cdot,j+1}$), rightward ($b_{i+1,\cdot}$), downward ($b_{\cdot,j-1}$), and leftward ($b_{i-1,\cdot}$) in a spiral path. In other words, we sequentially expand each side of the current rectangle by one pure bin-row or bin-column whose length is equal to the length of that side. We cease expanding in any of the four directions when we hit an edge of the 2D array or encounter a bin that is not pure. This strategy results in squarish rectangles that cover only pure or empty bins. Agrawal et al. (1998) use a similar method for clustering, but they follow a different computational path (up, down, left, right) that results in thinner rectangles.

We grow an HDR for each of the bins in the 2D bin matrix. For each HDR we record the number of instances of the current class that we have covered. We pick the HDR that results in the largest current-class count minus a penalty based on the count of other

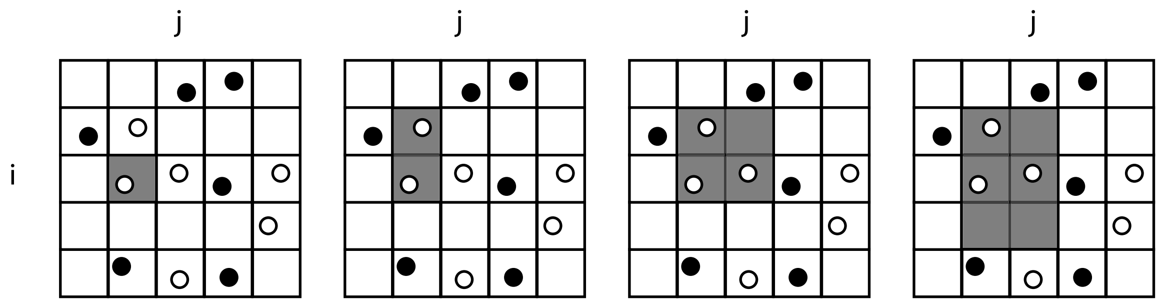

Figure 2: Growing a Hypercube Description Region (HDR) on a binned 2D projection. Each point is located at the centroid of the instances in each cell. Hollow symbols represent bins containing only instances of the current class. Solid symbols represent bins containing at least one instance of another class.

class instances immediately surrounding the HDR (we prefer an HDR that has a margin of relatively empty space around it). Finally, If the current-class count in the HDR exceeds 10, we add the HDR to the current CHDR list for that 2D projection.

This 10 is not a magic number. It is based on a rule-of-thumb for a slippage test. Tukey (1959) wrote:

> Given two groups of measurements, taken under conditions (treatments, etc.) A and B, we feel the more confident of our identification of the direction of difference the less the groups overlap one another. If one group contains the highest value and the other the lowest value, then we may choose (i) to count the number of values in the one group exceeding all values in the other, (ii) to count the number of values in the other group falling below all those in the one, and (iii) to sum these two counts (we require that neither count be zero). If the two groups are of roughly the same size, then the critical values of the total count are, roughly, 7, 10 and 13, i.e. 7 for a two sided 5% level, 10 for a two sided 1% level, and 13 for a two sided 0.1% level.

Our application fits this description because we construct an HDR to cover only instances outside the range of other-class instances. There are some caveats, of course. Our count of other-class instances is often substantially greater than the count of current-class instances inside an HDR; Tukey's approach assumes relatively balanced sample sizes. Second, we work in 2D; Tukey worked in 1D. Third, we count only highest values; Tukey counted highest and lowest. Tukey discusses several adjustments to deal with these problems, but we found little need to employ them since our method biases the test in a conservative direction. See Mosteller (1948) for more information on slippage tests.

We have so far described the procedure for computing a single HDR. Once we compute an HDR, we mark bins that it covers. Then we iterate this procedure over the 2D bin starting with uncovered bins until we can find no HDRs that meet Tukey's criterion. The result is a CHDR for a 2D bin matrix.

2.5 Unclassified Classes

It is possible that Linf will terminate on some datasets without constructing a rectangular cover for one or more classes. This happens rarely, but when it does, we relax our Tukey count criterion and classify each remaining unclassified class with the best CHDR we can find. We continue decrementing from 10 until we hit 0 or cover every unclassifed class at least once.

3. Linf Scoring

To score a new instance, we transform and rescale a new point. Then we pass through the list of CHDRs. For each CHDR, we project the point using the stored projections from the training data. Then we pass through the list of rectangles for that CHDR. The first rectangle to enclose our projected testing point determines the classification.

This scoring structure comprises a decision list (Rivest, 1987). Unlike trees, decision lists do not require traversal of the entire depth in order to score new instances (unless, of course, a cover is not encountered).

If no enclosing rectangle is encountered by the end of the list, we assign the point to the nearest rectangle in the CHDR list. This computation involves finding the shortest L^{∞} distance between a point and a rectangle. Because the perimeter of a CHDR is a zero level set for a naive density estimator based on the union of rectangular polygons (Silverman, 1986), this point-to-rectangle distance is asymptotically a nearest-neighbor statistic.

4. Performance

We will discuss in this section two different aspects of Linf performance: accuracy and efficiency. We tested Linf on ten datasets from the UCI Machine Learning Repository (Asuncion and Newman, 2007) and Hastie et al. (2001). We selected these particular datasets for their structural variety; each represents a different challenge for classifiers. Table 1 summarizes the datasets.

- Adult (Kohavi and Becker, 1996) is a dataset extracted from a 1994 US Census database. The challenge of this dataset is its mixture of categorical and continuous variables with skewed marginal distributions.

- Cancer (Ramaswamy et al., 2001) is a microarray dataset. The challenge of this dataset is its oversquare shape ($n << p$).

- Madelon (Guyon et al., 2007) is an artificial dataset constructed for the NIPS2003 feature-selection contest. The challenge of this dataset is the embedding of a low-dimensional signal in a high-dimensional space.

- Optdigits (Alpaydin and Kaynak, 1998) involves pixel values from images of hand-written digits. The challenge of this dataset is its circular covariance structure and highly non-normal marginal distributions.

- Orange10 (Hastie et al., 2001) is an artificial dataset designed by the authors to thwart support vector machines. Its challenge lies in having a target class completely enclosed in a high-dimensional shell of other-class points, so classes are not separable by hyperplanes. See Figure 1 for a graphic illustration of this problem.

- Satellite (Srinivasan, 1992) is one of the datasets used in the European Statlog project to evaluate classifiers. The challenge of this dataset is its relatively non-normal class distributions and severely nonlinear between-class margins.
- Shuttle (Catlett, 2002) is another Statlog dataset. The challenge of this dataset is its widely varying frequency distributions; approximately 80 percent of the data belong to class 1. The marginal distributions are also unusual; many look like double-exponentials.
- Spect (Kurgan et al., 2001) consists of binary features extracted from Single Proton Emission Computed Tomography (SPECT) images. The challenge is the completely binary structure of the dataset.
- Vowel (Deterding, 1989) involves speaker-independent recognition of the eleven steady-state vowels of British English. This is a popular benchmark dataset for neural network algorithms.
- Waveform (Breiman et al., 1984) is an artificial dataset. It was devised to illustrate shortcomings of tree classifiers.

The competitive classifier performance statistics were culled from papers referenced on the UCI site and from references in Hastie et al. (1993). They are based on almost 20 classifiers, including classification trees, margin trees, SVMs, kernel classifiers, linear, quadratic and logistic discriminant analysis, KNNs, naive Bayes, Bayesian nets, Kohonen Maps, AdaBoost, and neural nets.

Table 1: Characteristics of Datasets

	Training	Testing	Attributes	Groups	Cat Vars	Con Vars
Adult	32,561	16,281	14	2	Yes	Yes
Cancer	144	54	16,063	14	No	Yes
Madelon	2,000	600	500	2	No	Yes
Optdigits	3,823	1,797	64	10	No	Yes
Orange10	5,000	50,000	10	2	No	Yes
Satellite	4,435	2,000	36	6	No	Yes
Shuttle	43,500	14,500	9	7	No	Yes
Spect	80	187	22	2	Yes	No
Vowel	528	462	10	11	No	Yes
Waveform	300	500	21	3	No	Yes

4.1 Accuracy

The left panel of Figure 3 shows the error rates of Linf and the other classifiers. The other classifiers are unlabeled to reduce clutter. No classifier is best over all of the datasets, but Linf is competitive with the other classifiers.

There are several remarkable findings in this plot. First, Linf does relatively well on the cancer dataset. The simple feature-selection algorithm using the separation index appears effective. This strategy resembles the common practice of computing t-statistics on genes in microarray research. Of course, this strategy would not be as effective in high-dimensional datasets involving higher-order covariances among variables. Second, Linf does well on the orange10 dataset, outperforming support vector machine results reported in Hastie et al. (2001). Third, Linf achieves some of the lowest errors for the Shuttle dataset.

Linf does relatively poorly on the Vowel dataset, however. We are unable to find an explanation for this performance, since Vowel is amenable to several other classification strategies.

The most remarkable aspect of these results is that Linf produced them with no tuning for the specific datasets. Some of the results from other classifiers shown in Figure 3 were produced through data-based estimates of kernel families, bandwidths, pruning schedules, and neighborhoods. In some cases, these parameters were optimized by hand. Linf has no dataset-dependent parameters.

4.2 Efficiency

The right panel of Figure 3 shows the time Linf took to compute these classifications on the training datasets.

4.2.1 Complexity

Linf makes one pass through n rows of the training data to compute data limits and basic statistics. For each of the g classes, it makes an additional pass through the data to construct 25 2D bin matrices. Linf sorts this bin-matrix list and picks the top 5 candidate 2D bin matrices. It iterates through this process t times, adding a CHDR to the decision list at each step. For the test datasets in Figure 3, t varied approximately between 2 and 100. Thus, we should expect Linf to be $O(npgt)$ in time. To test this expectation, we did a simulation.

4.2.2 A Simulation

We generated spherical Gaussians for $n = \{100, 1000, 10000\}$, $p = \{20, 40, 60, 80\}$, and $g = \{2, 4, 6, 8\}$. In each of the 48 datasets, the first g Gaussians had unit variance with centroids located at the corners of a $(g-1)$-simplex with edges of length 7. Values for the remaining $p - g$ variates were $N(\mathbf{0}, \mathbf{I})$.

Figure 4 shows a graph of the performance of Linf on these random datasets. We have enhanced the plot with a distance-weighted least-squares smoother. The points are fit well ($R = .978$, with well-behaved residuals) with the simple linearized model:

$$E[\log(t)] = -10.491 + 0.847\log(n) + 2.436\log(g) + 0.816\log(p)$$

Our empirical results show that Linf is sub-linear in n and p but it is super-quadratic in g. It would appear that Linf is not a good candidate for problems involving hundreds or thousands of classes.

For small g, at least, Linf performance is similar to that of k-means clustering, which is $O(npgt)$ on n cases, p variables, g clusters, and t iterations. By contrast, optimal classi-

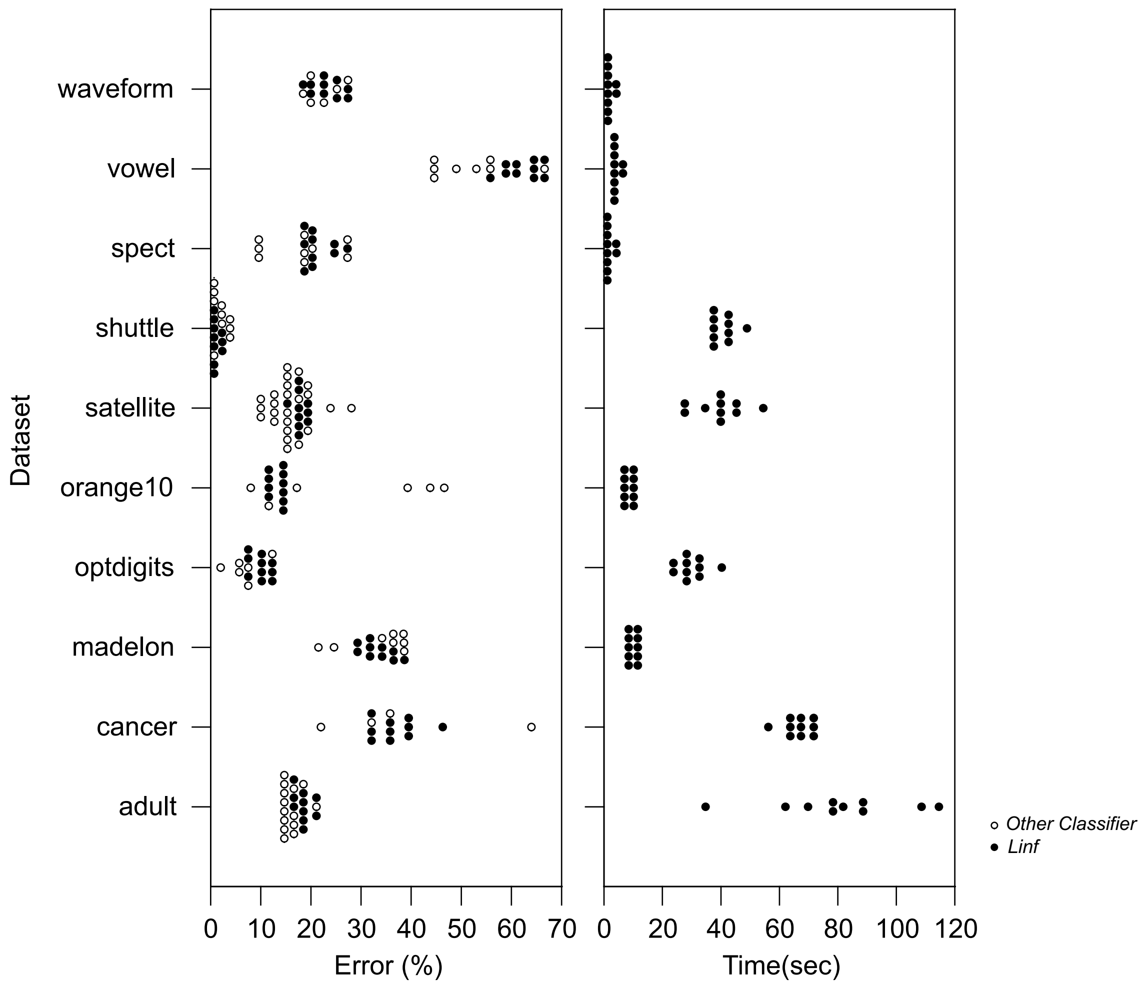

Figure 3: Dot plot of test dataset errors and training times for Linf and competitive classifiers. Hollow dots represent errors reported in selected publications from the UCI Machine Learning Dataset Repository and Hastie et al. (2001). Solid dots represent performance for the Linf classifier using 10 different random number seeds. Linf was run on a 2.5 GHz Intel Core 2 Duo Macintosh Powerbook with Macintosh OS X Version 10.5.7, 4 GB of RAM, and Java Version 1.5.0. Indistinguishable values are represented by a vertical stack of dots.

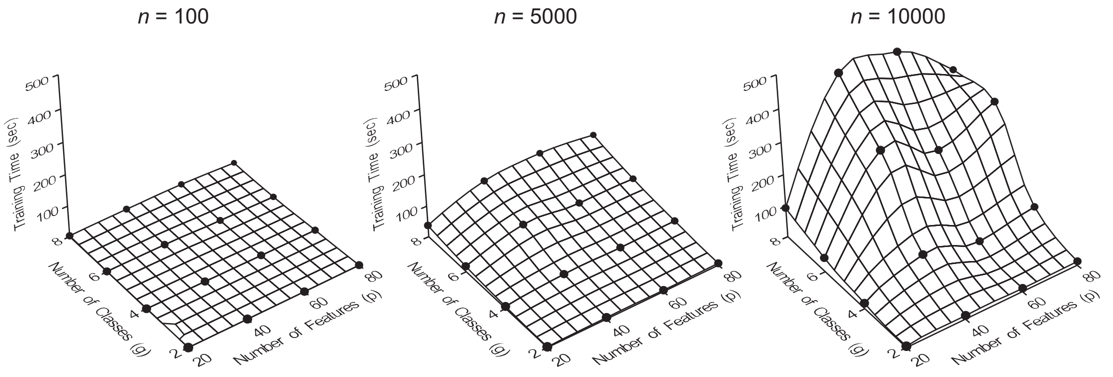

Figure 4: Training times for Linf on random datasets. The parameters in these plots are number of classes (g), number of features (p), and number of instances (n).

fication trees are NP-complete (Hyafil and Rivest, 1976)), and involve exponential time for categorical variables. Classification tree programs like CART and C4.5 are approximately $O(n^2log(n)p)$ when there are no categorical variables (Duda et al., 2000). SVMs are usually $O(max(n,p)min(n,p)^2)$, although computational algorithms vary (Chapelle, 2007). KNN methods are $O(n^2p)$. The n^2 terms in some of these methods can be reduced to $n\ log(n)$ by algorithmic design, although KNN algorithms are impractical for high-dimensional spaces because of the curse of dimensionality.

5. Ensemble Linf

Because Linf generates random projections, it is a natural candidate (like Random Forests) for building an ensemble classifier. This section discusses the performance of Ensemble Linf. We also take the opportunity to compare Ensemble Linf's performance on the same machine (Macintosh) and language (Java) with four classifiers in the Weka library (Waikato Machine Learning Group, 2009).

Ensemble Linf consists of running Linf five times and instituting an equal-weight, majority-voting procedure to score new cases. Figure 5 shows a labeled dot plot of the performance of Ensemble Linf vs. four popular classifiers – Naive Bayes, Random Forests, Support Vector Machine, and Classification Tree. Linf and Random Forests have the lowest errors on these datasets. Not surprisingly, Linf and Random Forests have different best-and-worst datasets. They are fundamentally different algorithms.

The right panel of Figure 5 shows the training times. In several cases, Linf is slower than the other classifiers by an order of magnitude. This result is not necessarily inconsistent with the $O()$ calculations in the previous section. A time-multiplier is not evidence of parameter complexity. Furthermore, in two cases (Adult and Madelon), the Support Vector Machine is an order of magnitude slower than Linf.

Figure 6 summarizes the error performance in the left panel of Figure 5. The box plots in this figure show error scores standardized within datasets (to remove dataset-specific variance). Boxes are sorted by median score. Linf shows the best overall median performance, although contrasts with the other classifiers are not statistically significant (a

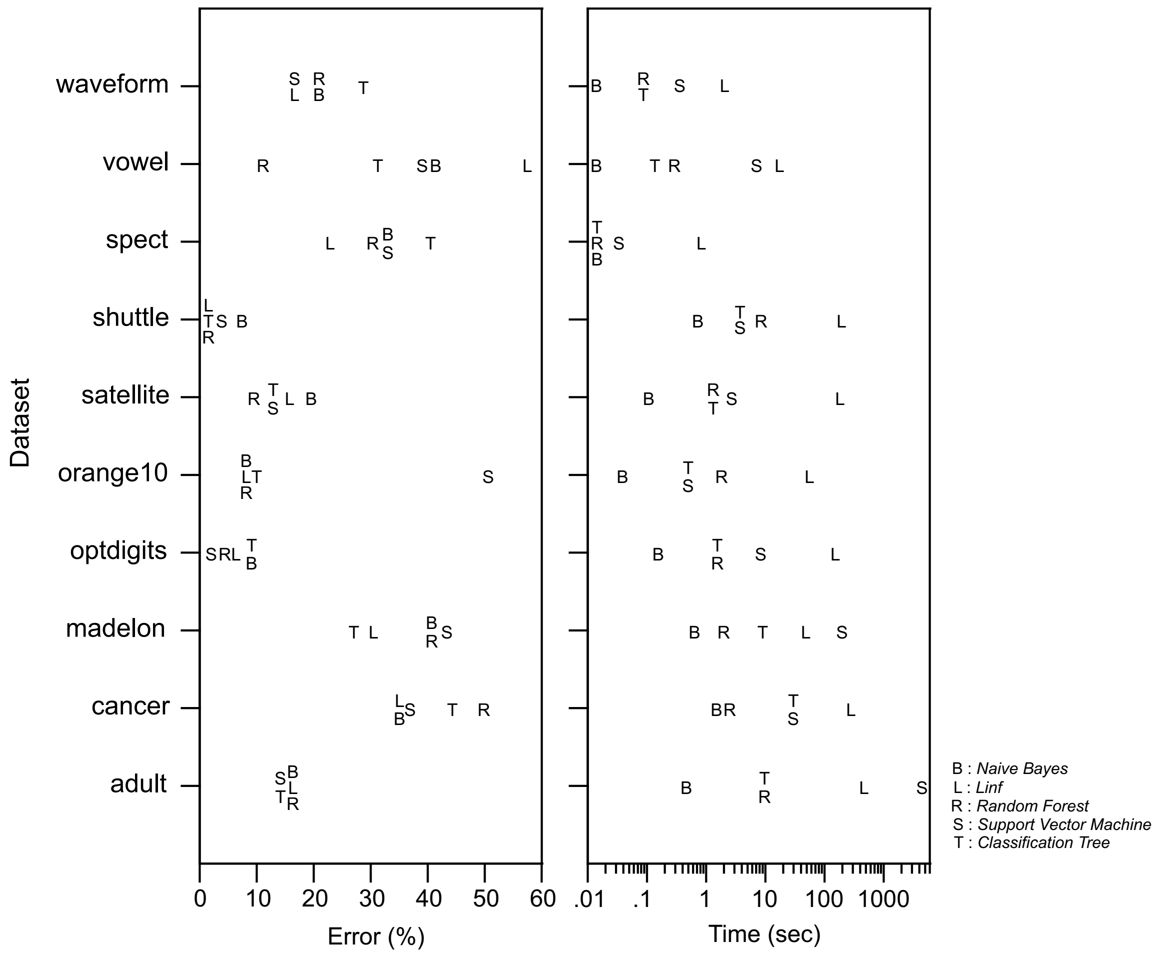

Figure 5: Plot of test dataset errors and training times for LINF and four competitive Weka classifiers (Naive Bayes = weka.classifiers.bayes.NaiveBayes, Linf = Ensemble LINF Classifier, Random Forests = weka.classifiers.trees.RandomForest, Support Vector Machine = weka.classifiers.functions.SMO, Trees = weka.classifiers.trees.J48). All software was run on a 2.5 GHz Intel Core 2 Duo Macintosh Powerbook with Macintosh OS X Version 10.5.7, 4 GB of RAM, and Java Version 1.5.0. Default parameter values were used for all classifiers. Indistinguishable values are represented by a vertical stack of letters.

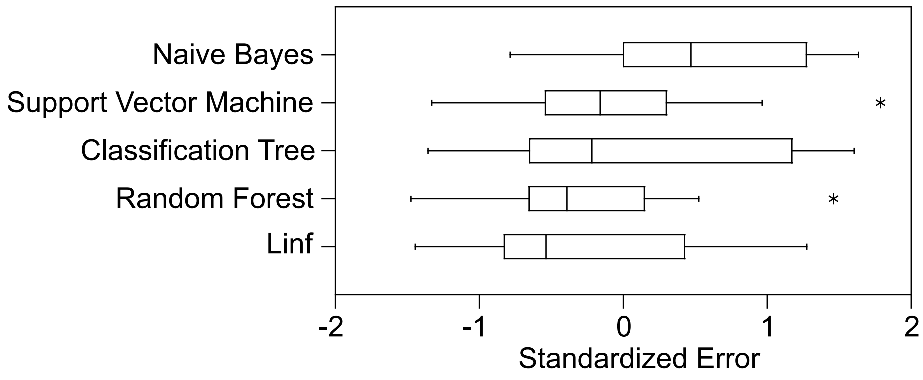

Figure 6: Grouped box plot of classification errors standardized within dataset.

Friedman repeated-measures rank test yields a p-value of .2). Linf and Random Forests would appear to be the best algorithms for classifying datasets in the absence of prior knowledge of their structure.

6. Related work

Perhaps the most widespread use of rectangular description regions is in recursive partitioning trees (Breiman et al., 1984; Quinlan, 1993). These methods partition a space into nested rectangular regions that are relatively homogeneous over the values of a predicted variable. Our approach differs from these models, however, because it is not restricted to a partitioning. Our description regions need not be disjoint or exhaustive.

Marchand and Shawe-Taylor (2002) and Sokolova et al. (2003) discuss decision lists for classification. They present a framework and a strategy for supervised classification, but do not deal with the high-dimensional computational complexity problem.

The GGobi team (Cook and Swayne, 2007) has worked on two aspects of this problem. First, they implemented rectangular brushing regions for visual classification. Their brushing operations were translated into SQL SELECT queries into a database. Second, they developed a projection-pursuit classifier (Lee et al., 2005). Others have followed similar lines, but failed to overcome the computational complexity of projection pursuit without resorting to parallel computation or similar measures (Flick et al., 1990; Jimenez and Landgrebe, 1995).

Although combinations of rectangles (unions and products of rectangles) have been used recently to define clusters in data mining (Agrawal et al., 1998; Bu et al., 2005; Gao, 2002; Gao and Ester, 2006; Pu and Mendelzon, 2005), ours is the first paper that we are aware of that uses rectangles efficiently in supervised classification.

7. Conclusion

Linf is not a "best of breed" classifier. Of course, there is no such classifier, unconditional to a particular process generating the data (Duda et al., 2000). Instead, Linf is an efficient classifier with accuracy comparable or superior to other classifiers commonly associated with benchmark datasets. Its virtues are:

- Categorical variables expend only one degree-of-freedom. We scale categorical variables on each iteration, so there is no dummy-coding to inflate dimensionality.
- The performance is linear in complexity on n or p. It is worse than quadratic on g (the number of classes), but solution times are practical for up to a hundred classes.
- Linf handles nonconvex, discrete and disjoint densities. Because Linf does not search for separating hyperplanes, it can cover a wide variety of 2D joint densities. Furthermore, Linf attacks higher-dimensional joint densities by peeling away regions containing pure class instances to reveal other pure regions normally obscured in lower-dimensional projections – a divide-and-conquer strategy. Finally, its categorical scaling algorithm allows us to combine discrete and continuous densities to search for homogeneous joint regions.
- Linf does not depend on sensitive adjustable parameters. We tested this assertion by assessing its performance over a wide range of parameter settings. Increasing *or* decreasing the 25-projection setting, for example, decrements performance on real and artificial datasets. The same is true for the 50-variable projection limit. We suspect this finding is related to bias-variance tradeoff – by limiting the search space probabilistically, we improve our chances of finding composites that do well in new samples (Friedman, 1997). Random Forests exploit similar devices to improve generalization error (Breiman, 2001).
- Linf is a novel algorithm; it is not a hybrid classifier. This fact would tend to support the idea that Linf can contribute relatively independent classification information to the results of other classifiers.

Given these distinctive features and its fundamental differences from other classifiers, Linf is a candidate for inclusion in portfolios of classifiers.

Acknowledgments

This work was supported by NSF/DHS grant DMS-FODAVA-0808860. Gerard Dallal, Jerome Friedman, Robert Grossman, John Hartigan, Trevor Hastie, and Robert Tibshirani made valuable suggestions.

Appendix A. Linf algorithms

Algorithm 1: **linf(X)**

```
Data: Training data set $X$ with $n$ instances, $p$ variables, and $g$ classes
Result: List of $CHDRs$ (Composite Hypercube Description Regions)
transformContinuousVariables ($X$)
$chdrList \leftarrow$ new List
$minCover \leftarrow 10$, $currentClass \leftarrow 0$, $nFailures \leftarrow 0$
while $nFailures < 2g$ do
    $nFailures \leftarrow$ processNextClass ($data$, $minCover$, $currentClass$, $chdrList$)
while there are unclassified classes and $minCover > 0$ do
    $minCover \leftarrow minCover - 1$
    processNextClass ($X$, $minCover$, $currentClass$, $chdrList$)
return $chdrList$
```

Algorithm 2: **transformContinuousVariables(X)**

```
foreach continuous variable in $X$ do
    compute standardized $skewness$ and $kurtosis$
sort $skewness$ and $kurtosis$ values
$k = 0$
foreach continuous variable in $X$ do
    $j \leftarrow$ column index of this variable, $n \leftarrow$ number of rows in $X$
    $k \leftarrow k + 1$
    $c_k \leftarrow$ Benjamini-Hochberg 99 percent critical value for $\Phi_k$
    if $skewness_k > c_k$ or $kurtosis_k > c_k$ then
        for $i \leftarrow 1$ to $n$ do
            $x_{i,j} \leftarrow sgn(x_{i,j}) sqrt(abs(x_{i,j}))$
```

Algorithm 3: **processNextClass(**X, *minCover*, *currentClass*, *chdrList***)**

```
increment ($currentClass$) // circular increment
$g \leftarrow$ number of classes in $data$, $p \leftarrow$ number of variables in $data$
scaleCategoricalVariables ($X$, $currentClass$)
$classMeans_{g \times p} \leftarrow$ class means for $p$ variables on currently unclassified instances
$classMeans \leftarrow$ selectVariables ($classMeans$, $currentClass$, 50)
$binList \leftarrow$ new List
$b \leftarrow 2\log_2$(number of currently unclassified instances) // number of bins
for $i \leftarrow 1$ to 25 do
    $projection \leftarrow$ generate2DProjection ($classMeans$, $currentClass$)
    $bin2D_{b \times b} \leftarrow$ binData ($projection$, $X$, $b$) // rectangular binning
    $bin2D.pureCount \leftarrow$ count of instances in bins that contain only $currentClass$
    append $bin2D$ to $binList$
sort $binList$ on $bin2D.pureCount$ values
$bestpureCount \leftarrow 0$
$bestCHDR \leftarrow$ null
for $i \leftarrow 1$ to 5 do
    $bin2D \leftarrow binList(i)$
    $CHDR \leftarrow$ coverBins ($bin2D$, $minCover$, $currentClass$, $b$)
    if $CHDR$ is null then
        $nFailures \leftarrow nFailures + 1$
    else
        $nFailures \leftarrow 0$
        $CHDR.pureCount \leftarrow$ count of $currentClass$ instances inside $CHDR$
        if $CHDR.pureCount > bestpureCount$ then
            $bestpureCount \leftarrow CHDR.pureCount$
            $bestCHDR \leftarrow CHDR$
if $bestCHDR$ is not null then
    append $bestCHDR$ to $chdrList$
return $nFailures$
```

Algorithm 4: **scaleCategoricalVariables(**X, *currentClass***)**

```
foreach categorical variable in $X$ over currently unclassified instances do
    $j \leftarrow$ column index of this variable, $m \leftarrow$ number of categories for this variable
    $counts_{1 \times m} \leftarrow$ counts of instances in each category
    $classCounts_{1 \times m} \leftarrow$ counts of instances of $currentClass$ in each category
    $classProportions_{1 \times m} \leftarrow classCounts_k / counts_k$ for each category ($k = 1, \ldots, m$)
    for $i \leftarrow 1$ to $n$ do
        $k \leftarrow x_{i,j}$ // data value points to category
        $x_{i,j} \leftarrow classProportions_k$ // now treat as continuous variable
```

Algorithm 5: **generate2DProjection**(*classMeans*, *currentClass*)

```
$g \leftarrow$ number of classes in $classMeans$, $p \leftarrow$ number of variables in $classMeans$
$maxDist \leftarrow 0$
$weights_{2\times p} \leftarrow$ null
for $i \leftarrow 1$ to 3 do
    $r \leftarrow p \times i/4$
    $wx_{1\times p} \leftarrow$ new array of $x$ weights
    $wy_{1\times p} \leftarrow$ new array of $y$ weights
    foreach $wt : wx, wy$ do
        randomly set $r$ elements of $wt$ to 1 and $p - r$ elements of $wt$ to 0
        for 5 iterations of simulated annealing do
            $wt \leftarrow$ negate a randomly chosen nonzero element of $wt$
            $projectedMeans_{g\times p} \leftarrow classMeans \times wt$
            $dist \leftarrow$ shortest distance from $currentClass$ projected mean to any other
            accept sign change probabilistically if $dist$ increases over last iteration
    if $dist > maxDist$ then
        $maxDist \leftarrow dist$ // record best of 3 weight vector candidates
        $weights \leftarrow wx, wy$
return $weights$
```

Algorithm 6: **coverBins**(*bin2D*, *minCover*, *currentClass*, *b*)

```
$CHDR \leftarrow$ new List
repeat
    $bestCount \leftarrow 0$
    for $i \leftarrow 1$ to $b$ do
        for $j \leftarrow 1$ to $b$ do
            $bin \leftarrow bin2D_{i,j}$
            if $bin$ is pure and not previously covered then
                $rectangle \leftarrow$ new Rectangle $(i, j)$ // loc, height, width
                repeat
                    expand $rectangle$ up one row into empty or pure cells
                    expand $rectangle$ right one column into empty or pure cells
                    expand $rectangle$ down one row into empty or pure cells
                    expand $rectangle$ left one column into empty or pure cells
                until cannot expand in any direction without hitting impure cells
                $count \leftarrow$ count instances of $currentClass$ inside $rectangle$
                $penalty \leftarrow$ count instances of $othertClass$ bordering $rectangle$
                if $count - penalty > bestCount$ then
                    $bestCount \leftarrow count$
                    $bestRectangle \leftarrow rectangle$
    if $bestCount > 0$ then
        append $bestRectangle$ to $CHDR$
until $bestCount = 0$
return $CHDR$
```

Algorithm 7: **selectVariables**($classMeans$, $currentClass$, q)

```
if number of variables in $classMeans < q$ then
    return $classMeans_{g \times p}$
foreach variable in $classMeans$ do
    find shortest distance from $currentClass$ mean to any other class mean
sort variables in $classMeans$ on shortest distances
return $classMeans_{g \times q}$ truncated to $q$ variables with longest shortest distances
```

References

Dimitris Achlioptas. Database-friendly random projections. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 274–281, New York, 2001. ACM.

Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD: International Conference on Management of Data*, pages 94–105, 1998.

E. Alpaydin and C. Kaynak. Optical recognition of handwritten digits. `http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits`, 1998.

A. Asuncion and D.J. Newman. UCI machine learning repository. `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 2007.

Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society, Series* B, 57:289–300, 1995.

G.E.P. Box and D.R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society, B*, 26:211–252, 1964.

L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, 1984.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

S. Bu, L. V. S. Lakshmanan, and R. T. Ng. MDL summarization with holes. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 433–444. VLDB Endowment, 2005.

Jason Catlett. Statlog (shuttle) data set. `http://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle)`, 2002.

Olivier Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5):1155–1178, 2007.

Dianne Cook and Deborah F. Swayne. *Interactive and Dynamic Graphics for Data Analysis: With R and GGobi (Use R)*. Springer, December 2007.

D. H. Deterding. *Speaker Normalisation for Automatic Speech Recognition.* PhD thesis, University of Cambridge, Cambridge, UK, 1989.

T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of AI Research*, 2:263–286, 1995.

R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* Wiley-Interscience Publication, 2000.

Thomas E. Flick, Lee K. Jones, Richard G. Priest, and Charles Herman. Pattern classification using projection pursuit. *Pattern Recognition*, 23:1367–1376, 1990.

Jerome H. Friedman. On bias, variance, 0/1 loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

B. J. Gao and M. Ester. Turning clusters into patterns: Rectangle-based discriminative data description. In *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, pages 200–211, Washington, DC, USA, 2006. IEEE Computer Society.

B.J. Gao. *Hyper-rectangle-based discriminative data generalization and applications in data.* PhD thesis, Simon Fraser University, 2002.

I. Guyon and A. Elisseef. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

Isabelle Guyon, Jiwen Li, Theodor Mader, Patrick A. Pletscher, Georg Schneider, and Markus Uhr. Competitive baseline methods set new standards for the nips 2003 feature selection benchmark. *Pattern Recognition Letters*, 28(12):1438–1444, 2007.

T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, New York, 2001.

Trevor Hastie, Robert Tibshirani, and A. Buja. Flexible discriminant analysis by optimal scoring. *Journal of the American Statistical Association*, 89:1255–1270, 1993.

L. Hyafil and R.L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5:15–17, 1976.

L. O. Jimenez and D. A. Landgrebe. Projection pursuit for high dimensional feature reduction: paralleland sequential approaches. In *Geoscience and Remote Sensing Symposium, 1995. IGARSS '95*, volume 1, pages 148–150, 1995.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

Ronny Kohavi and Barry Becker. Adult data set. `http://archive.ics.uci.edu/ml/datasets/Adult`, 1996.

L.A. Kurgan, K.J. Cios, R. Tadeusiewicz, M. Ogiela, and L.S. Goodenday. Knowledge discovery approach to automated cardiac spect diagnosis. *Artificial Intelligence in Medicine*, 23(2):149–169, 2001.

P.F. Lazarsfeld and N.W. Henry. *Latent structure analysis.* Houghton Mifflin, Boston, 1968.

Eun-Kyung Lee, Dianne Cook, Sigbert Klinke, and Thomas Lumley. Projection pursuit for exploratory supervised classification. *Journal of Computational and Graphical Statistics*, 14:831–846, 2005.

Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 287–296, New York, NY, USA, 2006. ACM.

Mario Marchand and John Shawe-Taylor. The set covering machine. *Journal of Machine Learning Research*, 3:723–746, 2002.

F. Mosteller. A k-sample slippage test for an extreme population. *The Annals of Mathematical Statistics*, 19:58–65, 1948.

K. Q. Pu and A. O. Mendelzon. Concise descriptions of subsets of structured sets. *ACM Transactions on Database Systems*, 30(1):211–248, 2005.

J. R. Quinlan. *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning).* Morgan Kaufmann, 1993.

S. Ramaswamy, P. S., Tamayo, R. Rifkin, S. Mukherjee, C. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. Mesirov, T. Poggio, W. Gerald, M. Loda, E. Lander, and T. Golub. Multiclass cancer diagnosis using tumor gene expression signature. *PNAS*, 98: 1514915154, 2001.

Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2:229–246, 1987.

D. W. Scott. On optimal and data-based histograms. *Biometrika*, 66:605–610, 1979.

B. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman & Hall, New York, 1986.

Marina Sokolova, Nathalie Japkowicz, Mario Marchand, and John Shawe-taylor. The decision list machine. In *Advances in Neural Information Processing Systems 15*, pages 921–928. MIT Press, 2003.

Ashwin Srinivasan. Statlog (landsat satellite) data set. `http://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite)`, 1992.

H.A. Sturges. The choice of a class interval. *Journal of the American Statistical Association*, 21:65–66, 1926.

J.W. Tukey. A quick, compact, two-sample test to Duckworth's specifications. *Technometrics*, pages 31–48, 1959.

Waikato Machine Learning Group. Weka, 2009. `http://mloss.org/software/view/16/`.

Howard Wainer. Estimating coefficients in linear models: It don't make no nevermind. *Psychological Bulletin*, 83(2):213–217, 1976.

M. P. Wand. Data-based choice of histogram bin width. *The American Statistician*, 51(1): 59–64, 1997.

B. L. Welch. The generalization of ”student's” problem when several different population variances are involved. *Biometrika*, 34:28–35, 1947.

S. S. Wilks. Weighting systems for linear functions of correlated variables when there is no dependent variable. *Psychometrika*, 3:23–40, 1938.