Feature Selection by Tree Search of Correlation-Adjusted Class Distances

Hassan Rabeti
Research and Development
vAuto Inc., Austin, Texas

Martin Burtscher
Department of Computer Science
Texas State University

***Abstract*—The rapidly growing dimensionality of datasets has made feature selection indispensable. We introduce the TS-CACD feature-selection algorithm, which uses a generalization of the Stern-Brocot tree to traverse the search space. This family of trees supports different divergence ratios, *i.e.*, enables the search to focus on and reach certain areas of interest more quickly. TS-CACD uses a continuous filter method, which combines an inter/intra-class distance measure with a pair-wise ranked feature correlation measure. It requires almost no parameters, explicitly selects the most important features, and performs well.**

I. Introduction

Datasets are rapidly increasing in size and complexity. This growth, especially in dimensionality, makes it progressively harder to discover important information and relationships. Hence, feature-selection algorithms that reduce this complexity are essential for the successful mining of non-trivial data.

In multidimensional datasets, each item has several attributes (called features). For example, a dataset of cars might include the make, model, color, *etc.* of a large number of cars, and we may want to classify which of these cars are likely to develop engine problems. Feature selection aims to minimize the number of features that need to be considered while minimally degrading the classification accuracy or even improving it. Thus, feature selection can be described as determining a combination of features (*i.e.*, a subset) that optimizes an evaluation function. This evaluation function takes into account the number of selected features as well as the classification accuracy, that is, the ability to predict the class of a given item from the dataset. In the car example, feature selection should eliminate all attributes that do not correlate with engine breakdowns, such as the cars' color.

Feature selection is widely used, including in hand-writing analysis [1], social media [2], diagnostic medicine and gene selection in micro-array data [3]. Each domain has its own set of preferences for the feature-selection algorithm. Some prefer to avoid manually optimizing parameters to fine tune the accuracy and instead want an algorithm that explicitly chooses a subset. Others favor algorithms with a complexity that is linear in the number of items in the dataset. The TS-CACD algorithm does not require any parameters, targets medium-sized data sets, and does not run in linear time but therefore considers the interaction between features. A number of excellent publications exist that summarize and compare many different feature-selection algorithms [4], [5], [6].

To successfully perform feature selection, two components are needed: (1) a method to determine subsets and (2) a measure to assess the quality of a subset. Our algorithm uses a tree-search (TS) approach to find subsets and the correlation-adjusted class distance (CACD) to evaluate their quality.

At its core, our feature selection process attempts to determine a scalar coefficient (*i.e.*, a weight) between zero and one for each feature to minimize the dimensionality while maximizing the classification accuracy. To achieve this goal, the weights are chosen to deemphasize redundancy among features by considering the correlation between them. Furthermore, the weights are selected such that items of the same class are placed close to each other whereas items from different classes are placed far apart, which promotes class locality and thus improves the accuracy of many classification algorithms.

For instance, in the car example, the mileage and the year typically correlate and are therefore indicative of the same chance of engine failure. Hence, the weights are selected to ensure that the generated feature list includes only one or the other, thus lowering the dimensionality while maintaining the same predictive power. In contrast, the license plate number generally is not indicative of engine failure and is hence not useful in the classification process or may even degrade it. Thus, this feature will be deemphasized (its weight will be small) whereas other features that do discriminate engine failures will be assigned correspondingly greater weights. To generate the final subset, our approach eliminates all features whose weights are insignificant, *i.e.*, close to zero.

To efficiently explore the search space for determining good sets of scalar coefficients, we introduce a novel tree-search approach that is a generalization of the Stern-Brocot tree. This tree represents a way to construct all rational numbers by starting with two fractions $(\frac{0}{1}, \frac{1}{0})$ and iteratively inserting the mediant between each two adjacent fractions. Thus, every iteration yields a refinement of the previous set of fractions. This construction generates a binary tree that contains every rational fraction exactly once and in reduced form. Figure 1 illustrates the first few levels of the Stern-Brocot tree [7].

By following the edges in this binary tree from the root, we can reach every fraction, *i.e.*, every possible weight assignment for two features. Similarly, in our k-dimensional generalization, we follow edges to explore the search space of the weights for k features. Note that descending down the tree corresponds to a refinement of the search space. In other

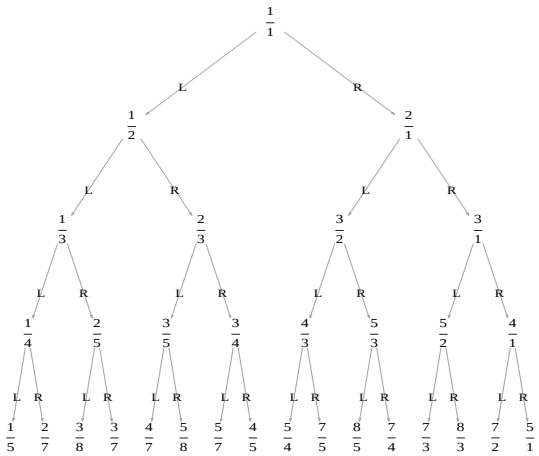

Fig. 1. Top five levels of the Stern-Brocot tree

words, every step narrows the range of possible weights.

The key operation of taking a left or right step in the Stern-Brocot tree can be expressed using the following matrices:

$$L = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \text{ and } R = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

Since multiplying by one of these two matrices corresponds to following an edge, a path in the tree is tantamount to the product of a sequence of these matrices. For example, starting at the root, going left, then left again, and then right, we end up at $\frac{2}{5}$. Using the matrix notation, we multiply L by L and then by R. The resulting fraction is the sum of the elements in the bottom row over the sum of the elements in the top row.

$$L \times L \times R = \begin{pmatrix} 3 & 2 \\ 1 & 1 \end{pmatrix} \longrightarrow \frac{1+1}{3+2} = \frac{2}{5}$$

The rest of this paper is organized as follows. Section II compares our approach to related work. Section III describes our correlation-adjusted class distance in detail. Section IV explains our tree-based search strategy. Section V discusses how we use the tree search in the context of our evaluation function. Section VI presents the results. Section VII concludes.

II. Related Work

A well-known filter method is Relief [8]. Relief and its variants rank features based on the number of near hits and misses in the original space. This is done by repeatedly selecting a random sample and measuring its distance to the nearest sample of the same class (hit) and the nearest sample of a different class (miss). The final result is a ranking of how relevant each feature is in predicting the class. The user then has to select the proper subset by deciding which features to keep or discard based on the ranked relevance.

The core principle, and a similarity to our approach, is that Relief assigns meaning to the distances between neighboring samples based on the distance of their classes. However, unlike our method, Relief measures these distances in the original space. This can be a problem because removing a feature potentially changes the distances, which might increase the relevance of another feature and rank it higher in the new space. Unfortunately, it is computationally infeasible to check every combination of features to find the optimal subset. Later we discuss how we address this issue using the tree search.

The feature ranks that Relief and similar algorithms provide are a guide to help the user select an appropriate subset. Whereas this allows the consideration of multiple subsets, it does not explicitly select features. Moreover, these choices are constrained to be 'near' each other in terms of possible subsets. In particular, by changing the cut-off point, the user can include more or fewer features, but he or she cannot try new combinations such as removing a higher ranked feature and adding two lower ranked features instead that, together, supersede the higher ranked feature. In contrast, our tree traversal produces explicit subsets from the entire range of possible subsets. This is an important yet often overlooked concept. A consequence of this restriction of Relief is that it has a lower asymptotic time complexity than our method.

Another filter approach that is related to ours is EUBAFES (EUclidean BAsed FEature Selection) [9]. It is similar in that it also uses a class distance measure and scalar feature weights to enable a continuous search. One key difference is that our method uses a correlation measure to balance the class distance measure and to promote a reduction in dimensionality. EUBAFES relies on parameters to stabilize its class distance measure. When switching from one dataset to another without adjusting these parameters, we found a similar design to often converge to one extreme of the subset size or to be dominated by the parameters rather than the data. In contrast, our approach only has two internal parameters that are designed to counterbalance each other, meaning that they work well across a range of datasets. Hence, the users of our approach do not have to tune parameters. Another major difference is that EUBAFES employs a Euclidean metric in the distance measure. We found an inverted sum, *i.e.*, the sum of the inverse distances, to result in better subset quality.

There are also feature selection algorithms that perform regularization on the feature weights (generally on binary weights) by penalizing larger feature subsets, which, in turn, can force a reduction in dimensionality. Whereas we also employ such a strategy, we do so by placing a constraint on our feature weights that can only force a reduction in dimensionality in the presence of the correlation adjustment.

III. Correlation-Adjusted Class Distance

In this section, we present the CACD criterion function and discuss the reasons for choosing this function.

A. Criterion

With Q denoting the number of features and N denoting the number of instances in a dataset, let our domain be $\Re = \{X_1, \ldots, X_N\} \in \mathbb{R}^{N \times Q}$, where $X_i = \{x_{i,1}, \ldots, x_{i,Q}\}$ for $1 \leq i \leq N$. Furthermore, let $C = \{c_1, \ldots, c_N\}$ be the set of class labels so that each c_i is associated with instance X_i for $1 \leq i \leq N$. Based on these inputs, we want to compute

a set $W = \{w_1, \ldots, w_Q\}$ of positive feature weights (scalar coefficients) that satisfy the L_2 constraint

$$\sum_{q=1}^{Q} w_q^2 = 1, w_q \geq 0 \tag{1}$$

We then define the distance $d_{i,j}$ between two instances X_i and X_j as a function of these feature weights

$$d_{i,j}(W) = \sum_{q=1}^{Q} w_q |x_{i,q} - x_{j,q}| \tag{2}$$

To also capture non-linear correlations between features, we use the Kendall Tau-b [10] distance, which is defined as

$$\tau_{i,j} = \frac{n_c - n_d}{\sqrt{n_1 n_2}} \tag{3}$$

where n_c is the number of concordant (correlated) pairs, n_d is the number of discordant (anti-correlated) pairs, n_1 is the number of pairs not tied (neither correlated nor anti-correlated) in X_i, and n_2 is the number of pairs not tied in X_j.

The above constraint on the weights and the two equations represent the fundamental components of our criterion. Constraint (1) acts as a regularization strategy by limiting the number of features selected. If some features have large weights, others must necessarily have small weights. Since we are only performing pair-wise correlation comparisons, we may end up with an over-approximation, especially for subsets that include more features. As a counterbalance, we inflate the score of larger subsets by using the L_2 (sum of squares) instead of the L_1 norm while reducing the score based on the correlation. Eq. (2) simply applies a weighted transformation and measures the L_1 distance between two samples. Finally, Eq. (3) measures the ranked correlation between pair-wise features without dependence on the linearity of the correlation. It does this by only considering the signs (but not the magnitude) of the values, thus making it unsusceptible to outliers.

The following defines the interaction between the above measures. Given a set of weights, we define an approximation of the general correlation between our features by

$$K(W) = \frac{Q(Q-1) - \sum_{i=1}^{Q-1} \sum_{j=i+1}^{Q} w_i w_j |\tau_{i,j}|}{Q(Q-1)} \tag{4}$$

To handle class imbalances, *i.e.*, inputs where most instances are of the same class, we define the frequency of the intra-class (equal or 'eq') and inter-class (not equal or 'ne') occurrences

$$f_{eq} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} [c_i == c_j] \tag{5}$$

$$f_{ne} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} [c_i \neq c_j] \tag{6}$$

With IS denoting an inverted sum, we define the intra-class distance $IS_{eq}(W)$ and the inter-class distance $IS_{ne}(W)$ as

$$IS_{eq}(W) = f_{ne} \times \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{[c_i == c_j]}{\mu + d_{i,j}(W)} \tag{7}$$

$$IS_{ne}(W) = f_{eq} \times \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{[c_i \neq c_j]}{\mu + d_{i,j}(W)} \tag{8}$$

with constant parameter $\mu = 1$. Any $\mu > 0$ is suitable to avoid divisions by zero, but choosing too large a μ results in the distances not mattering. Naturally, we want the distances to be small when the classes match and large when they do not. The value of μ is empirically chosen as part of our design. Figure 2 illustrates that this choice does not greatly impact the score function. The total class distance is

$$IS_{total}(W) = IS_{eq}(W) + IS_{ne}(W) \tag{9}$$

Now we can define our criterion $J(W)$, which is

$$J(W) = K(W) \frac{IS_{eq}(W)}{IS_{total}(W)} \tag{10}$$

where $K(W)$ serves as a correlation adjustment.

The effect of using the sum of inverse distances is similar to the least squares method. However, in the least squares method, we discourage the non-equal terms whereas in this 'most inverses' method we encourage the equal terms.

Our method fundamentally promotes locality, *i.e.*, it does not force unrelated clusters together but instead allows for disjoint clusters, which has a number of implications. It makes no assumptions about the general nature of the sample set (it does not dictate one rule globally), it allows for the existence of multiple centers within one system, it is minimally affected by outliers, and it measures simultaneous and possibly disjoint clusters while performing feature weighing.

B. Discussion

The first significant concept to note is that we deliberately inflate the distances when more features are present. This is achieved by the interaction between the distance metric (2) and the constraint on the feature weights (1). The use of squares rather than some other exponent in the constraint is one of the internal parameters mentioned earlier. We use an appropriate power to make the constraint act much like the correlation. Without the effect of μ on the correlation measure, if we used an L_1-projection constraint ($\sum |w| = 1$) instead of constraint (1), the number of features present (the subset size) would not affect the final score function because we would have a homogeneous space (*e.g.*, the weight vectors $\{1, 2, 5\}$ and $\{2, 4, 10\}$ would be the same after normalization). However, given our L_2 projection, the weight vectors are larger when considering a subset with more features. So, depending on the similarity of the feature weights ($w_1 \approx w_2 \approx \ldots \approx w_Q$), the

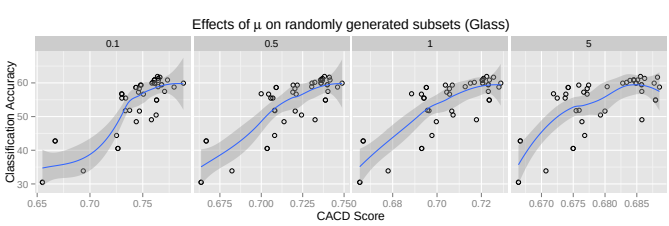

Fig. 2. Illustration of the effect of different values for μ

size of the weight vector will vary akin to how the respective points on the surface of a unit hypersphere are greater than or equal to the points on the surface of a unit hypercube. As mentioned above, this 'buffer' is needed to compensate for the nonlinear overestimation that is introduced by only considering pair-wise correlations. Considering all correlations would be computationally intractable for high-dimensional datasets and would typically only yield a small improvement in accuracy.

Since μ determines the degree of this increase for larger subsets, varying μ allows to adjust the relative influence of the correlation measure (4) on our criterion and, thus, we have a mechanism to 'encourage' pair-wise uncorrelated subsets with little extra computation. In other words, μ represents a knob to bias the criterion more towards correlation or more towards class distance. For example, in case of redundant dimensions with well-organized classes, a larger μ would devalue subsets that include redundant dimensions despite the subset's excellent ability to discriminate the classes. Note that all of this can be achieved with just one simple parameter.

Another important feature of our approach is the choice of an inverted sum to measure the quality of the class separation. It provides several key benefits. First, it (along with using Kendall's Tau-b) makes the criterion robust in the presence of outliers. Since we are dividing by the distance, outliers, which have a large distance, contribute essentially nothing to the inverted sum. In contrast, a least squares approach is easily thrown off by a single large outlier. Second, our method naturally encourages locality, *i.e.*, many small distances. Hence, it encourages class separation on a local context as the large-distance terms are insignificant. Third, it does all this without making assumptions about independence or a general inherent structure among the features. We consider these aspects fundamental advantages of our design.

The inverted sum also provides an important performance-optimization opportunity. Since only the short-distance terms matter, using a nearest-neighbor list in the search method incurs a minimal approximation error. When restricting the algorithm to only considering a fixed number of nearest neighbors, the time complexity decreases by a factor of $\mathcal{O}(n)$.

Many of the components of our criterion are embedded within our design (*e.g.*, μ) or pre-computed and therefore do not add significantly to the computation cost. However, they do add to the complexity of the score function. Since we are performing feature selection in a continuous manner, this makes it more difficult to find optimal subsets and may cause some search strategies to require extra iterations to achieve equivalent quality. This is one of the reasons why, in the following section, we introduce a search strategy that is not dependent on analytic methods or linear programming models.

IV. Tree Search

This section introduces our tree-based search and discusses its matrix and tree representation as well as some of its computational qualities. Since each feature weight can have any value between 0.0 and 1.0, it is important to establish a systematic and efficient way of subdividing and traversing the search space. We use a generalization of the Stern-Brocot tree.

A. Overview

A number of pre-existing search strategies that produce reliable results only work for criterion functions that meet certain strict requirements, such as having to be monotonic. Other search strategies tend to get stuck in local optima. Our approach does not suffer from either of these limitations.

One of the fundamental qualities of our tree search is the effect that taking a direction, *i.e.*, choosing a child to follow, has on the range of possible values for the weights. Consider the Stern-Brocot tree in Figure 1. Given an initial choice of 'left', all subsequent paths are limited to values below 1. Hence, the tree represents an effective way of partitioning the search space into independent regions with little computational overhead or dependency among workers. Another important quality is that the tree makes it possible to quantify the distance between different subsets in a meaningful manner without the need to compute the actual weights (*cf.* Section V).

Additional benefits of using a tree-based search include the following. In the presence of a dominant path, *i.e.*, a monotonic score function, only one child needs to be followed after each step and the search converges exponentially. Even when optimizing a function that is not analytic, our approach is guaranteed to terminate. First, unlike in gradient descent, the tree search cannot jump back and forth between the same two values because the step size diminishes with each level in the tree. Second, every downward step moves us closer to the (local) optimum as the range of possible weights is further confined. Hence, any desired degree of accuracy can be attained. Finally, the generated weights are guaranteed to be unique and therefore fit well with and can also be used in the parameterized coefficient search paradigm that is common in many combinatorial optimization problems that occur in fields like statistics, data mining, and linear programming.

B. Matrix representation

We consider the matrix representation very useful for reasoning about the tree-based search method. After all, deconstructing the matrices enabled us to greatly reduce the computational complexity of our approach (*cf.* Section V).

Given the real numbers $r \in (0, 1)$ (a factor that determines how quickly the tree 'spreads out') and $d \in \mathbb{N}$ (the number of features), let J_d denote a d-element vector of all ones, $S_d(i)$ a $d \times d$ zero matrix with r in every row of the i^{th} column,

$$S_d(i) = rJ_d e_i = \begin{bmatrix} 0 & \cdots & r & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & & r & & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & r & \cdots & 0 \end{bmatrix}_{d \times d}$$

and $I_d^-(i)$ the $d \times d$ identity matrix with the i^{th} 1 removed

$$I_d^-(i) = I_d - e_i^\top e_i = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & & & \vdots \\ & & 1 & & \\ \vdots & \ddots & & \ddots & \vdots \\ & & & 1 & \\ \vdots & & \ddots & & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix}_{d \times d}$$

The transformation matrix $T_d(i)$ $1 \leq i \leq d$ is their sum.

$$T_d(i) = S_d(i) + I_d^-(i)$$

Finally, we define a sequence $P = \{p_1, p_2, \ldots, p_k \mid 1 \leq p_i \leq d \, \forall i\}$ as a *path*, with each p_i denoting a step or direction in the path (*i.e.*, choosing the i^{th} child of the current node in the tree), and the vectorization operation V of a path P as

$$V(P) = r\Big(T_d(p_1)T_d(p_2)\ldots T_d(p_k)J_d\Big)^\top = \vec{v} \in \mathbb{R}^d$$

To simplify the notation, we assume that d and r are given constants unless otherwise stated. For example, $r = 1$ and $d = 3$ yield the following on the path $\{2, 3\}$

$$\begin{aligned} V(\{2,3\}) &= \Big(T(2)T(3)J_d\Big)^\top \\ &= \left(\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\right)^\top \\ &= \left(\begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\right)^\top \\ &= (4, 2, 3) \end{aligned}$$

This is analogous to summing across each row, multiplying by r, and placing the result in the i^{th} column. Furthermore, right-multiplication by $I^-(i)$ results in the same matrix with the i^{th} row removed as illustrated in the following

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{d1} & \cdots & \cdots & a_{dd} \end{bmatrix}$$

$$\begin{aligned} A \times T(i) &= A \times S(i) + A \times I^-(i) \\ &= \begin{bmatrix} 0 & \cdots & r\sum a_{1i} & \cdots & 0 \\ \vdots & \ddots & r\sum a_{2i} & \ddots & \vdots \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & r\sum a_{di} & \cdots & 0 \end{bmatrix} + \begin{bmatrix} a_{11} & \cdots & 0 & \cdots & a_{1d} \\ a_{21} & \ddots & 0 & \ddots & \vdots \\ \vdots & \ddots & 0 & \ddots & \vdots \\ a_{d1} & \cdots & 0 & \cdots & a_{dd} \end{bmatrix} \\ &= \begin{bmatrix} a_{11} & \cdots & r\sum a_{1i} & \cdots & a_{1d} \\ a_{21} & \ddots & r\sum a_{2i} & \ddots & \vdots \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{d1} & \cdots & r\sum a_{di} & \cdots & a_{dd} \end{bmatrix} \end{aligned}$$

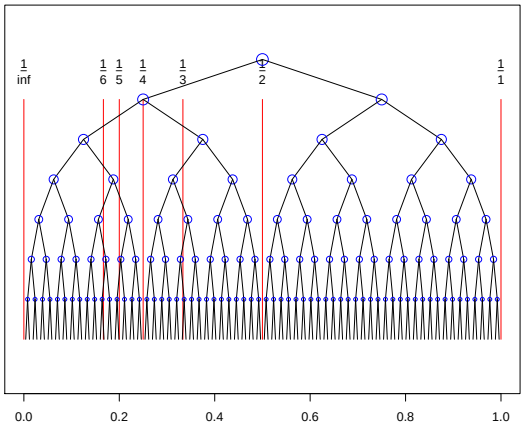

Fig. 3. $r = \frac{1}{d}$ divergence ($r = \frac{1}{2}$, $d = 2$)

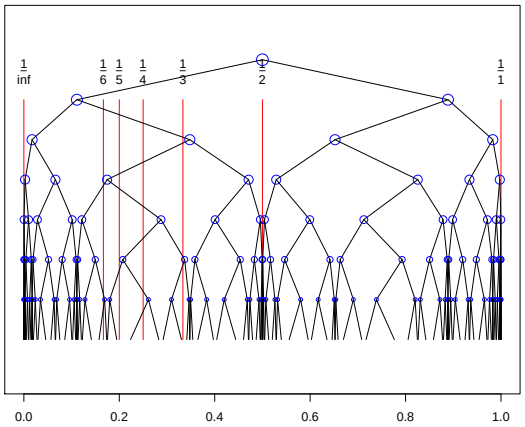

Fig. 4. $r < \frac{1}{d}$ divergence ($r = \frac{1}{7}$, $d = 2$)

This construction ensures that, for any node, there exists no other node in the tree that is a scalar multiple of it (this is equivalent to the reduced-form quality of the rational numbers in the original Stern-Brocot tree). This is important since multiples of the same set of weights would be mapped to the same final weights and thus result in redundant evaluations.

C. Tree representation

In the following illustrations, we use the projection (normalization) of $\vec{v}$ onto $\sum_i v_i = 1$ to show the effect of the divergence ratio r on the tree. Informally, the divergence determines how rapidly the tree 'spreads out'. Note that our Stern-Brocot variant is not identical to the original as we evaluate $\frac{v_1}{v_1+v_2}$ while the original Stern-Brocot approach evaluates $\frac{v_1}{v_2}$. We made this change because our projection generalizes to higher dimensions. This difference is inconsequential for demonstrating the effect of the divergence. Whereas our variant deviates from the original tree (which is no longer a subset of our generalization), we can still obtain the rest of the original tree, which spans $(1, \infty)$, by inverting all nodes $\in (0, 1)$.

Figure 3 presents a perfectly balanced version of our tree. With $r = \frac{1}{d}$, the projection space evenly partitions the region $(0, 1)$ for any d, that is, any number of features. From the point of view of a pre-projection vector, this divergence ratio

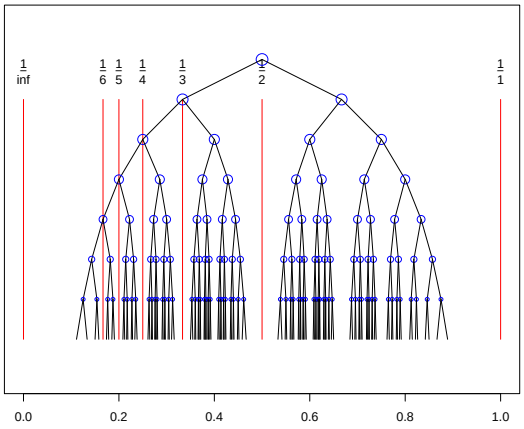

Fig. 5. Stern-Brocot divergence ($r = 1$, $d = 2$)

means that we adjust the resulting vector length after adding two vectors in each step down the tree to form a new vector of the same length. It is important to note that not all weights are reachable using a finite number of steps given $r \neq 1$, but this is not an issue in terms of our search as we can still get arbitrarily close and then terminate the search.

Figure 4 illustrates a divergence above the balance point. We found such $r < \frac{1}{d}$ ratios to be the most useful because, for dimensionality reduction, we prefer faster divergence to the boundaries. If the good solutions lie near the boundaries, the original Stern-Brocot divergence ($r = 1$) performs poorly since the rate of expansion towards the borders decreases as we go down the tree, as Figure 5 illustrates.

Figure 6 presents these divergence ratios in $d = 3$ projected to $\vec{u} = \left(\frac{v_1}{v_1+v_2+v_3}, \frac{v_2}{v_1+v_2+v_3}\right) \in \mathbb{R}^2$. It shows the limitations of using an $r = 1$ divergence in higher-dimensional domains, which only covers a small part of the search space (it takes many steps to get close to the borders) whereas $r = \frac{1}{d}$ covers the space evenly and $r < \frac{1}{d}$ emphasizes the border regions.

V. Computational Qualities

Figure 7 illustrates, for a 5D dataset, how every path from the root (at the center) of our search tree leads to a unique distribution of the five weights (each slice of a pie represents a weight). In particular, every step along a path subdivides the search space into ever smaller partitions, that is, it restricts the range of possible values for each weight. We repeat this process until there is no more improvement in $J()$ or we approach a local optimum. Given a specific path, we can define our score function as

$$J(W) = J\left(\frac{V(P)}{||V(P)||_2}\right)$$

where V is the vectorization function and P is the path.

Our search method starts from identity (where all weights are equal) and uses a divergence parameter r that is either equal to or less than the inverse of the dimensionality. In the latter case, the tree emphasizes the boundary regions where we assume the optimal results to be located, *i.e.*, where the lower-dimensional subsets reside since some of

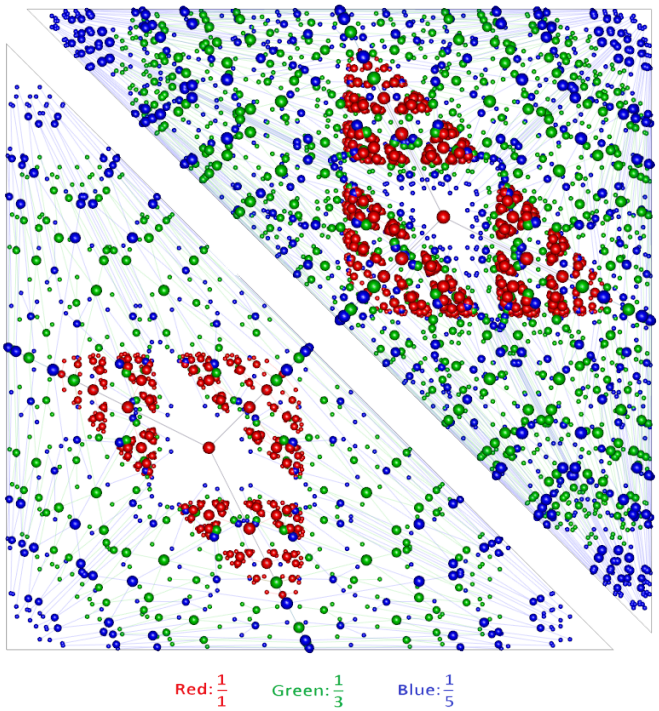

Fig. 6. Comparison of the 5^{th} layer (bottom triangle) and 6^{th} layer (top triangle) ($r_{red} = \frac{1}{1}$, $r_{green} = \frac{1}{3}$, $r_{blue} = \frac{1}{5}$, $d = 3$)

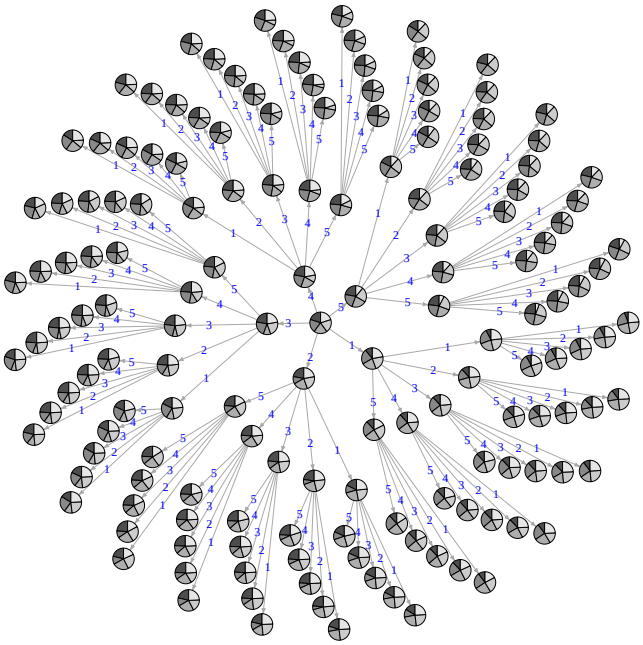

Fig. 7. First four levels of our five-dimensional search tree where the size of the pie slices represents the magnitude of each of the five weights

the weights are zero on the boundaries. This is similar to a gradient descent except the step size is given by the tree depth. Moreover, instead of moving locally (additively), we are moving globally (multiplicatively) within the region of the search space determined by the parent node. Every step down the tree further constrains this region. Note that this is all done

as part of the tree traversal without additional computations.

Algorithm 1: Vectorize(P)

Input: r is a scalar constant, k is the number of steps in the path, and d is the number of features

Input: $P = \{p_1, p_2, \ldots, p_k \mid 1 \leq p_k \leq d\,, k \in \mathbb{N}\}$

Output: $\vec{v} = \{v_1, v_2, \ldots, v_d\}$

begin

/* populate matrix */

for $i = 1 \rightarrow d$ **do**

col$[i] \longleftarrow e_i$ $\quad \{\{1, 0, \ldots, 0\}, \{0, 1, \ldots, 0\}, \ldots\}$

end

sums $\longleftarrow \{1, 1, ..., 1\}$ $\quad$ {given identity I_d}

/* apply transformations */

for $i = 1 \rightarrow k$ **do**

old $\longleftarrow$ col$[p_i]$

col$[p_i] \longleftarrow r \times$ sums

sums $\longleftarrow$ col$[p_i]$ + sums − old

end

return *sums*

end

Note that each assignment requires d operations.

In high-dimensional datasets, concurrently following multiple tree branches using the matrix representation is expensive as multiple large matrices have to be stored. Fortunately, this problem can be remedied by identifying a tree node using the path leading to it, which allows us to exploit the vectorization process of a path. Without this optimization, performing a $V(P)$ calculation for some path $P = \{p_1, p_2, \ldots, p_k\}$ requires k matrix multiplications, which take $\mathcal{O}(kd^3)$ operations. Reusing the old row sums in Algorithm 1 reduces the complexity to $\mathcal{O}(d^2 + kd)$, where the $\mathcal{O}(d^2)$ component is for generating the initial sums and the $\mathcal{O}(kd)$ component is for applying the transformations. We can further lower this complexity to $\mathcal{O}(kd)$ by recording the sum for each step, *i.e.*, by storing $d^2 + d$ values instead of only d^2 values. This small increase in storage greatly reduces the amount of computation because it eliminates the need to reinitialize the matrix; instead, we can simply continue the traversal from a tree node using the recorded sum. This benefit makes the path approach a computationally interesting alternative to the matrix approach. Moreover, both types of search are amenable to parallelization on multicore processors if desired.

Another useful aspect of the path notation is that it allows the direct measurement of the distances between subsets without having to compute any weights. As mentioned, this benefit is often overlooked. For example, given $r = 1$, $d = 3$ and the paths $a = (3, 1, 1, 1), b = (2, 1, 1, 1)$ and $c = (1, 3, 3, 3)$, path a may at first glance appear to be closer to b than to c, but this is not the case. Path c is quite close to a much like paths $(LRRR\ldots)$ and $(RLLL\ldots)$ converge towards each other in the original 2D Stern-Brocot tree in Figure 1. To formally show that path a is closer to c than to b, let us deconstruct the three vectors into binary form with respect to each possible direction. As shown below, we decompose the k elements of each path into d binary vectors with k elements where each element indicates the presence or absence of a particular direction in that position. For example, following child 3 in the first step of path a results in 0s in the first position of all d vectors except in the vector representing 3, which has a 1 in the first position. The remaining bits are determined similarly.

$$a = (3, 1, 1, 1) = \{[0, 1, 1, 1]_1, [0, 0, 0, 0]_2, [1, 0, 0, 0]_3\}$$
$$b = (2, 1, 1, 1) = \{[0, 1, 1, 1]_1, [1, 0, 0, 0]_2, [0, 0, 0, 0]_3\}$$
$$c = (1, 3, 3, 3) = \{[1, 0, 0, 0]_1, [0, 0, 0, 0]_2, [0, 1, 1, 1]_3\}$$

Given this format, we can easily compare the paths by subtracting one from the other. Since 1 and -1 are equally far away from 0, it suffices to only consider the absolute value of each term. For improved readability, we convert the binary differences into decimal vectors as a final step.

$$a - b = \{[0, 0, 0, 0]_1, [1, 0, 0, 0]_2, [1, 0, 0, 0]_3\} = \{0, 8, 8\}$$
$$a - c = \{[0, 0, 0, 1]_1, [0, 0, 0, 0]_2, [0, 0, 0, 1]_3\} = \{1, 0, 1\}$$

Since $d = 3$, the distance between two paths is a vector of three elements. Clearly, the vector $\{0, 8, 8\}$ is longer than the vector $\{1, 0, 1\}$, confirming that path a is closer to c than b.

We are explaining this computation in such detail to highlight that one can quickly compute the distance between two paths using simple binary operations, *i.e.*, without the matrices, vectorization, or explicit computation of the weights. This is useful for exploring multiple tree branches based on diversity because the distance of each candidate path can be rapidly evaluated so that sufficiently different paths can be chosen.

Finally, let us take a look at the coefficients (vectorization) generated by these paths: $V(a) = (5, 8, 4)$, $V(b) = (5, 4, 8)$ and $V(c) = (4, 8, 5)$. Again, $V(a)$ is more similar to $V(c)$ than to $V(b)$. More importantly, this technique can also be reversed to generate a new path from existing paths to promote diversity in the search space, which is another benefit of our approach.

Although the presented type of search is ideal for continuous $J()$ functions, it is also capable of producing a binary backwards or forwards search. For $r = 0$, the algorithm performs a binary backward selection with the restriction that it can only take a specific step of a path once to ensure termination. If we define a new vector $\vec{u} = 1 - \frac{\vec{v}}{\sum v_i}$ with $r = 0$ and a similar path restriction on repeating a step, we perform a binary forward selection. We are not suggesting that anyone use our method in this way. Rather, we want to point out that the popular binary backwards and forwards searches emerge as two special cases of our more general approach.

A. Traversal procedure and termination

Our search method starts at the root, with identity weights, and calculates a coefficient of variation based on all children's $J()$ score, which determines the number of branches to consider. The higher the variation is, the more children are visited. Generally, only few children need to be followed, thus avoiding and drastically outperforming an exhaustive search.

This process continues recursively, with some decay based on the depth as a soft termination criterion to limit the total computation. The coefficient of variation also serves as a hard termination criterion because, with each increase in depth, the $J()$s of the children get closer to each other. Eventually, the coefficient becomes arbitrarily small (indicating that the score is not improving) because each step considers a finer granularity than the previous step. An added benefit is that, given a change in our weights (let us call this distance δ), we can make certain assumptions about the corresponding change in our $J()$ function. This allows us to finish the search outside the tree using an iterative uphill climb with δ as its step size because we can guarantee (with arbitrarily high probability depending on the threshold) that we will not miss a local optimum, *i.e.*, we will be in the safe zone. This hybrid approach between the tree search and the following uphill climb greatly decreases the running time as the convergence of the tree search slows down with increased depth.

VI. Evaluation

A. Datasets

We use nine popular datasets from the UCI machine learning repository for our evaluation [11], [12], [13]. Aside from ensuring proper formatting, we did not modify these datasets. Table I summarizes pertinent information about each dataset.

TABLE I
Datasets

Dataset	Number of classes	Number of features	Number of instances
Madelon	2	500	2000
Breast cancer	2	10	699
Indian liver patient	2	10	583
Musk	2	166	476
Ionosphere	2	34	351
Glass identification	7	9	214
Sonar	2	60	208
Parkinsons disease	2	23	197
Lung cancer	3	56	32

B. Methodology

We compare our TS-CACD approach to the L0, mRMR, InfoGain, ReliefF, and CfsSubset feature-selection methods. The L0 subsets were generated using Matlab with the mc_svm (multi-class support vector machine) function. For mRMR, we used the mRMR website to perform the feature selection [14]. We employed Weka for the remaining three methods [15].

To evaluate each method, including ours, we averaged the classification accuracy of the NaiveBayes (a Bayesian model with a probabilistic metric), J48 (a decision tree with a region-based metric), IBk (a k-nearest neighbor algorithm with a distance metric), and SMO (a support vector machine with a kernel metric) classifiers with ten-fold cross-validation on the resulting subset recommendation using the R package RWeka [16]. We performed our testing with these four archetypal classifiers to encourage diversity, *i.e.*, so as not to bias the results toward a single classifier and its associated metric. Moreover, we made sure that none of the four algorithms consistently performed poorly for a given classifier, which would otherwise have resulted in an unfair comparison due to a poor match of classifier to feature-selection method.

The TS-CACD results were obtained on an implementation that works as described in this paper. In particular, it uses a fixed $\mu = 1$ so as not to rely on a parameter (note that, for some subsets, a slightly smaller μ would give better results). For the methods that produce a ranked list of features, we attempted to match either the score or the dimensionality to remain unbiased unless there was a strong preference for a specific subset, in which case we show results for that subset.

The evaluated TS-CACD implementation does not take advantage of some of the discussed performance optimizations (such as the nearest neighbor search) as it is primarily intended to demonstrate the quality of our proposed feature-selection method. Nevertheless, preliminary tests of a nearest-neighbor implementation on randomly generated subsets from the Sonar dataset resulted in less than a 1% change in $J()$ score, indicating that only considering the k nearest neighbors does not hurt the classification accuracy significantly while, at the same time, making the implementation much faster.

C. Results

Table II compares the classification accuracy and the number of selected features of the six methods on the nine datasets. TS-CACD strictly outperforms the other algorithms on Parkinsons and ILPD. On these two datasets, it ties with one other method each for the smallest subset but outperforms all methods in classification accuracy. On Sonar, Ionosphere, and Glass, TS-CACD also achieves the highest accuracy of all tested methods but not the smallest subset size. Nevertheless, in two cases it produces the second smallest subset. On Madelon and Musk, our method yields the smallest subset. On Madelon, no method is strictly better than any other evaluated method, *i.e.*, for each pair of methods, one has a higher accuracy whereas the other results in a lower subset size. On the remaining two datasets, Lung Cancer and BCW, our method is a close second in terms of accuracy but yields much smaller subsets than the most accurate methods. Importantly, there is no case where one of the other methods provides better accuracy and a smaller number of features. In summary, TS-CACD results in the best or close to best accuracy on seven of the nine datasets and yields the smallest subsets on the remaining two datasets. This outcome highlights the merit of our approach, that is, the importance of considering multiple dimensions simultaneously when selecting features.

Since TS-CACD's performance is a combination of the CACD metric and the tree search, we also evaluated the CACD metric in isolation, *i.e.*, independent of any search. In particular, we score random subsets using the CACD metric. The subsets were generated by first choosing a random size between one and the number of features and then randomly selecting features until the chosen subset size has been reached. Figure 8 presents the results where the lines in each panel are linear trend lines and the shaded regions represent the confidence interval for the associated plot.

TABLE II
CLASSIFICATION ACCURACY AND NUMBER OF SELECTED FEATURES BY METHOD

Dataset	w/o feat. sel.		L0		mRMR		InfoGain		ReliefF		CfsSubset		TS-CACD	
Madelon	59.462	500	63.650	3	70.275	11	70.637	16	**72.682**	20	68.175	7	62.050	**2**
Musk	81.985	166	78.571	23	72.899	20	77.836	**14**	73.897	21	**82.983**	36	77.416	18
Sonar	75.360	60	78.606	**6**	76.201	30	78.125	21	77.884	21	76.923	19	**79.206**	24
Lung Cancer	50.781	56	69.531	7	**75.120**	20	71.093	8	72.949	**4**	68.750	7	72.656	7
Ionosphere	87.250	34	86.609	13	89.316	10	88.176	18	89.102	**5**	89.814	14	**90.384**	9
Parkinsons	83.333	23	84.102	7	83.903	5	85.897	**3**	86.410	5	85.128	10	**86.538**	**3**
ILPD	65.112	10	71.012	**2**	64.193	5	65.780	3	64.837	3	66.166	5	**71.549**	**2**
BCW	95.815	9	93.919	3	94.921	5	**96.030**	8	94.563	**2**	95.815	9	95.565	3
Glass	60.864	9	60.981	5	60.981	5	62.500	7	60.514	**3**	60.864	7	**62.850**	5

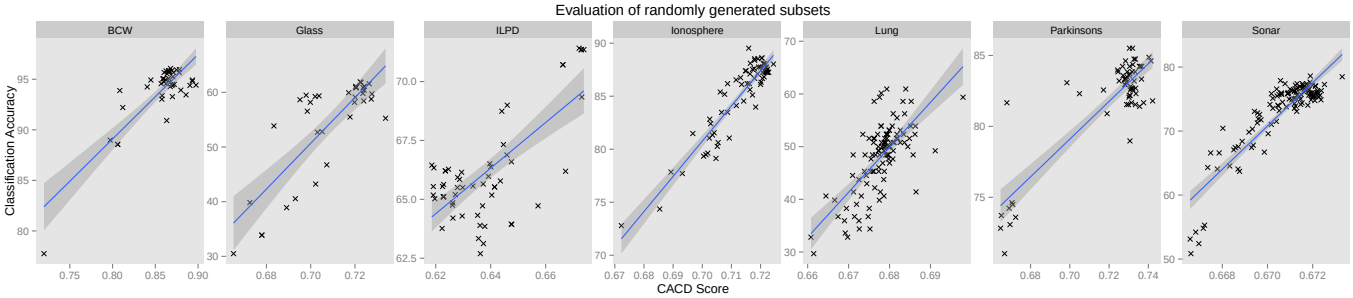

Fig. 8. Analysis of the CACD metric without any search

The results from Figure 8 are consistent with the results from Table II. Moreover, they show a strong correlation between the CACD score and the final classification accuracy, indicating that the CACD metric works well. Considering that some of these subsets are unlikely to appear in combination with a search, this demonstrates that the CACD metric accurately assesses subsets independent of our tree search. Hence, the CACD metric is likely to also be useful in combination with other (non-tree-based) search techniques.

VII. CONCLUDING REMARKS

This paper presents and evaluates our correlation-adjusted class distance criterion. In combination with our novel tree-search approach, it is very successful in finding competitive subsets in 9 datasets. The paper further presents an effective way of combining a distance and a dependency measure.

In future work, we plan to present some of the more mathematically oriented qualities of this tree and to generalize its notation. We would also like to find better alternatives to our coefficient of variation branching method so that it can consider multiple consecutive steps in a path in each iteration.

VIII. ACKNOWLEDGMENTS

Martin Burtscher and his research team are supported by NSF grants 1141022 and 1217231 as well as grants and gifts from NVIDIA Corporation and Texas State University.

REFERENCES

[1] T. Caesar, J. Gloger, and E. Mandler, "Preprocessing and feature extraction for a handwriting recognition system," pp. 408–411, 1993.

[2] E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne, "Finding high-quality content in social media," pp. 183–194, 2008.

[3] Y. Saeys, I. Inza, and P. Larraaga, "A review of feature selection techniques in bioinformatics," *Bioinformatics*, vol. 23, no. 19, pp. 2507–2517, 2007. [Online]. Available: http://bioinformatics.oxfordjournals.org/content/23/19/2507.abstract

[4] H. Liu and L. Yu, "Toward integrating feature selection algorithms for classification and clustering," *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, no. 4, pp. 491–502, Apr. 2005. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2005.66

[5] P. Pudil, J. Novovicova, and J. Kittler, "Floating search methods in feature selection," *Pattern Recognition Letters*, vol. 15, no. 11, pp. 1119 – 1125, 1994. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0167865594901279

[6] C. Yun and J. Yang, "Experimental comparison of feature subset selection methods," 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.107.6075

[7] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete mathematics: a foundation for computer science*, 2nd ed. Addison-Wesley, 1994.

[8] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *Proceedings of the ninth international workshop on Machine learning*, ser. ML92. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992, pp. 249–256.

[9] M. Scherf and W. Brauer, "Feature selection by means of a feature weighting approach," Forschungsberichte Kunstliche Intelligenz, Institut fur Informatik, Technische Universitat Munchen, Tech. Rep., 1997.

[10] M. Kendall and J. D. Gibbons, *Rank Correlation Methods*. Edward Arnold, 1990.

[11] M. F. Akay, "Support vector machines combined with feature selection for breast cancer diagnosis," *Expert Systems with Applications*, vol. 36, no. 2, Part 2, pp. 3240 – 3247, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417408000912

[12] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml

[13] M. A. Little, P. E. McSharry, E. J. Hunter, and L. O. Ramig, "Suitability of dysphonia measurements for telemonitoring of parkinson's disease," *IEEE Transactions on Biomedical Engineering*, 2008.

[14] H. Peng, F. Long, , and C. Ding, "Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy," vol. 27, no. 8.

[15] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H., "The weka data mining software: An update," *SIGKDD Explorations*, vol. 11, no. 1, 2009.

[16] K. Hornik, C. Buchta, and A. Zeileis, "Open-source machine learning: R meets Weka," *Computational Statistics*, vol. 24:2, pp. 225–232, 2009.