Computational Statistics and Data Analysis 96 (2016) 12–23

Contents lists available at ScienceDirect

Computational Statistics and Data Analysis

journal homepage: www.elsevier.com/locate/csda

HHCART: An oblique decision tree

D.C. Wickramarachchi [a,*], B.L. Robertson [b], M. Reale [b], C.J. Price [b], J. Brown [b]

[a] Department of Statistics, Faculty of Applied Sciences, University of Sri Jayewardenepura, Gangodawila, Nugegoda, Sri Lanka
[b] School of Mathematics and Statistics, University of Canterbury, Private Bag 4800, Christchurch, 8042, New Zealand

ARTICLE INFO

Article history:
Received 28 August 2014
Received in revised form 9 November 2015
Accepted 9 November 2015
Available online 29 November 2015

Keywords:
Oblique decision tree
Data classification
Statistical learning
Householder reflection
Machine learning

ABSTRACT

Decision trees are a popular technique in statistical data classification. They recursively partition the feature space into disjoint sub-regions until each sub-region becomes homogeneous with respect to a particular class. The basic Classification and Regression Tree (CART) algorithm partitions the feature space using axis parallel splits. When the true decision boundaries are not aligned with the feature axes, this approach can produce a complicated boundary structure. Oblique decision trees use oblique decision boundaries to potentially simplify the boundary structure. The major limitation of this approach is that the tree induction algorithm is computationally expensive. Hence, as an alternative, a new decision tree algorithm called HHCART is presented. The method uses a series of Householder matrices to reflect the training data at each non-terminal node during tree construction. Each reflection is based on the directions of the eigenvectors from each class' covariance matrix. Considering of axis parallel splits in the reflected training data provides an efficient way of finding oblique splits in the unreflected training data. Experimental results show that the accuracy and size of HHCART trees are comparable with some benchmark methods. The appealing feature of HHCART is that it can handle both qualitative and quantitative features in the same oblique split.

1. Introduction

Decision trees (DTs) are an increasingly popular method used for classifying data. In the typical tree building procedure, the space that the data occupies (feature space) is iteratively partitioned into disjoint sub-regions until each sub-region is homogeneous (or nearly so) with respect to a particular class. In a DT, each sub-region is represented by a node in the tree. The node can be either terminal or non-terminal. Non-terminal nodes are impure and can be split further using a series of tests based on the feature variables, a process called splitting. Each split is determined by considering a series of hyperplanes that separate the feature space into two sub-regions. The best hyperplane split is chosen as the one which maximises the change in an impurity function ($\Delta(I)$). To obtain a fully grown tree, this process is recursively applied to each non-terminal node until terminal nodes are reached. The terminal nodes correspond to homogeneous or near homogeneous sub-regions in the feature space. Each terminal node is assigned the class label that minimises the misclassification cost at the node.

DTs play an important role in statistical learning and have been a popular technique for data classification over several decades (see Breiman et al., 1984; Murthy et al., 1994; López-Chau et al., 2013). In the tree building process, the aim is to

* Corresponding author. Tel.: +94 714899637; fax: +94 112802914.
E-mail addresses: chitraka@sjp.ac.lk (D.C. Wickramarachchi), Blair.Robertson@canterbury.ac.nz (B.L. Robertson), Marco.Reale@canterbury.ac.nz (M. Reale), Chris.Price@canterbury.ac.nz (C.J. Price), Jennifer.Brown@canterbury.ac.nz (J. Brown).

http://dx.doi.org/10.1016/j.csda.2015.11.006
0167-9473/

produce accurate and smaller trees while minimising the computational time. Accuracy, size and time mainly depend on the way non-terminal nodes are split in a DT. Three types of splits are considered including axis parallel, oblique and non-linear splits. Axis parallel splits partition the space parallel to the feature axes. Therefore, axis parallel trees are desirable when the decision boundaries are aligned with the feature axes. Oblique splits are hyperplane splits defined by a linear combination of the feature variables. These splits are more appealing when the decision boundaries are not aligned with the feature axes. Non-linear splits (Ittner and Schlosser, 1996; Li et al., 2005) are general class of splits. Decision boundaries generated by these splits can take arbitrary shapes and can easily be influenced by noise data (Li et al., 2005).

Many algorithms have been proposed to induce DTs. In general, these algorithms differ in the way they search for the best split at each non-terminal node. Many studies show that trees which use oblique splits generally produce smaller trees with better accuracy compared with axis parallel trees (Li et al., 2003). Therefore, they have become increasingly popular in DT literature and motivated us to propose a new methodology to construct a DT that uses oblique splits at each non-terminal node. These DTs are called oblique decision trees (Murthy et al., 1994). More specifically, let the feature vector consists of p attributes, $\mathbf{x} = [x_1, x_2, \ldots, x_p]^T$ where $x_i \in \mathbb{R}$. The oblique splits can be defined as linear combinations of features of the form:

$$\sum_{k=1}^{p} a_k x_k + a_{p+1} \leq 0, \quad \text{where } a_1, a_2, \ldots, a_{p+1} \in \mathbb{R}. \tag{1}$$

One of the major issues when inducing an oblique DT is the time complexity of the induction algorithm. In a data structure with p feature variables and n examples at a non-terminal node, the number of splits to be evaluated to find the best axis parallel split is $O(np)$. Therefore, the globally optimal split (with respect to an impurity function) at a non-terminal node can be found by exhaustively searching all possible splits along the feature axes. However, the number of splits to be evaluated to find the best oblique split at a node by exhaustive searching is, at most, $O\left(2^p \times \binom{n}{p}\right)$ (Murthy et al., 1994). Hence an exhaustive search for the best oblique split is impractical. Furthermore, the best split at a node does not necessarily lead to the optimal tree. Spending more time searching for the best split at a node in general may not be beneficial (Iyengar, 1999). Furthermore, Hyafil and Rivest (1976) pointed out that the problem of finding an optimal binary DT is an NP-complete problem. This led us to search for efficient heuristics for constructing near optimal DTs. In this work, we propose a simple, and effective heuristic method to induce oblique decision trees.

The remaining sections of this paper are organised as follows: Section 2 highlights related work. Section 3 introduces the proposed method. Comparisons with some commonly used DT algorithms are presented in Section 4. Section 5 concludes the paper with a discussion.

2. Related work

Most of the oblique DT induction algorithms construct DTs in a top-down fashion (Rokach and Maimon, 2005). The induction algorithms differ in the way they search for the best split and can be categorised as follows. We define three categories: (1) induction algorithms that use optimisation techniques, (b) standard statistical techniques and (3) those that use heuristic arguments.

2.1. Tree induction methods based on optimisation techniques

The first major oblique DT algorithm was Classification and Regression Trees—Linear Combination, which is commonly known as CART-LC (Breiman et al., 1984). CART-LC uses a deterministic hill-climbing algorithm to search for the best oblique split at a non-terminal node. A backward feature elimination process is also carried out to delete irrelevant features from the split. CART-LC will not necessarily find the best split at each node because there is no built-in mechanism to avoid getting stuck in the local maxima of $\Delta(I)$. The best split found may be only a local, rather than global, maximiser of $\Delta(I)$.

Simulated Annealing Decision Tree (SADT) was introduced by Heath et al. (1993). This DT uses the simulated annealing optimisation algorithm, which uses randomisation, to search for the best split. The use of randomisation potentially avoids getting stuck in local maxima of $\Delta(I)$. The main disadvantage of the algorithm is the time taken to find the best split. In some cases, it may require the evaluation of tens of thousands of hyperplanes before finding an optimal split (Murthy et al., 1994).

The concepts of CART-LC and SADT are combined to produce a new oblique DT methodology called OC1 by Murthy et al. (1994). Their method uses a deterministic hill-climbing algorithm to perturb the coefficients of an initial hyperplane until a local maximum of $\Delta(I)$ is found. The hyperplane is then perturbed randomly in an attempt to find a hyperplane that improves $\Delta(I)$ further. These two steps are repeated several times. Each time, the algorithm starts with a different initial hyperplane, with one being the best axis parallel split and the others chosen randomly. After many hyperplanes have been evaluated, the one that maximises $\Delta(I)$ is taken as the splitting hyperplane. The time complexity at each non-terminal node for OC1 in the worst case scenario is shown to be $O\left(pn^2 \log n\right)$, provided that Max Minority or Sum Minority impurity measures are used. However, the complexity may increase for other impurity measures and for multi-class problems. One

feature of both SADT and OC1 is that both algorithms can construct different decision trees on different runs using the same learning sample. Therefore, it is possible to run these algorithms multiple times and pick the best tree. However, this advantage is only realised on relatively small training example sets.

2.2. Tree induction methods based on standard statistical techniques

Various oblique DT induction algorithms have been developed using standard statistical techniques, and can be found in Gama and Brazdil (1999), Kolakowska and Malina (2005), Li et al. (2003) and López-Chau et al. (2013). The advantage of this approach is that the time required to induce DTs is generally lower than those based on optimisation algorithms. Fisher's Decision Tree (FDT) (López-Chau et al., 2013) finds separating hyperplanes using Fisher's Linear Discriminant. At each non-terminal node, examples are projected onto the vector given by Fisher's Linear Discriminant analysis. Axis parallel splits are then searched along the vector to find the best split. Quick Unbiased Efficient Statistical Tree (QUEST) (Loh and Shih, 1997) uses Linear Discriminant Analysis (LDA) to find the best split at each node and hence there is no requirement for searching for the best split. QUEST's axis parallel tree begins by performing an analysis of variance test at each non-terminal node to select the best feature. LDA is then applied on the selected feature to find the best splitting point. QUEST's oblique DT simply applies LDA on all features to find the best splitting hyperplane. Furthermore, QUEST is able to find oblique splits that are a linear combination of qualitative and quantitative features. For multi-class problems, QUEST groups the classes into two super-classes using the k-means clustering algorithm, where $k = 2$, which increases the time complexity of the algorithm.

2.3. Tree induction methods based on heuristics

DTs based on heuristic arguments have gained popularity in the recent past (Amasyah and Ersoy, 2008; Manwani and Sastry, 2012). In this approach, an argument is constructed by assuming the structure of the class boundaries. If the assumption is true, DTs based on heuristic arguments can produce accurate and small trees.

The Cline algorithm (Amasyah and Ersoy, 2008) uses three heuristics to find three "best" splits at each non-terminal node. The overall best split is taken as the separating hyperplane. The Cline algorithm is a two-class classifier, so therefore, in multi-class problems, it constructs several classifiers each to distinguish each class from the other classes. More specifically, if there are C classes then the Cline algorithm constructs $\binom{C}{2}$ classifiers.

The CARTopt algorithm introduced by Robertson et al. (2013) uses a two-class oblique tree to find a minimiser of a non-smooth function $f(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^n$. Initially, the examples in $\mathbb{R}^n$ are labelled as high and low, depending on their value of $f(\mathbf{x})$. An oblique DT is then used to form a partition on $\mathbb{R}^n$ that separates the low points from high points. Rather than forming the oblique DT directly, the authors reflected the training examples using a Householder matrix. Axis parallel splits are then searched in the reflected training data. These splits are oblique in the original space.

In summary, DT algorithms use various methods to find the best split at a non-terminal node. Some methods try to expand the search space, as higher search space potentially leads to finding better splits. For example, OC1 uses a randomisation step to potentially avoid the local maximum of $\Delta(I)$ found by the hill-climbing algorithm. This can be perceived as an attempt to expand the search space. Furthermore, the FDT uses axis parallel splits along Fisher's discriminate vector. The Cline algorithm uses three different methods to find the best split at a non-terminal node. The search space of each method is different from that of the others and thus indirectly creates a higher space for searching. Hence, the literature shows that expansion of the search space would be one of the strategies for finding better splits in the DT context. However, the key point is to use an efficient searching method in the expanded search space.

CARTopt introduces a new heuristic to induce oblique decision trees. It uses the simplest form of splits, axis parallel splits, to find oblique splits. Hence the time complexity of searching oblique splits using CARTopt's approach is less than those based on optimisation algorithms. In this study, we extend CARTopt's idea in a number of ways to develop a complete oblique DT for statistical data classification. In particular, the Householder reflection is used to: (1) find the oblique splits in the original feature space using the axis parallel splits in the reflected feature space and (2) expand the search space where the best split can be found at a minimal cost.

3. Methodology

We extend the oblique DT method used in the CARTopt optimisation algorithm of Robertson et al. (2013) in a number of ways to develop a complete oblique DT called HHCART. First, CARTopt is designed to classify two classes, whereas HHCART can handle multi-class classification problems. Second, CARTopt reflects the training examples at the root node only, whereas HHCART performs reflections at each non-terminal node during tree construction. Finally, CARTopt is only defined for quantitative features, whereas HHCART is capable of finding oblique splits, which can be linear combinations of both quantitative and qualitative features.

In our approach, we find each separating hyperplane by considering the orientation of each class. We propose the dominant eigenvector of the covariance matrix of a class to represent the orientation of that class. If this orientation is parallel to one of the feature axes, the best separating hyperplane may be found by performing axis parallel splits. Otherwise,

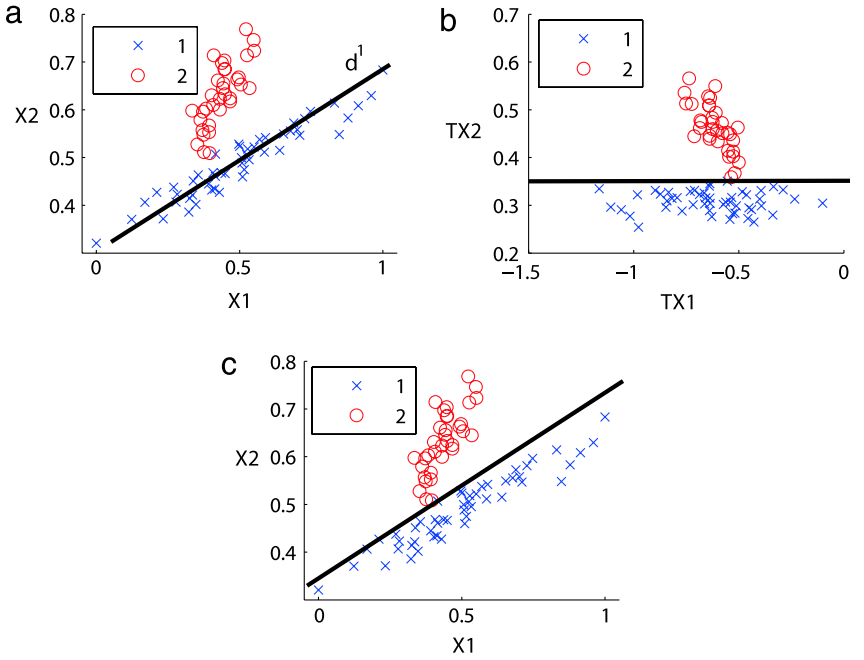

Fig. 1. Mechanism of the Householder reflection. The separating hyperplane is parallel to the dominant eigenvector of Class 1. (a) Scatter in the original space. d^1 is the dominant eigenvector of the class covariance matrix of Class 1. (b) Scatter in the reflected space and the best axis parallel split found. (c) Oblique split in the original space.

we reflect the set of examples to a new coordinate system such that the orientation of one of the classes becomes parallel to one of the axes in the reflected feature space. Axis parallel splits can then be searched in the reflected feature space to find the best split. This split will be oblique in the original feature space (Robertson et al., 2013).

Consider the two-dimensional two-class classification problems shown in Figs. 1(a) and 2(a). The separating hyperplane in Fig. 1(a) is parallel to the dominant eigenvector of Class 1, whereas the separating hyperplane for in Fig. 2(a) is perpendicular to the dominant eigenvector of Class 1.

We reflect the examples using a Householder matrix that can be defined as follows. Let $\mathbf{d}^1$ be the dominant eigenvector of the estimated covariance matrix of Class 1 examples. Therefore, an orthogonal symmetric matrix $\mathbf{H}_{p\times p}$ (where p is number of features) exists such that:

$$\mathbf{H} = I - 2\mathbf{u}\mathbf{u}^T \quad \text{where } \mathbf{u} = \frac{\mathbf{e}_1 - \mathbf{d}^1}{\|\mathbf{e}_1 - \mathbf{d}^1\|_2} \quad \text{and } \mathbf{e}_{1p\times 1} = (1, 0, \ldots, 0)^{\mathrm{T}}. \tag{2}$$

Let $\mathfrak{D}_{n\times p}$ be the training example set. The reflected example set $\hat{\mathfrak{D}}_{n\times p}$ is obtained using $\hat{\mathfrak{D}} = \mathfrak{D}\mathbf{H}$. Since $\mathbf{H}_{p\times p}$ is symmetric and orthogonal, a point in the transformed space can be mapped back to the original space at a minimal cost ($\mathbf{HH} = \mathbf{I}$). The mechanism of the Householder reflection is that it reflects vector $\mathbf{d}^1$ onto $\mathbf{e}_1$ by a reflection through the plane perpendicular to vector $\mathbf{e}_1 - \mathbf{d}^1$. The reflected example sets are shown in Figs. 1(b) and 2(b). Each column of $\mathbf{H}$ represents the direction of a coordinate axis in the reflected space. Axis parallel splits are searched along these axes. These splits are oblique in the original space. For each illustration, the best axis parallel split found in the reflected space that is oblique in the original space is shown in Figs. 1(c) and 2(c).

The time complexity of the proposed method is mainly dependent on the time taken to: (1) transform the example set and (2) perform the axis parallel splits. The time complexity of the transformation can be reduced using the Householder reflection. A simple rotation could have been used for the transformation; however, mapping an arbitrary non-zero vector to a coordinate axis can be done more efficiently using a Householder reflection than a rotation.

Therefore, the Householder reflection helps efficient oblique DT construction because, firstly, if the class separation boundary is not aligned with the feature axes, the Householder reflection potentially makes it parallel to a feature axis of the transformed space that can search using axis parallel splits. Secondly, one Householder reflection constructs a new p-dimension feature space and therefore can enhance the search space for split finding. Better splits can potentially be found by expanding the search space but this increases the time complexity of the tree induction process. However, the Householder reflection provides an elegant way to construct new feature spaces where axis parallel splits are effective; hence searching for the best split can be performed with a minimal cost in a higher search space. Thirdly, transformation of example set can be done efficiently.

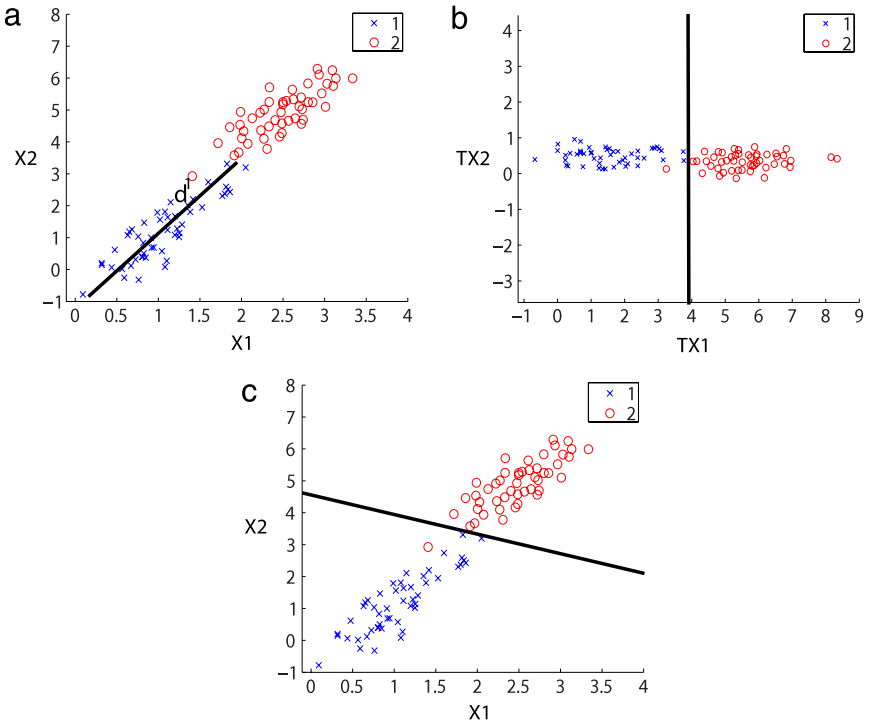

Fig. 2. Mechanism of the Householder reflection. The separating hyperplane is perpendicular to the dominant eigenvector of Class 1. (a) Scatter in the original space. d^1 is the dominant eigenvector of the class covariance matrix of Class 1. (b) Scatter in the reflected space and the best axis parallel split found. (c) Oblique split in the original space.

The axis parallel search space can be further enhanced by using all possible eigenvectors for reflection. For a p-dimensional classification problem with C classes, there are Cp eigenvectors and hence there are Cp p-dimensional search spaces. However, this increases the time complexity of tree induction but provides an opportunity to produce better trees.

Here, we explain the complete algorithm of HHCART. We propose two versions of HHCART: HHCART(A) is based on all possible eigenvectors of all classes and HHCART(D) is based on only the dominant eigenvector of each class. For any given non-terminal node t, let $\mathfrak{D}_t$ and C_t be the set of examples and classes available at that node respectively. At node t, both algorithms first find the best axis parallel split, h_t^{ap}. HHCART(A) then finds all eigenvectors of the estimated covariance matrix for each class, whereas HHCART(D) finds the dominant eigenvector of each class only. A Householder matrix is constructed for each eigenvector. $\mathfrak{D}_t$ is then reflected using each Householder matrix, and axis parallel splits are performed along each coordinate axis in the reflected space. Let h_t be the best axis parallel split found in the reflected space. Out of h_t and h_t^{ap} the better one is chosen as the separating hyperplane at node t. However, if the eigenvector is already parallel to any of the feature axes, no reflection is done. The hyperplane found divides node t into two child nodes. The algorithm is recursively run on all child nodes until each child node satisfies either of the following:

(a) The misclassification rate at the child node is either 0 or is not greater than a user-specified threshold (MisRate); or
(b) The number of examples in the node is less than or equal to a user-specified threshold (MinParent).

In both algorithms, the reflection is done if an eigenvector is not parallel to any feature axis. However, there may be a situation where the eigenvector is almost parallel to a feature axis and thus the reflection may not be beneficial. Therefore, another parameter τ is introduced to the algorithms that can be used as a threshold to determine the parallelism between an eigenvector and a feature axis. That is, if $\|\mathbf{e} - \mathbf{d}\| \leq \tau$, where $\mathbf{e}$ is a basis vector and $\mathbf{d}$ is an eigenvector, no reflection is done. If τ is a small positive value, then those eigenvectors which are nearly parallel to any feature axis are not considered for the reflection because: (1) the axis parallel search in the reflected space may not be more effective than the axis parallel search in the original space and (2) it speeds up the tree induction by avoiding possible duplicate search spaces. In the experiments, τ was set arbitrary to 0.05. However, the user can choose any small positive value or use a separate cross-validation procedure to estimate the optimal value for τ.

An overview of HHCART(A) algorithm at node t is given in Algorithm 1. The time complexity at a node for HHCART(A) in the worst case is $O\left(Cp^2\ (p + n \log n)\right)$ (see Appendix for the derivation). However, if HHCART(D) is used, the time complexity reduces to $O\left(Cnp\ (p + \log n)\right)$. For $n \gg p$, the time complexities of HHCART(A) and HHCART(D) simplify to $O\left(Cp^2 n \log n\right)$ and $O\left(Cnp \log n\right)$ respectively. The time complexity of OC1 in the worst-case scenario is $O\left(pn^2 \log n\right)$ when Max Minority or Sum Minority impurity measures are used. Hence for large, n, both HHCART(A) and HHCART(D) are more efficient than OC1.

Data: Input: Examples at node t, called $\mathfrak{D}_t$, Minparent, MisRate, and $\tau > 0$.
initialisation;
Define N_t = Number of examples in $\mathfrak{D}_t$;
Define mp_t = misclassification rate at node t;
Define C_t = number of classes at node t;
Define p = number of features;
$\Delta(I_{max}) = 0$;
$h_t = empty$;
if *($N_t > Minparent$) and ($MisRate < mp_t$)* **then**
  h_t^{ap} = The best axis parallel split;
  $h_t = h_t^{ap}$;
  $\Delta(I_{max})$ = Impurity reduction due to h_t^{ap};
  for *i=1:C_t* **do**
    Extract the examples that belong to the ith class in $\mathfrak{D}_t$, called D_i;
    Compute the normalised eigenvectors and eigenvalues of estimated covariance matrix for D_i;

$$((d^{1i}, \lambda^{1i}), \ldots (d^{pi}, \lambda^{pi}))$$

    for *j=1:p* **do**
      if $\lambda^{ji} \neq 0$ **then**
        if $\|e_1 - d^{ji}\| \leq \tau$ *or* $\|e_2 - d^{ji}\| \leq \tau$ *or* $\ldots$ *or* $\|e_p - d^{ji}\| \leq \tau$ **then**
          $H_t^{ji} = I$, the Identity matrix;
        else
          Construct the Householder matrix H_t^{ji} using d^{ji} ;
        end
        Reflect $\mathfrak{D}_t : \hat{\mathfrak{D}}_t = \mathfrak{D}_t * H_t^{ji}$;
        Find the best axis parallel hyperplane split, called h_t^{ji};
        if *impurity reduction of* $h_t^{ji} > \Delta(I_{max})$ **then**
          Replace h_t with h_t^{ji}, the best hyperplane found so far;
          Replace $\Delta(I_{max})$ with the impurity reduction of h_t^{ji}
        end
      end
    end
  end
end

Algorithm 1: Overview of the HHCART(A) algorithm at a single node.

3.1. Small samples

As the tree grows, the number of examples at each node usually becomes small. This raises two questions to be answered. (a) Is it worthwhile searching for an oblique split or is an axis parallel split sufficient? (b) Covariance matrices tend to be singular with small samples. The first problem is common for any oblique DT. In the OC1 algorithm, the authors suggest using oblique splits if the number of examples at a node is greater than twice the number of feature variables. The second question has two parts: The effect of a small sample for HHCART is as follows:

1. Lack of information in eigenvectors with zero eigenvalues from a singular covariance matrix;
2. Eigenvectors are not informative for classes with only one example or several examples with the same feature vector.

The first problem can be solved without modifying the method because the reflection is done using the available eigenvectors. For the second problem, these classes are disregarded from eigenanalysis. However, if all the classes suffer from this problem, axis parallel splits are performed.

3.2. Qualitative variables

Data classification problems often contain a mixture of quantitative and qualitative feature variables. Since the class discriminatory information may be contained in both types of feature variable, an effective classifier should be able to handle both types of features in the classification process. For a qualitative feature variable X, the form of the split is given by $X \in A$, where A is a non-empty subset of values taken by X. If a qualitative feature has M non-empty levels, $2^{M-1} - 1$ splits are possible. Axis parallel algorithms that consider qualitative splits can be found in Quinlan (1986). Incorporating qualitative

Table 1
Real example sets with quantitative features, downloaded from the UCI repository.

Dataset	No. of feature	No. of classes	No. of examples
Heart (HRT)	13	2	270
Pima Indian (PIND)	8	2	768
Breast Cancer (BC)	9	2	638
Boston Housing (BH)	13	2	506
Wine (WINE)	13	3	178
BUPA	6	2	345
Balance Scale (BS)	4	3	625
Glass (GLS)	9	7	214
Shuttle (SHUT)	9	7	58 000
Letter (LET)	10	26	20 000
Survival (SUR)	3	2	306

features in oblique splits has not been explored to any great extent. The QUEST algorithm (Loh and Shih, 1997) is capable of finding oblique splits with both qualitative and quantitative features. QUEST transforms each unordered qualitative feature variable into a new ordered quantitative feature variable. Each level of an unordered qualitative feature is mapped to a ordered value called a CRIMCOORD. The exact CRIMCOORD algorithm can be found in Loh and Shih (1997). We implement the same CRIMCOORD algorithm in HHCART to induce oblique splits that contain both qualitative and quantitative features. At each node, a new quantitative feature is constructed for each qualitative feature by mapping its levels to CRIMCOORDS. These new quantitative features are then amalgamated with the existing quantitative features in the example set. The HHCART algorithm can then be applied to find the best oblique split. At each node, the CRIMCOORD corresponding to each level of each qualitative feature is stored. During prediction, the level of each qualitative feature of an unclassified observation is replaced by the corresponding CRIMCOORD attached to each node along its path.

4. Experiments

Two sets of experiments were carried out to compare the performance of HHCART with other DT methods. The first experiment considered quantitative example sets and the second experiment considered example sets with both qualitative and quantitative features. Both HHCART(A) and HHCART(D) methods were considered in the experiments.

4.1. Comparison on example sets having quantitative features only

In this section, we compare the HHCART methods with FDT, OC1, OC1-LC (the OC1 version of Breiman's linear combination methods) and OC1-AP (the OC1 version of axis parallel splits). All of the OC1's methods are freely available in the OC1 system at Murthy et al. (1993) and FDT is freely available at http://alchau.com/research/src/. However, the backward feature elimination process of Breiman's CART-LC method is not included in OC1-LC and hence is somewhat different from the original method. For all the methods, accuracy and tree size are estimated using fivefold cross-validations. FDT uses stratified random sampling to construct cross-validation samples, whereas the OC1 algorithms use simple random sampling. Hence, comparisons of the methods are done separately. In the first case, the HHCART methods are compared with the OC1 algorithms using simple random sampling cross-validations. In the second case, the HHCART methods are compared with FDT using stratified random sampling cross-validations. Experiments were performed on real example sets that were downloaded from Bache and Lichman (2013) and are given in Table 1.

4.1.1. HHCART methods versus OC1 algorithms

For HHCART(A) and HHCART(D), the parameters are set as follows: MinParent = 2, MisRate = 0 (to avoid pre-pruning) and $\tau = 0.05$ (arbitrary). All the algorithms used the Twoing rule as the measure of impurity (Breiman et al., 1984) and Cost complexity pruning (Breiman et al., 1984) with zero standard error. For OC1, the number of restarts and the number of jumps were set to 20 and 5 (default values) respectively. Fivefold cross-validations (based on simple random sampling) were used to estimate the classification accuracy. For each fold, 10% of the training set was used exclusively for pruning. We then used 10 fivefold cross-validations to estimate the accuracy and the size of the tree. Therefore, to estimate accuracy and tree size (number of terminal nodes), the average over 10 runs was used. The results are reported in Table 2 along with the respective standard deviations.

The Shuttle dataset comes with its own training set containing 43,500 examples and a test set with 14,500 examples. Therefore, instead of performing a cross-validation experiment, we induced 10 trees, each using 90% of the training examples for induction and the remaining 10% for pruning. The accuracy of all the trees was estimated using the Shuttle data test set. Since approximately 80% of the examples belong to Class 1, the aim is to achieve an accuracy of 99%–99.9% (Bache and Lichman, 2013).

Table 2 shows the results for our first experiment. The average accuracies and the average tree sizes of 10 fivefold cross-classifications are listed in the table. It is clear that oblique splits reduce the average tree size for all the datasets

Table 2
Results of HHCART and other DT methods. The tree size is measures as the number of terminal nodes.

Dataset	DT	Average accuracy	Average tree size	Dataset	DT	Average accuracy	Average tree size
BS	HHCART(A)	**92.8** ± 1.3	7.4 ± 1.3	PIND	HHCART(A)	73.2 ± 1.4	11.9 ± 6.5
	HHCART(D)	88.3 ± 1.7	12.1 ± 3.3		HHCART(D)	73.7 ± 1.5	11.5 ± 8.4
	OC1	91.9 ± 0.9	8.7 ± 3.4		OC1	73.4 ± 1.0	9.2 ± 5.4
	OC1-AP	78.2 ± 1.3	37.5 ± 16.8		OC1-AP	**73.6** ± 1.4	15.9 ± 8.7
	OC1-LC	84.3 ± 1.5	12.6 ± 6.5		OC1-LC	72.8 ± 1.8	11.4 ± 9.6
BH	HHCART(A)	**83.4** ± 1.2	7.0 ± 2.9	SHUT	HHCART(A)	99.94 ± 0.02	25.4 ± 5.9
	HHCART(D)	82.0 ± 1.1	8.0 ± 2.8		HHCART(D)	99.97 ± 0.05	28.8 ± 4.9
	OC1	82.2 ± 1.2	9.3 ± 3.4		OC1	99.95 ± 0.03	32.6 ± 7.7
	OC1-AP	82.0 ± 0.7	13.0 ± 5.3		OC1-AP	**99.97** ± 0.02	26.5 ± 5.6
	OC1-LC	81.5 ± 1.3	10.6 ± 6.0		OC1-LC	88.4 ± 7.07	44.7 ± 42.4
BC	HHCART(A)	**97.0** ± 0.3	2.3 ± 0.4	WINE	HHCART(A)	**91.4** ± 1.8	3.4 ± 0.3
	HHCART(D)	**97.0** ± 0.3	2.6 ± 1.1		HHCART(D)	88.3 ± 1.8	4.7 ± 0.7
	OC1	95.4 ± 0.5	3.3 ± 1.4		OC1	89.2 ± 2.1	3.5 ± 0.3
	OC1-AP	94.0 ± 0.8	8.3 ± 3.3		OC1-AP	89.2 ± 4.6	4.6 ± 0.6
	OC1-LC	95.5 ± 0.6	3.4 ± 1.6		OC1-LC	89.4 ± 2.7	3.8 ± 0.6
BUPA	HHCART(A)	64.9 ± 3.0	7.8 ± 1.5	LET	HHCART(A)	82.1 ± 0.3	759.2 ± 88.1
	HHCART(D)	64.8 ± 2.1	10.2 ± 3.0		HHCART(D)	83.1 ± 0.3	1135.9 ± 122
	OC1	**66.9** ± 2.2	8.9 ± 6.1		OC1	83.6 ± 0.4	1197.2 ± 88.9
	OC1-AP	64.7 ± 2.5	13.2 ± 10.5		OC1-AP	**86.3** ± 0.3	1611.7 ± 60.0
	OC1-LC	64.4 ± 2.4	8.9 ± 3.6		OC1-LC	84.5 ± 0.2	1332.6 ± 146.3
GLS	HHCART(A)	61.9 ± 3.1	8.8 ± 3.1	SUR	HHCART(A)	**72.5** ± 1.7	6.5 ± 2.6
	HHCART(D)	61.7 ± 3.4	10.7 ± 2.7		HHCART(D)	72.2 ± 2.2	10.6 ± 5.5
	OC1	61.1 ± 3.5	10.8 ± 4.3		OC1	71.0 ± 2.1	6.4 ± 3.5
	OC1-AP	64.6 ± 3.9	14.6 ± 8.7		OC1-AP	71.9 ± 1.5	10.7 ± 6.5
	OC1-LC	**67.4** ± 2.0	12.0 ± 3.6		OC1-LC	70.2 ± 2.4	8.1 ± 4.4
HRT	HHCART(A)	75.0 ± 2.3	5.5 ± 1.9				
	HHCART(D)	75.2 ± 3.6	8.1 ± 3.1				
	OC1	**77.1** ± 2.5	3.6 ± 1.0				
	OC1-AP	76.3 ± 2.3	6.7 ± 2.4				
	OC1-LC	76.3 ± 2.5	4.0 ± 1.1				

while increasing the accuracy for most datasets. The average accuracy of HHCART(A) is significantly (more than 2 standard deviations) higher than all the other methods tested for the BC dataset except for HHCART(D). For all other example sets, there is no significant difference between HHCART(A) and the other methods in terms of the average accuracies, except for LET. However, the average accuracy of HHCART(A) is higher than the other methods for the BS, BH, WINE and SUR example sets.

The average tree sizes of HHCART(A) are consistently smaller than the average tree sizes of other methods except for the HRT, PIND and SUR datasets. Therefore, the performance of HHCART(A) with respect to accuracy and tree size is better than the other methods for the BS, BH, BC, WINE and SUR datasets.

Eight of the 11 datasets have at least eight features. For five (BH, BC, PIND, WINE and SHUT) of these relatively high-dimensional datasets, the performance of HHCART(A) is comparable with OC1 and OC1-LC. Therefore, we can conclude that the proposed method works well in relatively high-dimensional feature spaces.

For all the datasets except BS and WINE, HHCART(D) performs as well as HHCART(A) in terms of the average accuracy. In addition, the tree sizes of HHCART(D) are comparable with those produced by HHCART(A) except for the BS, BUPA, HRT, LET and SUR datasets. The performance of HHCART(D) is similar to OC1 with respect to both the accuracy and tree size for all the datasets except the BS, BUPA, HRT and SUR datasets. The time complexity of HHCART(A) is higher than that of HHCART(D) by a factor of $O(p)$. The results show that HHCART(D) produces DTs with similar accuracies and sizes to HHCART(A) and OC1 for most of the datasets. Hence HHCART(D) would be a more efficient method to use for higher-dimensional problems.

Some of the example sets have imbalanced class distributions. Hence, in addition to the average accuracy, the measures such as the true positive rate, the true negative rate and the F-measure (F-measure is the harmonic mean of precision and recall.) are computed and reported in Tables 3 and 4 respectively. However, OC1, OC1-LC and OC1-AP do not have an inbuilt functions to obtain the true negative rate and the F-measure. Therefore, only the true positive rate is reported for the OC1 algorithms. All additional measures are reported for HHCART(A) and HHCART(D). The true positive rate and the true negative rate are computed for each class. However, for the F-measure, a common value is obtained by combining the class-wise F-measures. Two combined F-measures are proposed in Sokolova and Lapalme (2009) namely: (1) the macro F-measure and (2) the micro F-measure. The macro F-measure is suitable for classification problems with imbalanced class distributions (Sokolova and Lapalme, 2009). Therefore, in this study, the macro F-measure is used as the combined F-measure. All values in Tables 3 and 4 are the averages of 10 repetition of fivefold cross-validations. The true negative values for HHCART(A) and HHCART(D) are given in HHCART(A) and HHCART(D) columns in Table 3 within parentheses.

It is clear from Table 3 that in general, minority classes have lower true positive rates for all methods. For the BS and BC example sets, HHCART(A) has higher true positive rates for minority classes, whereas for the BUPA and PIND example sets, OC1 has higher true positive rates for minority classes. HHCART(D) performs well for the BC, SHUT and SUR example

Table 3
Comparison of true positive rates. True negative rates for HHCART(A) and HHCART(D) are given in the parenthesis in the respective columns. Standard deviations which are less than 0.2 are omitted. Algorithms are ranked based on the true positive rates of the minority classes and the mean rank is reported. The LET example set has 26 classes and is therefore omitted from the table due to space constraints.

Example set	Class (No. of examples)	HHCART(A)	HHCART(D)	OC1	OC1-AP	OC1-LC
	1(49)	0.66 (0.97)	0.34 (0.96)	0.6	0.02	0.1
BS	2(288)	0.95 (0.96)	0.92 (0.93)	0.95	0.84	0.9
	3(288)	0.94 (0.96)	0.93 (0.91)	0.94	0.85	0.91
BH	1(246)	0.83 (0.83)	0.84 (0.80)	0.83	0.84	0.82
	2(260)	0.83 (0.83)	0.80 (0.84)	0.81	0.81	0.81
BC	1(444)	0.96 (0.98)	0.96 (0.98)	0.95	0.95	0.95
	2(239)	0.98(0.96)	0.98 (0.96)	0.95	0.92	0.96
BUPA	1(145)	0.50 (0.76)	0.52 (0.74)	0.55	0.48	0.46
	2(200)	0.76 (0.50)	0.74 (0.52)	0.76	0.77	0.78
GLS	1(70)	0.75 (0.79)	0.7 (0.80)	0.69	0.78	0.76
	2(76)	0.63 (0.79)	0.64 (0.77)	0.61	0.59	0.68
	3(17)	0.08 (0.97)	0.1 (0.97)	0.12	0.19	0.18
	5(13)	0.32 (0.97)	0.43 (0.97)	0.47	0.59	0.52
	6(9)	0.3 (0.98)	0.34 ± 0.22 (0.98)	0.52 ± 0.27	0.52	0.68 ± 0.22
	7(29)	0.82 (0.94)	0.81 (0.95)	0.8	0.81	0.82
HRT	1(150)	0.78 (0.71)	0.82 (0.67)	0.8	0.8	0.77
	2(120)	0.71 (0.78)	0.67 (0.82)	0.73	0.72	0.75
PIND	1(500)	0.83 (.54)	0.84 (0.55)	0.82	0.83	0.82
	2(268)	0.54 (0.83)	0.55 (0.84)	0.58	0.57	0.56
SHUT	1(11478)	1 (1)	1 (1)	1	1	0.97
	2(13)	0.85 (1)	0.9 (1)	0.85	0.92	0.2 ± 0.24
	3(39)	0.97 (1)	0.99 (1)	0.97	0.99	0.26 ± 0.37
	4(2155)	1 (1)	1 (1)	1	1	0.57 ± 0.34
	5(809)	1 (1)	0.1 (1)	1	1	0.39 ± 0.43
	6(4)	0.45 ± 0.49 (1)	0.85 ± 0.31 (1)	0.88 ± 0.24	0.7 ± 0.48	0.2
	7(2)	1 (1)	1 (1)	1	1	0.25 ± 0.42
WINE	1(59)	0.93 (0.96)	0.91 (0.94)	0.91	0.92	0.93
	2(71)	0.88 (0.95)	0.85 (0.93)	0.87	0.87	0.87
	3(48)	0.95 (0.96)	0.88 (0.95)	0.9	0.88	0.88
SUR	1(225)	0.89 (0.28)	0.87 (0.31)	0.87	0.88	0.87
	2(81)	0.28 (0.89)	0.31 (0.87)	0.27	0.28	0.26
Mean rank		3.1	2.4	2.5	3.1	3.7

Table 4
Comparison of F-measures.

Example set	HHCART(A)	HHCART(D)
BS	0.85 ± 0.03	0.74 ± 0.04
BH	0.83 ± 0.01	0.82 ± 0.01
BC	0.97 ± 0.01	0.97 ± 0.003
BUPA	0.63 ± 0.03	0.63 ± 0.02
GLS	0.50 ± 0.08	0.52 ± 0.05
HRT	0.74 ± 0.02	0.74 ± 0.04
PIND	0.7 ± 0.02	0.70 ± 0.01
SHUT	0.89 ± 0.08	0.97 ± 0.03
WINE	0.91 ± 0.02	0.88 ± 0.02
SUR	0.6 ± 0.03	0.6 ± 0.02

sets. CART-LC and OC1-AP produces a higher classification accuracy for the minority classes in the GLS and SHUT example sets respectively. The BH, WINE, HRT and LET example sets are excluded from the comparison, as they have nearly uniform class distributions. Each algorithm is ranked based on the true positive rates of the minority classes and the mean rank for each algorithm is given in the last row of Table 3. For example sets with several minority classes (GLS and SHUT), the algorithms are ranked by considering the overall performance of the true positive rates of the minority classes. The HHCART methods perform well when the class imbalance is not too large. However, based on the mean rank, the OC1 and HHCART(D) algorithms produce better true positive rates for the minority classes.

Table 4 shows the F-measures (along with the respective standard deviations) obtained for HHCART(A) and HHCART(D) on each example set. According to the results, it can be seen that the performances of both algorithms on the example sets are similar except BS and SHUT. HHCART(A) has a higher F-measure for BS, whereas HHCART(D) has a higher F-measure for SHUT. For all other example sets the F-measures are more or less the same.

Table 5
Results of HHCART methods and FDT. HH(A) stands for HHCART(A) and HH(D) stands for HHCART(D).

Example set	Decision tree	Average accuracy	Average tree size	Classes (size)	Class 1 true positive	Class 2 true positive	F-measure
BH	HH(A)	83.1 ± 1.2	6.7 ± 3.1	1(246), 2 (260)	0.85 ± 0.03	0.81 ± 0.02	0.83 ± 0.01
	HH(D)	82 ± 1.0	7.3 ± 4.1		0.85 ± 0.04	0.78 ± 0.03	0.82 ± 0.01
	FDT	83.9 ± 0.76	21		0.83 ± 0.02	0.85 ± 0.02	0.84 ± 0.01
BC	HH(A)	96.9 ± 0.3	2.7 ± 0.7	1 (444), 2 (239)	0.96 ± 0.002	0.97 ± 0.01	0.97 ± 0.003
	HH(D)	97 ± 0.4	3.0 ± 1.4		0.97 ± 0.002	0.98 ± 0.01	0.97 ± 0.004
	FDT	95.7 ± 0.7	7		0.97 ± 0.01	0.94 ± 0.02	0.95 ± 0.01
BUPA	HH(A)	66.1 ± 2.6	8.0 ± 3.6	1 (145), 2 (200)	0.5 ± 0.07	0.78 ± 0.06	0.65 ± 0.03
	HH(D)	65.2 ± 2.5	12.54 ± 5.4		0.53 ± 0.04	0.74 ± 0.03	0.64 ± 0.03
	FDT	63.2 ± 1.7	43		0.57 ± 0.04	0.68 ± 0.03	0.62 ± 0.02
PIND	HH(A)	73.5 ± 1.0	9.1 ± 4.7	1 (500), 2 (268)	0.84 ± 0.03	0.54 ± 0.04	0.7 ± 0.01
	HH(D)	73.7 ± 1.1	16.8 ± 7.1		0.83 ± 0.03	0.56 ± 0.04	0.7 ± 0.01
	FDT	69 ± 1.3	67		0.76 ± 0.02	0.56 ± 0.03	0.66 ± 0.01
HRT	HH(A)	75.2 ± 1.7	5.2 ± 1.2	1 (150), 2 (120)	0.78 ± 0.03	0.71 ± 0.03	0.75 ± 0.02
	HH(D)	75.4 ± 2.8	7.8 ± 3.2		0.78 ± 0.05	0.71 ± 0.04	0.75 ± 0.03
	FDT	75.7 ± 1.7	20		0.77 ± 0.01	0.75 ± 0.03	0.76 ± 0.02
SUR	HH(A)	72.4 ± 1.9	8.4 ± 4.1	1 (225), 2 (81)	0.88 ± 0.03	0.29 ± 0.05	0.60 ± 0.02
	HH(D)	72.8 ± 1.3	7.2 ± 2.9		0.89 ± 0.01	0.28 ± 0.03	0.6 ± 0.02
	FDT	67.5 ± 1.8	42		0.82 ± 0.02	0.31 ± 0.04	0.56 ± 0.03

4.1.2. HHCART methods versus FDT

The parameters of both HHCART methods were set as described in Section 4.1.1. The parameters of FDT were set as in López-Chau et al. (2013) and they are as follows: Minimum number of examples per terminal node = 2, the pruning method was pessimistic pruning with a confidence factor of 0.25 (default) and sub-tree raising was set to true. The comparison was done on average accuracy, average tree size (number of terminal nodes), the F-measure and the true positive rate. FDT estimates all these parameters (except the tree size) using stratified random sampling-based cross-validations. The cross-validation estimate for the tree size is not available. Therefore, the size of the tree built on the entire example set is reported. Ten, fivefold cross-validations are run and hence the results given in Table 5 are averaged over 10 runs. Since FDT is only capable of classifying two-class problems, the comparisons are made on example sets with two classes.

It is evident from Table 5 that the trees produced by HHCART(A) and HHCART(D) are substantially smaller than those of FDT. The average accuracy of HHCART(A) for BC, PIND and SUR are significantly higher (more than one standard deviation) than that of FDT. Furthermore, HHCART(A) produces higher F-measures for BC, BUPA, PIND and SUR, whereas FDT had higher F-measures for BH and HRT. FDT produced higher true positive rates for the minority classes of BUPA, HRT and SUR, whereas HHCART(D) produced a higher true positive rate for BC and PIND. HHCART(A) has lower true positive rates for minority classes than FDT. However, HHCART(A) has higher true positive rates for majority classes. Hence, based on the overall performances (F-measure, accuracy and tree size), the HHCART methods produce better results than FDT.

4.2. Comparison on example sets with qualitative and quantitative features

Experiments were performed to study the performance of the HHCART methods when the training examples contained both qualitative and quantitative features. Since FDT, OC1, OC1-AP and OC1-LC are not designed to handle oblique splits containing both qualitative and qualitative features, QUEST (Loh and Shih, 1997) was used for comparison purposes. Example sets were downloaded from Bache and Lichman (2013) and are given in Table 6. Ten, fivefold cross-validations were used and the average accuracies and tree sizes (over 10 cross-validations) are reported in Table 7. The Income dataset comes with its own training and testing set of 30,162 and 15,060 examples respectively. We induced 10 trees, each using 90% of the training examples; the remaining 10% was used for pruning. The accuracy of all the trees was estimated using the same test set. QUEST uses the following parameter setting: estimated priors, unit misclassification cost, zero standard error for pruning, linear splits, linear discriminant analysis for the split point, minimum node size for splitting = 2. The parameters of the HHCART methods were set as described in Section 4.1.1. For the Income dataset, HHCART(A)'s performance was significantly (more than 2 standard deviations) better than QUEST both in terms of the average accuracy and average tree size. For the other two datasets, HHCART(A) produces comparable accuracies with smaller trees. These results also suggest that the HHCART algorithms perform well in relatively high dimensions. Though HHCART(D) produces larger trees compared with HHCART(A), its classification accuracy is comparable with that of HHCART(A).

5. Conclusions and discussion

In this work, we have presented a new way of inducing oblique DTs called HHCART. It uses the eigenvectors of the estimated covariance matrices of the respective classes to define a Householder matrix that is used to reflect the examples so that axis parallel splits can be searched. Two versions of HHCART have been presented: HHCART(A) uses all possible

Table 6
Real example sets with qualitative and quantitative features, downloaded from UCI Repository.

Dataset	No. of features (no. of Qualitative)	No. of classes	No. of examples
Income	14(8)	2	45 222
Bank	16(9)	2	45 211
StatLog	14(8)	2	690

Table 7
Results of HHCART and QUEST. The tree size is measured as the number of terminal nodes.

Dataset	Decision Tree	Average accuracy	Average tree size
Income	HHCART(A)	85.1 ± 0.2	32.7 ± 12.9
	HHCART(D)	$\mathbf{85.5} \pm 0.2$	59.5 ± 19.7
	QUEST	83.9 ± 0.2	68.0 ± 23.1
Bank	HHCART(A)	90.2 ± 0.1	22.6 ± 11.9
	HHCART(D)	$\mathbf{90.4} \pm 0.1$	44.4 ± 14.2
	QUEST	90.1 ± 0.1	27.0 ± 15.2
StatLog	HHCART(A)	85.1 ± 0.9	5.6 ± 1.9
	HHCART(D)	$\mathbf{85.8} \pm 0.7$	6.5 ± 3.0
	QUEST	85.7 ± 0.9	6.1 ± 3.6

eigenvectors of the estimated covariance matrices of respective classes, whereas HHCART(D) uses only the dominant eigenvector of each class. Based on the empirical results obtained, it is clear that both HHCART methods perform well in terms of accuracy and tree size. Furthermore, as discussed in Section 2.1, optimisation algorithms based DTs take considerable time to find the best split at a node. For example, the SADT and OC1 algorithms iterate many times until a locally optimal point of $\Delta(I)$ is found. In particular, OC1 takes $O(pn^2 \log n)$ time to find the best split at a non-terminal node (If Max Minority or Sum Minority functions are used as impurity functions), whereas HHCART(A) takes $O(Cp^2 n \log n)$ time when $n \gg p$. Therefore, the ratio of the two time complexities HHCART(A):OC1 is $O(cp) : O(n)$. Furthermore, HHCART is capable of classifying example sets with both qualitative and quantitative features and this is useful in a diverse range of applications. Hence, the proposed heuristic algorithms are competitive alternatives for the existing oblique decision trees especially for those based on optimisation algorithms.

According to the true positive rates, the HHCART methods perform well when the class imbalance is not too large. Both HHCART methods have similar F-measure values and hence both methods have similar recall and precision capacities. Moreover, based on the two-class classification results, the performance of the HHCART methods is better than that of FDT in terms of accuracy, tree size and the F-measure.

HHCART is effective in data classification if class orientation is properly captured by the eigenvectors. However, the eigenvectors may fail to capture class orientation if outliers and/or clusters are present in a class. Eigenvectors can be influenced by outliers, making them ineffective at capturing class orientation if outliers are present. This is most evident in classes with few examples, where a single outlier can dramatically change the eigenvectors for the class. Clusters within a class can also affect the proposed heuristic because each cluster may have an orientation that is different from the orientation of the entire class. Hence, defining splitting directions using the orientation of the entire class may be ineffective.

Appendix. Time and space complexity of HHCART

Here, we derive the maximal time complexity at a node of HHCART(A) and HHCART(D). Assume there are n examples with p quantitative features and C classes at the node.

A.1. Time complexity of HHCART

1. **HHCART(A)** and **HHCART(D)**—The complexity for constructing the estimated covariance matrix for one class of examples is $O(np^2)$. For C classes, the complexity is $O(Cnp^2)$.
2. **HHCART(A)**—The complexity of the complete eigenanalysis for one class of examples is $O(p^3)$. For C classes, the complexity is $O(Cp^3)$.
   HHCART(D)—The complexity for finding the dominant eigenvector for one class of examples is $O(p^2)$. For C classes, the complexity is $O(Cp^2)$.
3. **HHCART(A)**—The complexity for the reflection of n examples using one Householder matrix is $O(np^2)$. Since there are Cp Householder matrices, the complexity is $O(Cnp^3)$.
   HHCART(D)—The complexity for the reflection of n examples using one Householder matrix is $O(np^2)$. For C Householder matrices, the complexity is $O(Cnp^2)$.
4. **HHCART(A)**—The complexity of finding the best axis parallel splits for one reflected space is $O(p(n + n \log n))$. That is, along one dimension, sorting the examples takes $O(n \log n)$ time and impurity function evaluation takes a maximum of $O(n)$ time. Hence, the total time along p dimensions (one reflected space) is $O(p(n + n \log n))$. Since there are Cp reflected spaces, the complexity is $O(Cp^2(n + n \log n)) = O(Cp^2 n(1 + \log n)) = O(Cp^2 n \log n)$.

HHCART(D)—The complexity of finding the best axis parallel splits for one reflected space is $O(p(n + n\log n))$. For C classes, the complexity is $O(Cp(n + n\log n)) = O(pn(1 + \log n)) = O(Cpn\log n)$.

5. **HHCART(A)**—The maximal time complexity at a node is therefore $O(Cnp^2) + O(Cp^3) + O(Cnp^2) + O(Cp^2 n\log n) = O(Cp^2(p + n\log n))$.
   HHCART(D)—The maximal time complexity at a node is therefore $O(Cnp^2) + O(Cp^2) + O(Cnp) + O(Cpn\log n) = O(Cnp(p + \log n))$.

A.2. Space complexity of HHCART

1. The space required for storing the entire example set is $O(np)$.
2. The space required for the examples in one transformed space $O(np)$. There are Cp transformed spaces. However, axis parallel splits are performed in one space after the other. Therefore, once the search of one transformed space is completed, examples in that space can be deleted. Hence the space complexity remains at $O(np)$.
3. The final decision tree holds some information at each node. The largest tree has n nodes and each node holds: (a) the class label, (b) the class distribution vector (C-dimensional) and (c) the status of the node whether it is a terminal or non-terminal node. All this information requires the maximal space complexity of $O(C)$. Moreover, each non-terminal node holds a $p + 1$-dimensional coefficient vector of the separating hyperplane and there is a maximum of $(n - 2)$ non-terminal nodes (for the binary trees of minimum node size 2). Hence, the space requirement for holding the hyperplanes is $O((p + 1)(n - 2))$.
4. Therefore, the total space complexity of HHCART is: $O(np) + O(np) + O(C) + O((p+1)(n-2)) = O(np) + O(C) = O(np)$.

References

Amasyah, M., Ersoy, O., 2008. Cline: A new decision-tree family. IEEE Trans. Neural Netw. 19 (2), 356–363.
Bache, K., Lichman, M., 2013. UCI machine learning repository. URL: http://archive.ics.uci.edu/ml.
Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A., 1984. Classification and Regression Trees. CRC Press, New York.
Gama, J., Brazdil, P., 1999. Linear tree. Intell. Data Anal. 3 (1), 1–22.
Heath, D., Kasif, S., Salzberg, S., 1993. Induction of oblique decision trees. In: Proceedings of the 13th International Joint Conference on Artificial Intelligence, pp. 1002–1007.
Hyafil, L., Rivest, R.L., 1976. Constructing optimal binary decision trees is NP-complete. Inform. Process. Lett. 5 (1), 15–17.
Ittner, A., Schlosser, M., 1996. Non-linear decision trees-NDT. In: 13th International Conference on Machine Learning. Citeseer, Italy, pp. 252–257.
Iyengar, V.S., 1999. HOT: Heuristics for oblique trees. In: 11th International conferance on Tools with Artificial Intelligence. IEEE, New York, pp. 91–98.
Kolakowska, A., Malina, W., 2005. Fisher sequential classifiers. IEEE Trans. Syst. Man Cybern. B 35 (5), 988–998.
Li, Y., Dong, M., Kothari, R., 2005. Classifiability-based omnivariate decision trees. IEEE Trans. Neural Netw. 16 (6), 1547–1560.
Li, X.-B., Sweigart, J.R., Teng, J.T., Donohue, J.M., Thombs, L.A., Wang, S.M., 2003. Multivariate decision trees using linear discriminants and tabu search. IEEE Trans. Syst. Man Cybern. A 33 (2), 194–205.
Loh, W.-Y., Shih, Y.-S., 1997. Split selection methods for classification trees. Statist. Sinica 7 (4), 815–840.
López-Chau, A., Cervantes, J., López-García, L., Lamont, F.G., 2013. Fisher's decision tree. Expert Syst. Appl. 40 (16), 6283–6291.
Manwani, N., Sastry, P., 2012. Geometric decision tree. IEEE Trans. Syst. Man Cybern. B 42 (1), 181–192.
Murthy, S.K., Kasif, S., Salzberg, S., 1993. The oc1 decision tree software system. URL: http://www.cbcb.umd.edu/salzberg/announce-oc1.html.
Murthy, S.K., Kasif, S., Salzberg, S., 1994. A system for induction of oblique decision trees. J. Artificial Intelligence Res. 2 (1), 1–32.
Quinlan, J.R., 1986. Induction of decision trees. Mach. Learn. 1 (1), 81–106.
Robertson, B., Price, C., Reale, M., 2013. CARTopt: a random search method for nonsmooth unconstrained optimization. Comput. Optim. Appl. 56 (2), 291–315.
Rokach, L., Maimon, O., 2005. Top-down induction of decision trees classifiers—a survey. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 35 (4), 476–487.
Sokolova, M., Lapalme, G., 2009. A systematic analysis of performance measures for classification tasks. Inf. Process. Manage. 45 (4), 427–437.