Expert Systems with Applications 40 (2013) 6283–6291

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

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

Fisher's decision tree

Asdrúbal López-Chau [a,*], Jair Cervantes [b], Lourdes López-García [a], Farid García Lamont [b]

[a] Centro Universitario UAEM Zumpango, Camino viejo a Jilotzingo continuación calle Rayón, Valle Hermoso. Zumpango, Estado de México C.P. 55600, Mexico
[b] Centro Universitario UAEM Texcoco, Av. Jardn Zumpango s/n, Fracc. El Tejocote, Texcoco, México C.P. 56259, Mexico

ARTICLE INFO

Keywords:
Oblique decision tree
Fisher's linear discriminant
C4.5

ABSTRACT

Univariate decision trees are classifiers currently used in many data mining applications. This classifier discovers partitions in the input space via hyperplanes that are orthogonal to the axes of attributes, producing a model that can be understood by human experts. One disadvantage of univariate decision trees is that they produce complex and inaccurate models when decision boundaries are not orthogonal to axes. In this paper we introduce the Fisher's Tree, it is a classifier that takes advantage of dimensionality reduction of Fisher's linear discriminant and uses the decomposition strategy of decision trees, to come up with an oblique decision tree. Our proposal generates an artificial attribute that is used to split the data in a recursive way.

The Fisher's decision tree induces oblique trees whose accuracy, size, number of leaves and training time are competitive with respect to other decision trees reported in the literature. We use more than ten public available data sets to demonstrate the effectiveness of our method.

1. Introduction

Classification is an important task in pattern recognition, machine learning and data mining. It consists in predicting the category, class or label of previously unseen objects. Methods that implement this task are known as classifiers.

Decision trees (DT) are powerful classifiers widely used in many applications such as intrusion detection (Li, 2005), stock trading (Wu, Lin, & Lin, 2006), health (Cruz-Ramírez, Acosta-Mesa, Carrillo-Calvet, & Barrientos-Martínez, 2009; Fan, Chang, Lin, & Hsieh, 2011), fault detection of induction motors (Pomorski & Perche, 2001) and online purchasing behavior of users (Xiaohu, Lele, & Nianfeng, 2012).

DT have some advantages over other methods such as to be tolerant to noise, support missing values and be able to produce models easily interpreted by human beings. In addition, the induction of DT is not costly.

The general algorithm to induce decision trees from data works separating or splitting the data recursively, in such way that partitions are increasingly purer up to certain criterion is satisfied.

Classical DT such as CART (Breiman, Friedman, Olshen, & Stone, 1984) and C4.5 (Quinlan, 1993) split the data using hyperplanes that are orthogonal to axes, this kind of trees is known as univariate or axis-parallel DT. The splits are found testing every axis at each possible splitting point.

They are two intrinsic characteristics of axis-parallel decision trees: (a) The separating boundaries may result in complicated or inaccurate trees if the decision boundary of data set is described by hyperplanes not orthogonal to axes of features (Cantu-Paz & Kamath, 2003), (b) The computational cost increases whether the attributes are of numeric type or there is a large number of features.

The oblique (or multivariate) trees are DT whose separating hyperplanes are not necessarily parallel to axes, in contrast, the hyperplanes can have an arbitrary direction. The induced oblique DT are usually smaller and in certain cases they achieve better classification accuracy than univariate DT, however the characteristic to be interpretable vanishes.

Several algorithms to induce oblique DT have been proposed before, they can be categorized according to the technique used to create the decision boundaries. One type uses an optimization step to determine separating hyperplanes (Murthy, Kasif, & Salzberg, 1994; Shah & Sastry, 1999; Menkovski, Christou, & Efremidis, 2008; Manwani & Sastry, 2009), usually these methods are costly in terms of complexity. Other use heuristics (Iyengar, 1999) to avoid the computational cost, the accuracy achieved is comparable to univariate trees. Other methods apply genetic algorithms to produce more accurate trees (Cantu-Paz & Kamath, 2003; Shali, Kangavari, & Bina, 2007), an important remark with this approach is that the efficiency of greedy methods to induce trees is not preserved.

In our case we use projection on a vector that maximizes the distance between the means, which corresponds to the vector of Fisher's linear discriminant. Similar methods such as (Henrichon

* Corresponding author. Address: Universidad Autónoma del Estado de México, Centro Universitario Zumpango. Tel.: +52 555919174140.
E-mail address: alchau@uaemex.mx (A. López-Chau).

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.05.044

& Fu, 1969; Gama, 1997, 1999) follow this approach however they need to add more attributes to the training set and they use the first principal components.

1.1. Our contribution

In this paper we present the design, implementation and evaluation of an oblique decision tree induction method that is designed for binary classification problems.

The main characteristics of our method are the following: (a) The Fisher's Linear Discriminant Tree induces oblique trees from data, using only one artificial attribute to split the data, recursively. (b) The test of split points is realized on the artificial attribute, so the number of tests is equal to the number of objects at each node. (c) The supported attribute type is numeric, this common in most oblique trees. (d) The method is designed for binary classification.

Our method is programmed using in Java programming language and based on the Weka platform. An executable jar file of Fisher's decision tree is public available at http://www.mediafire.com/?g1meq364bsvs18l.[1]

The Fisher's decision tree works as follows:

1. A vector (ω) that maximizes the distance between mean of two classes is computed.
2. All objects in data set are projected on ω.
3. An hyperplane that best splits the data is computed using the entropy gain criterion.
4. A sub tree is grown in each partition.
5. The steps are recursively applied.

To the best of our knowledge this is the first time that a decision tree method is implemented as presented here and also the complete implementation and source code is made public available, other proposals with a similar approach are shown in the next section.

The rest of this paper is organized as follows. The related work is shown in the Section 2. A brief review on decision trees and Fisher's linear discriminant is presented in Section 3, the proposed method is explained in Section 4. The results of experiments and discussion are in Section 5, finally the conclusions and future work are in Section 6.

2. Related work

The paper presented in Henrichon and Fu (1969) is the first intent to induce models from data that resemble current decision trees. In that work authors gave "*examples of indications of some methods which may be useful for improving recognition accuracies*", they propose to use "linear transgeneration units"' that can be seen as linear combinations of inputs. In Henrichon and Fu (1969) it is mentioned that observation points can be mapped into a line corresponding to the direction of the eigenvector associated with the largest eigenvalue of an estimated covariance matrix, which is similar to our work, however, there are some differences: first, some features are added to the training set (maximum eigenvector, linear discriminant a Quadratic form), in our proposal this is not necessary. Second, they use pre-prunning technique, whereas our approach use post-pruning. Third, the other method uses the first principal components.

The Robust Linear Decision Trees (John, 1994) is a method that can induce oblique decision trees, the method is based on a softened linear discriminant function to discover best split points, such function has the form (1). They iteratively solve an optimization problem using gradients.

$$g_\theta = 1/(1 + exp(-\theta^T x)) \tag{1}$$

Another idea to build oblique decision trees was presented in 1984 in the Classification and Regression Trees methodology (Breiman et al., 1984), the method was called CART-LC (CART Linear Combination). In that work the set of allowable splits was extended to include all linear combinations splits, which have the form (2), this model is currently used in many other decision trees.

The CART-LC was reported to achieve competitive accuracy with respect to other classification methods (see Breiman et al., 1984, p. 39), however solving the set of linear combinations using deterministic heuristics search is computational costly.

$$\sum_{i=1}^{m} a_m x_m \leqslant C? \tag{2}$$

The LMDT algorithm (Utgoff & Brodley, 1991) solves a set of R multi-class linear discriminant functions at each node, they are collectively used to assign each instance to one of the R classes. A problem with LMDT is that convergence can not be guaranteed, so a heuristic must be used to determine when a node has stabilized.

The OC1 (Murthy et al., 1994) method combines hill climbing with randomization to find good oblique separating hyperplanes at each node of a decision tree, the key idea relies on perturbing each of the parameters of the hyperplane split rules individually several times to achieve the optimization. The OC1 computes both the axis-parallel splits and then the separating hyperplanes to decide which to use.

In Setiono and Liu (1999) an artificial neural network is used to build an oblique decision tree as follows, the data is first used to train a neural network having only one hidden layer. After that the hidden unit activation values of the network are used to build a decision tree. Then each test at the nodes of the tree is replaced by a linear combination of initial attributes.

The APDT algorithm in Shah and Sastry (1999) uses the concept "degree of linear separability", it uses a function that minimizes non separable patterns in children nodes. When a child node is linearly separable, the split that puts more patterns in that child node is preferred, preserving linear separability.

The Linear Discriminant Trees presented in Yildiz and Alpaydin (2000) resembles our method, but it uses the K main components of PCA, and addition it requires to train a neural network for the splits.

In the HOT heuristics (Iyengar, 1999) the projections on a vector are used to create oblique splits. The vector is produced joining the centroids of the two adjacent regions discovered by an axis-parallel decision tree.

The Oblique Linear Tree and Discriminant Tree were proposed in Gama (1997, 1999) respectively. The main differences with our method is that they do not use the same discriminant function that we apply and in their method the number of attributes can change along the tree.

In Vadera (2005) an oblique decision tree is developed, that algorithm uses the linear discriminant analysis to identify oblique relationships between (continuous) attributes and then carries out an appropriate modication to ensure that the resulting tree errors on the side of safety. Because that algorithm is oriented to optimize the cost of misclassification it searches in every attribute, producing an accurate classifier at expense of investing more time to induce the tree.

The LST algorithm (He, Xu, & Chen, 2008) transforms local data from its original attribute space to the singular vector space, it uses linear independent combinations of the original attributes to build a oblique tree. From the results in He et al. (2008), it can been

[1] For obtaining the complete source code, write a mail to alchau@uaemex.mx.

observed that the local singular vector space constructed by LST does not necessarily lead to the better classification result.

The algorithm in Menkovski et al. (2008) is based on a combination of support vector machine and C4.5 algorithm. This can become a bottleneck because it requires to solve quadratic programming problems, which computation has a complexity between $O(n^{1.5})$ and $O(n^2)$.

Recently, other methods to induce oblique decision trees using evolutionary algorithms have been proposed (Vadera, 2010), see (Barros, Basgalupp, de Carvalho, & Freitas, 2011) for a survey on these techniques. We think that comparison of greedy methods to induce decision trees against evolutionary methods is not just, because although the reported results in Barros et al. (2011) show that the second methods produce better trees in terms of accuracy, these methods explore the space of solutions in a different way, usually using randomly procedures that allow to scape from local optima at expense of increasing computational cost.

3. Preliminaries

The core of proposed method is based on a combination of classic univariate decision tree and the Fisher's discriminant classifier, before presenting our method, an overview of these two topics is exposed.

3.1. Decision trees

Decision tree is a classifier method able to produce models that can be comprehensible by human experts (Breiman et al., 1984). Among the advantages of decision trees over other classification methods are (Cios, Pedrycz, Swiniarski, & Kurgan, 2007): robustness to noise, ability to deal with redundant attributes, generalization ability via post-pruning strategy and a low computational cost, this last is more notorious when decision trees are trained using data sets with nominal attributes.

A trained decision tree is a tree structure that consists of a root node, links or branches, internal nodes and leaves. The root node is the topmost node in the tree structure, the branches connect the internal nodes (root, internal and leaves), finally the leaves are terminal nodes which have not further branches.

Classic decision tree algorithms such as CART, C4.5 and ID3, create or grow trees using a greedy top-down recursive partitioning strategy. In order to explain the general training process of decision trees on numeric attributes, let's represent the training data sets as in (3).

$$X = \{(x_i, y_i),\ i = 1, .., M.\ s.t.\ x_i \in R^d,\ y_i \in \{C_1, \dots, C_L\}\} \tag{3}$$

With

X	Training set,
x_i	A vector that represents an example or object in X,
y_i	The category or class of x_i,
M	Number of examples in training set,
d	Number of features,
L	Number of classes.

The general methodology to build a decision tree is as follows: beginning from root node (it contains X), split the data into two or more smaller disjoint subsets, each subset should contain all or most of its elements with the same class, however this is not necessary. Each subset can be considered as a leaf whether certain criterion is satisfied (usually a minimum number of objects within the node or a level of impurity), in this case the partition process is stopped and the node is labeled according majority class in that node, otherwise the process is recursively applied on each subset. A branch or link is created from a node to each one of its partitions.

The data in a node is split computing the attribute that produces the purer partitions. The test to determine best attribute typically takes the form (4). Among all possible values, the general heuristic is to choose the attribute that decreases the impurity as much as possible (Duda, Hart, & Stork, 2000).

There are different impurity measures, most common are information impurity or entropy impurity (5), variance impurity for two class problems (6), gini impurity (7) and miss classification impurity (8).

$$x_{i,j} \leqslant x_{k,j}? \tag{4}$$

with

$x_{i,j}$ The value of jth feature $(0 < j \leqslant d)$ of the ith object in X,

$x_{k,j}$ The value of jth feature of the kth object in $X, k = 1, \dots N, N$ the number of elements within the node.

$$i(Node) = -\sum_{l=1}^{L} P(y = c_l) log_2(P(y = c_l)) \tag{5}$$

with $P(y = c_l)$ the fraction of objects within the node that have class c_l.

$$i(Node) = P(y = c_1)P(y = c_2) \tag{6}$$

$$i(Node) = \sum_{l=1, m=1, m \neq l}^{L} P(y = c_l)P(y = c_m) \tag{7}$$

$$i(Node) = 1 - max_{c_l}\ P(y = c_l) \tag{8}$$

Once a decision tree has been growth, the general methodology suggests to (post) prune the tree to avoid over-fitting.

3.2. Fisher's linear discriminant

Fisher linear discriminant can be considered as a method for linear dimensionality reduction. The method is based on minimizing the projected class overlapping that maximizes the distance between class means while minimizing the variance within each class.

Considering a binary classification problem, the training data set has same structure (3). Let's separate X into two subsets X^+ and X^-.

$$X^+ = \{x_i \in X\ s.t.\ y_i = c_1\} \tag{9}$$

$$X^- = \{x_i \in X\ s.t.\ y_i = c_2\} \tag{10}$$

The means μ^+ and μ^- for X^+ and X^- respectively are computed using (11).

$$\mu^{\pm} = \frac{1}{|X^{\pm}|} \sum_{i=1}^{|X^{\pm}|} x, \quad x \in X^{\pm} \tag{11}$$

With

$X^{\pm}$ Represent X^+ or X^-,

$\mu^{\pm}$ Represent either $\mu^+ \in R^d$ or $\mu^- \in R^d$

Let be $\boldsymbol{\omega} \in R^d$ a vector used to project every example in X. The mean of projections on $\boldsymbol{\omega}$ is given by (12).

$$\mathbf{m}^{\pm} = \frac{1}{|X^{\pm}|} \sum_{i=1}^{|X^{\pm}|} x_i, \quad x \in X^{\pm} \tag{12}$$

With

$x_i = \boldsymbol{\omega}^T x$

$\mathbf{m}^{\pm}$ representing the mean of projections of objects either in X^{+} ($\mathbf{m}^{+}$) or in X^{+} ($\mathbf{m}^{-}$).

Combining (12) and (11) we obtain (13).

$$\mathbf{m}^{\pm} = \frac{1}{|X^{\pm}|}\sum_{i=1}^{|X^{\pm}|} p_i = \frac{1}{|X^{\pm}|}\sum_{i=1}^{|X^{\pm}|}\omega^T x = \omega^T \frac{1}{|X^{\pm}|}\sum_{i=1}^{|X^{\pm}|} x = \omega^T \mu^{\pm} \tag{13}$$

Fisher linear discriminant method search for a vector $\boldsymbol{\omega}$ that maximizes the separation between means $\mathbf{m}^{+}$ and $\mathbf{m}^{-}$ and at the same time that minimizes the scattering of subsets X^{+} and X^{-}.

The distance between $\mathbf{m}^{+}$ and $\mathbf{m}^{-}$ is

$$|\mathbf{m}^{+} - \mathbf{m}^{-}| = |\omega^T \mu^{+} - \omega^T \mu^{-}| \tag{14}$$

In order to measure the scattering of $X^{\pm}$ the scatter for projected objects is defined as in (15), which is called the within-class variance (Duda et al., 2000; Bishop, 2006).

$$\tilde{s}^2_{\pm} = \frac{1}{|X^{\pm}|}\sum_{i=1}^{|X^{\pm}|}(y_i - \mu_{\pm}) \tag{15}$$

The optimization problem becomes (16)

$$\max_{\omega} J(\omega) = \frac{|\mathbf{m}^{+} - \mathbf{m}^{-}|^2}{\tilde{s}^2_{+} + \tilde{s}^2_{-}} \tag{16}$$

The denominator in (16) is known as total within-class variance. A more useful form of (16) is given in (17)

$$\max_{\omega} J(\omega) = \frac{\omega^T S_B \omega}{\omega^T S_W \omega} \tag{17}$$

With

S_B=$(\mu_{-} - \mu_{+})(\mu_{-} - \mu_{+})^T$ called the between-class covariance matrix.

$S_W = \sum_{x_i \in X^{+}}(x_i - \mu_{+})(x_i - \mu_{+})^T + \sum_{x_j \in X^{-}}(x_j - \mu_{-})(x_j - \mu_{-})^T$ called the total within-class covariance matrix.

Solution of (17) is (18), which computation has complexity $O(d^2|X|)$.

$$\omega = S_W^{-1}(\mu_{+} - \mu_{-}) \tag{18}$$

A problem with Fisher linear discriminant occurs when data distribution is multi-modal and when there exist overlapping between classes, under these situations the vector ω is not enough to clearly discriminate between classes (Li, 2005), this can be though as a weak learning algorithm (Schapire, 1990). Based on the recursive strategy of decision tree, we propose to come up with a stronger classifier.

4. Proposed method

The Fisher's decision tree is a two class classifier that takes advantage of dimensionality reduction of Fisher's linear discriminant, which is able of perfectly classify objects that belong to data sets that are linearly separable. Because most real world data sets are not linearly separable, the method follows a decision tree's philosophy splitting the data, recursively. The splits are produced using only one artificial attribute.

The projections of objects x_i on the artificial attribute ω eq. (see Eq. (19)) are used to create the possible splitting points, each split point is between two consecutive projections.

The splitting points p_{test} Eq. (20) are used to create two partitions and then an impurity measure is used. The number of tests that must be realized at each node of the tree is $N - 1$. Each p_{test} creates a split which separates the data similar to a linear decision boundary, this is completely determined: it is orthogonal to vector ω and passes through the point Eq. (20).

$$p_i = \frac{\omega^T x_i}{\|\omega\|}\omega, \; i = 1, \ldots, N - 1 \tag{19}$$

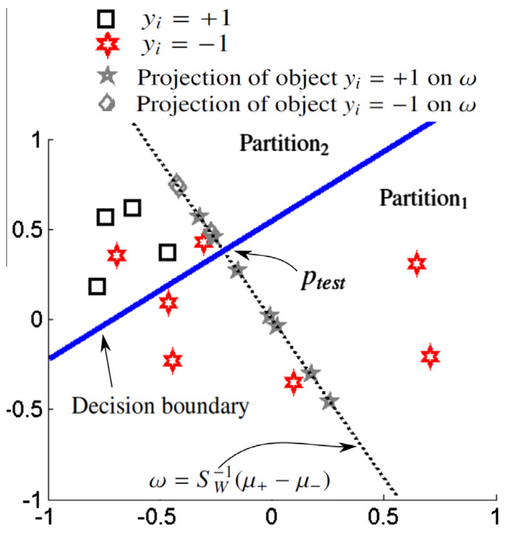
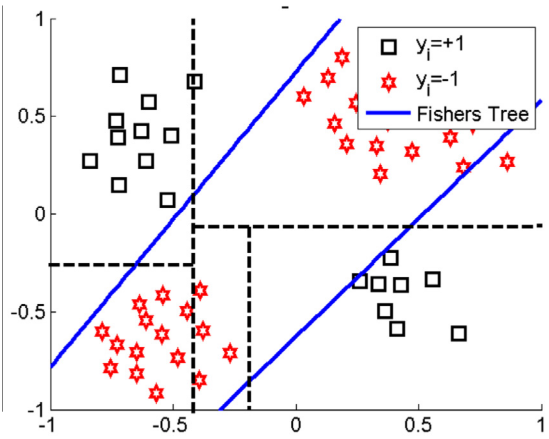
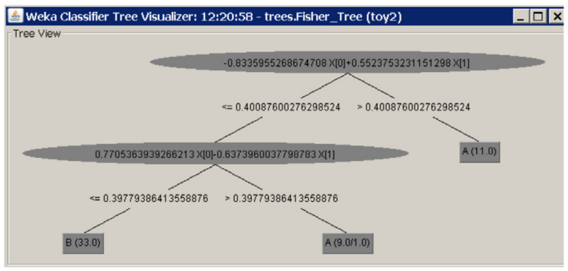


Fig. 1. Splitting a set of points using the artificial attribute.



(a) Decision boundaries found by Fisher's tree (solid line) and C4.5 (dashed line)



Weka Classifier Tree Visualizer: 12:20:58 - trees.Fisher_Tree (toy2)
Tree View
-0.8335955268674708 X[0]+0.5523753231151298 X[1]
<= 0.40087600276298524
> 0.40087600276298524
0.7705363939266213 X[0]-0.6373960037798783 X[1]
A (11.0)
<= 0.39779386413558876
> 0.39779386413558876
B (33.0)
A (9.0/1.0)

(b) The induced tree

Fig. 2. Fisher's decision tree applied on a toy example.

with

p_i the ith projection of object x_i on ω.

The Fig. 1 shows an example of the underlying idea of the splitting: All objects in a toy data set of Fig. 1 have been projected on the optimal vector ω, the splitting point shown in this figure creates two partitions, one of them (Partition$_1$) is pure whereas the other partition (Partition$_2$) contains mixed objects. Impure partitions can be recursively analyzed up to a criterion is satisfied. The current implementation of Fisher's decision tree uses the Information gain (see Eq. (5)) criterion, however it is easy to implement the Gain ratio, Gini index or others. The maximum number of possible partitions is two in the current implementation of the classifier.

$$p_{test} = \frac{\omega^T(p_i + p_{i+1})}{2\|\omega\|}\omega \tag{20}$$

The Fisher's tree growing method is shown in Algorithm 1, it resembles to traditional decision trees, however the main difference is in how the splits are created; instead of testing every attribute only the artificial attribute is used. Although it seems that Fisher's tree suffers from repetition, which is the phenomenon that occurs when an attribute is repeatedly tested along a given branch of the tree, this is different because there is only one attribute to split.

Algorithm 1: Fisher's decision tree growing

```
Data:
S: a set of objects
$Min_{obj}$: Minimum number of objects per leaf
Output:
T: a decision tree
Begin algorithm
Create a node N using S
if (all elements in N are in the same class C)
  return N as leaf node, assign it class C;
else if (N contains less than $Min_{obj}$ objects)
  return N as leaf node, assign it to majority class C;
else
Build $X^+$ and $X^-$ according to (9) and (10).
Compute artificial attribute $\omega$ using (18)
Compute best split point (Algorithm 2)
Split N according to best split point Eq. (21)
for each partition do
    Grow a subtree on partition (a son of N)
end for
return Fisher's decision tree
End algorithm
```

Algorithm 2: Compute split point

```
Data:
S: a set of objects
$\omega$: Artificial attribute
Output:
$p_{testOpt}$: The best split point (according to some criterion)
Begin algorithm
Project every object in S on $\omega$
Sort S in ascending order w.r.t projection
//Choose the projection that produce the best split
$p_{testOpt} \leftarrow$ worst possible impurity value//e.g. $-\infty$
for each $p_{test}$ (20) do
    Compute impurities of partitions, use (5),
(6) or (7)
    if $p_{test}$ is better than $p_{testOpt}$then
//Update best split point (save the index and real
  value)
        $p_{testOpt} \leftarrow p_{test}$
    end if
end for
return $p_{testOpt}$
End algorithm
```

The partitions are created according to Eq. (21).

$$\begin{aligned} Partition_1 &= \{(x_i, y_i) \in N, s.t.\ p_i <= p_{test}\} \\ Partition_2 &= \{(x_i, y_i) \in N, s.t.\ p_i > p_{test}\} \end{aligned} \tag{21}$$

In order to efficiently split N in Algorithm 1, the Algorithm 2 actually returns the index of the best split point and the real value (Eq. (22)) which is used in the prediction phase.

$$p_{testOpt} = \frac{\omega^T(p_i + p_{i+1})}{2\|\omega\|} \tag{22}$$

When a node can not be split any more because it is pure or there are not enough objects, the node becomes a leaf of the tree. The leaf is assigned to the class having the highest probability.

The Fisher's decision tree avoids to model anomalies in the training data due to noise or outliers by pruning the tree. Among several options such as pre pruning, the use of training set to estimate error rates and pessimistic post pruning, Fisher's decision tree uses the last. We adopt an implementation similar to C4.5.

The Fig. 2 shows an example of an induced decision tree from a toy data set using the proposed method. The Fig. 2(a) shows the decision boundaries and the Fig. 2(b) shows the tree structure. The boundaries discovered by the Fisher's decision tree are less complicated than the obtained with a univariate decision tree, the size of the tree is also smaller.

Once the tree has been induced each non terminal node contains the vector ω and the optimal split point, the label of unseen examples is predicted by following down the tree down to a leaf as follows:

(1) If current node is a leaf then the label of the example is the class associated to the leaf, stop.
(2) Otherwise, project the new example on ω, and go to left or right node according to the following rule
    (a) If projection is greater than split point go to right node,
    (b) otherwise go to left node,
(3) recursively repeat this process.

The proposed method involves the following steps to induce the tree: (a) Separate in positive and negative class, (b) compute vector ω, (c) compute split point and split a set of examples.

At each non terminal node, the step (a) runs in $O(|S|)$ time with $|S|$ the size of set being partitioned. The step (b) is the most costly. The vector ω is computed in $O(d^2|S|)$ time, with d the number of dimensions. The step (c) requires resorting the data at each recursive call, which adds $O(S\,log(|S|))$ time. An advantage of our method with respect to other decision trees is that it is not necessary to use any discretization either test every attribute.

5. Experiments and discussion

We study the performance of the Fisher's decision tree using 12 public available data sets.

Two experiments were conduced. In the first experiment, we explore how the performance of our classifier is affected by changing the parameters. In the second experiment we compare our method against C4.5, a well known univariate decision tree The accuracy, size and training time of produced trees is presented.

5.1. Data sets

In order to test the effectiveness of the proposed method, it was tested on data sets commonly used for classification, the Table 1 shows the main characteristics of them. Because some of the data

Table 1
Data sets for experiments.

Data set	Size	Dim	Class 1	Class 2
Iris-setosa	100	4	50	50
Iris-versicolor	100	4	50	50
Iris-virginica	100	4	50	50
Wine-1	119	13	71	48
Wine-2	107	13	59	48
Wine-3	130	13	59	71
Heart	270	13	120	150
Ionosphere	351	34	126	225
Breast cancer	683	10	444	239
Diabetes	768	8	500	268
Svmguide3	1243	22	296	947
Waveform-0	3308	40	1653	1655
Waveform-1	3347	40	1692	1655
Waveform-2	3347	40	1692	1653
Ijcnn1	35,000	22	3415	31,585
Bank full	45,211	16	39,922	5289
Cod rna	59,535	8	19,845	36,690

sets are for multi-class problems, we created binary versions of them by retaining two classes, the class that was removed from original data set is indicated as -class, for example in Table 1 the data set *Iris-setosa* means that class setosa has been removed from *Iris* data set. A total number of 18 training sets are used in the experiments.[2]

5.2. Experiments setup

The Fisher's decision tree was implemented on Java as programming language and Weka as base platform.

All the experiments were run on a Laptop with Intel core *i*7 2670*QM* CPU at 2.2 GHz 8 GB RAM, Windows 7 Operating System installed. We used 100 fold cross validation in order to validate the results. The amount of memory available for the Java Virtual Machine is 500 MB.

5.3. Experiment 1: effect of parameters

The main parameters of Fisher's decision tree are the minimum number of objects in leaves (Min_{obj}), the confidence factor, subtree raising, pruned and MDL correction. Among these parameters the Min_{obj} one affects considerably the size of the tree and the accuracy.

It can be seen in the Fig. 3 that for the svmguide3 data set that in general the accuracy of our Fisher's decision tree outperforms the accuracy of C4.5 tree. The number of leaves and the size of the induced tree using our method is significantly smaller than the produced with the C4.5 algorithm when the number of objects in the leaves in lesser than 50. For more than 50 objects in each leaf, the algorithms converge to the same size and number of leaves.

A similar behavior was observed with the other data sets, but svmguide3 set was chosen to be presented because both C4.5 and Fisher's decision tree obtain the lowest accuracy.

5.4. Experiment 2: performance with respect to number of dimensions and size of data set

In order to observe the performance of our method with respect to the number of dimensions and the size of training set, we created a synthetic linearly inseparable data set. This data set consists of groups of examples that form hyper spheres containing all its examples with the same class, we called it "Spheres". The number

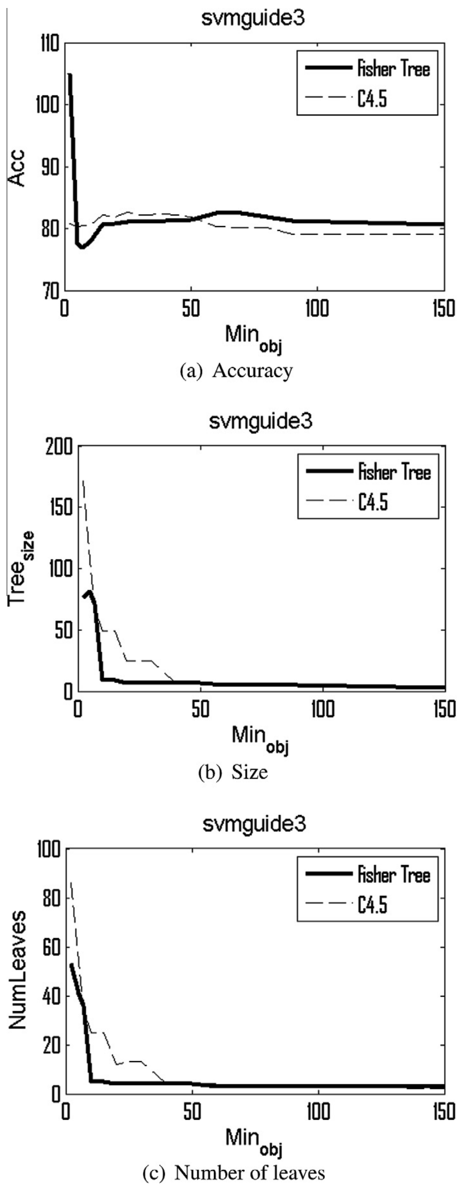


Fig. 3. Effect of parameter Min_{obj} on the induced tree for svmguide3 data set.

of dimensions and the number of examples are after the name. The Fig. 4 shows the Spheres3D1K data set, it has three features (dimensions) and 1000 examples. The blue squares correspond to examples with positive label while the red circles correspond to examples with negative label.

We first test how our method behaves with the number of features. The Table 2 shows the results. With a few number of features, the Fisher' Decision Tree is slower than C4.5. This can be explained because for our method the computation of artificial attribute (vector ω) is $O(d^2|X|)$ and in addition it is necessary to search the split point which takes almost linear time. The C4.5 tree only searches in a small number of features and finds the best split point also in linear time. As the number of features becomes higher, our method becomes more efficient than C4.5. This is because the later needs to search the split point in every feature for each point, which is a time consuming task. For the Fisher's Decision Tree the training time is only slightly increased due to the computation of artificial attribute, however our method does not need to search the split point in every attribute.

[2] Data sets are available at http://www.mediafire.com/?43u8tl9h21y1c7q.

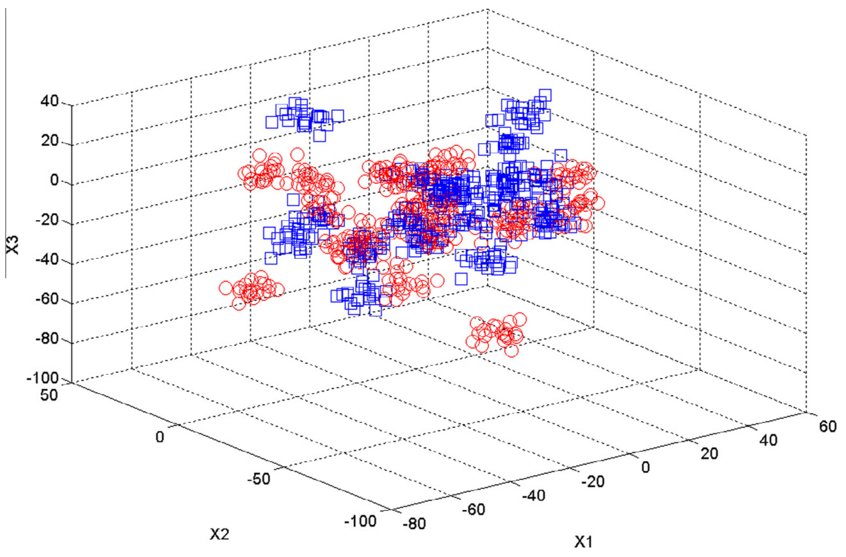


Fig. 4. Example of synthetic data set in three dimensions.

Table 2
Performance of the method with respect to number of features.

Data set/Method	*Min*$_{obj}$	*Tree*$_{size}$	Time (ms)	Acc (%)
Spheres2D40K				
Fisher's Tree	2	61	1186	90.76
C4.5	2	159	690	91.07
Spheres3D40K				
Fisher's Tree	2	351	1790	97.95
C4.5	2	283	900	98.61
Spheres4D40K				
Fisher's Tree	2	343	1360	98.15
C4.5	2	301	1100	98.42
Spheres6D40K				
Fisher's Tree	2	377	1810	98.85
C4.5	2	253	2130	99.49
Spheres10D40K				
Fisher's Tree	2	27	1005	99.96
C4.5	2	47	1720	99.94
Spheres15D40K				
Fisher's Tree	2	75	1260	99.94
C4.5	2	91	3270	99.85
Spheres20D40K				
Fisher's Tree	2	49	1230	99.93
C4.5	2	83	5200	99.89
Spheres40D40K				
Fisher's Tree	2	15	1790	99.96
C4.5	2	85	13,030	99.91

Table 3
Performance of the method with respect to size.

Data set/Method	*Min*$_{obj}$	*Tree*$_{size}$	Time (ms)	Acc (%)
Spheres10D1K				
Fisher's Tree	2	11	10	99.30
Fisher's Tree	20	6	10	98.40
C4.5	2	17	50	98.60
C4.5	20	17	50	93.80
Spheres10D2K				
Fisher's Tree	2	29	30	99.10
Fisher's Tree	20	11	30	97.70
C4.5	2	27	80	99.20
C4.5	20	13	30	97.15
Spheres10D10K K				
Fisher's Tree	2	67	200	99.69
Fisher's Tree	2	31	120	99.14
C4.5	2	37	470	99.73
C4.5	2	25	330	99.30
Spheres10D40K K				
Fisher's Tree	2	27	1005	99.96
Fisher's Tree	20	19	900	99.93
C4.5	2	47	1720	99.94
C4.5	20	23	1450	99.90
Spheres10D80K K				
Fisher's Tree	2	99	4650	99.94
Fisher's Tree	20	57	4140	99.82
C4.5	2	201	6790	99.85
C4.5	20	105	4910	99.68

Now we explore how the proposed method behaves with respect to the size of training data set. We chose the Sphere10D and we vary the number of examples from 1000 up to 80,000. The Table 3 shows the training times and accuracy for several sizes of the synthetic data set. It can be observed that in general the Fihser's Decision Tree produces more accurate results and the training time is better than the other method.

We can say that our method scales better than C4.5, with respect to the number of attributes. Our method scales similar to the other method with respect to the size of training set.

5.5. Experiment 3: accuracy, tree size and training time

For the third experiment, we fixed the parameters of Fisher's decision tree. The values used are shown in the Table 4.

Table 4
Values of parameters used in experiment 3.

Parameter	Fisher's Tree	C4.5
Binary splits	N/A	true
Collapse Tree	true	true
Confidence factor	0.25	0.25
Num Folds	NA	3
Subtree raising	true	true
Pruned	true	true
Use Laplace	false	false
use MDL Correction	false	false

The Fisher's decision tree is compared with C4.5 classifier, the former produces oblique trees and the last one produces parallel-axis partitions.

Table 5
Results for real world data sets (a).

Data set/Method	Min_{obj}	$Tree_{size}$	Time (ms)	Acc (%)
Iris-setosa				
Fisher's Tree	2	3	30	96
C4.5	2	7	20	93
Iris-versicolor				
Fisher's Tree	2	3	1	100
C4.5	2	3	1	100
Iris-virginica				
Fisher's Tree	2	3	1	100
C4.5	2	3	1	99
wine-1				
Fisher's Tree	2	3	1	98.32
C4.5	2	5	1	92.44
wine-2				
Fisher's Tree	2	3	1	100
C4.5	2	3	1	99.07
wine-3				
Fisher's Tree	2	3	20	99.23
C4.5	2	9	20	94.62
Heart				
Fisher's Tree	2	33	50	75.56
C4.5	2	47	20	71.48
Ionosphere				
Fisher's Tree	2	15	50	88.604
C4.5	2	35	60	91.45
Breast cancer				
Fisher's Tree	2	15	50	95.90
C4.5	2	25	60	96.05
Breast cancer				
Fisher's Tree	2	15	50	95.90
C4.5	2	25	60	96.05

Table 6
Results for real world data sets (b).

Data set/Method	Min_{obj}	$Tree_{size}$	Time (ms)	Acc (%)
Diabetes				
Fisher's Tree	2	133	140	70.96
Fisher's Tree	10	37	90	74.09
C4.5	2	141	20	71.75
svmguide3				
Fisher's Tree	2	105	200	76.11
Fisher's Tree	15	9	160	80.61
C4.5	2	119	30	80.69
Waveform-0				
Fisher's Tree	2	55	230	92.92
C4.5	2	189	380	89.72
Waveform-1				
Fisher's Tree	2	79	340	88.52
C4.5	2	199	390	86.71
Waveform-2				
Fisher's Tree	2	79	200	89.08
C4.5	2	263	440	86.37
ijcnn1				
Fisher's Tree	2	849	3600	94.65
C4.5	2	797	12,870	96.96
Bank full				
Fisher's Tree	2	439	3990	88.87
C4.5	2	2671	8300	89.31
cod rna				
Fisher's Tree	2	793	3550	94.26
C4.5	2	2273	19,640	94.71

The performance comparative between the methods can be seen in Tables 5 and 6 which show the results obtained with the Fisher's decision tree and C4.5 classifier.

The accuracy of Fisher's decision tree is comparable with C4.5. For some training sets, our classifier is trained in lesser time than C4.5. The number of attributes of training set seems to have a negligible effect on the training time, e.g., *Diabetes* and *cod rna* sets have both 8 attributes, in the former the C4.5 is faster but in the second set it is not. This happens because oblique partitions are more appropriate for some training sets and parallel-axis partitions for other sets.

In general both classifiers achieve similar accuracies, the training times are smaller for our classifier specially with large data sets, however the size of the Fisher's tree is smaller in practically all cases.

The minimum number of objects per partition (Min_{obj}) affects the performance of the induced tree. The default value of $Min_{obj} = 2$ is not always a good choice, increasing this number produces a smaller number of partitions.

6. Conclusions

In this paper we introduced a novel method to induce oblique trees called the Fisher's Decision Tree. Our method uses an artificial attribute created with the vector computed by the Fisher's linear analysis, and the objects in training data set are projected on it. This creates a unique artificial attribute to split the data set. The Fisher's decision tree method was compared against C4.5, one of the most effective induction tree algorithms using public available data sets.

We explored the performance of the proposed method through a number of experiments with synthetic and real world data sets. In general the accuracy obtained with our method is not worst than the obtained with a univariate decision tree, however the size of the tree and the training time are both better.

The Fisher's Decision Tree is efficient when the number of attributes and the size of data set are both high, this is because it only searches for the optimal split point on the artificial attribute, other methods need to search in every attribute. We provide a complete implementation of our method in the Java language and the source code is also available. Currently we are studying how to improve the performance using Graphical Processing Units.

References

Barros, R. C., Basgalupp, M. P., de Carvalho, A. C. P. L. F., & Freitas, A. A. (2011). A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* (99), 1–10.

Bishop, C. M. (2006). *Pattern recognition and machine learning (Information science and statistics)*. Secaucus, NJ, USA: Springer -Verlag New York, Inc..

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Wadsworth.

Cantu-Paz, E., & Kamath, C. (2003). Inducing oblique decision trees with evolutionary algorithms. *Transactions on Evolutionary Computation, 7*(1), 54–68<http://dx.doi.org/10.1109/TEVC.2002.806857>.

Cios, K.J., Pedrycz, W., Swiniarski, R.W., & Kurgan, L.A. (2007). Data mining: a knowledge discovery approach (pp. 391–393). Springer.

Cruz-Ramírez, N., Acosta-Mesa, H.-G., Carrillo-Calvet, H., & Barrientos-Martínez, R.-E. (2009). Discovering interobserver variability in the cytodiagnosis of breast cancer using decision trees and bayesian networks. *Applied Soft Computing, 9*(4), 1331–1342.

Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern classification* (2nd ed.). Wiley-Interscience.

Fan, C.-Y., Chang, P.-C., Lin, J.-J., & Hsieh, J. (2011). A hybrid model combining case-based reasoning and fuzzy decision tree for medical data classification. *Applied Soft Computing, 11*(1), 632–644.

Gama, J. (1997). Oblique linear tree. In *Proceedings of the 2nd international symposium on advances in intelligent data analysis. Reasoning about Data IDA '97* (pp. 187–198). London, UK: Springer-Verlag.

Gama, J. (1999). Discriminant trees. In *Proceedings of the 16th international conference on machine learning. ICML '99* (pp. 134–142). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc..

He, P., Xu, X.-H. & Chen, L. (2008). Tree classifier in singular vertor space. In *2008 International Conference on Machine Learning and Cybernetics* (Vol. 3, pp. 1801–1806).

Henrichon, E. G., & Fu, K.-S. (1969). A nonparametric partitioning procedure for pattern classification. *IEEE Transactions on Computers, 18*(7), 614–624.

Iyengar, V. (1999). Hot: Heuristics for oblique trees. In *Proceedings of the 11th IEEE international conference on tools with artificial intelligence* (pp. 91–98).

John, G. H. (1994). *Robust linear discriminant trees. AI& Statistics-95* (Vol. 7). Springer-Verlag [pp. 285–291].

Li, X.-B. (2005). A scalable decision tree system and its application in pattern recognition and intrusion detection. *Decision Support Systems, 41*(1), 112–130.

Manwani, N., & Sastry, P. (2009). A geometric algorithm for learning oblique decision trees. In *Pattern recognition and machine intelligence*. In S. Chaudhury, S. Mitra, C. Murthy, P. Sastry, & S. Pal (Eds.). *Lecture notes in computer science* (Vol. 5909, pp. 25–31). Berlin Heidelberg: Springer.

Menkovski, V., Christou, I. & Efremidis, S. (2008). Oblique decision trees using embedded support vector machines in classifier ensembles. In *7th IEEE international conference on cybernetic intelligent systems CIS 2008* (pp. 1–6).

Murthy, S. K., Kasif, S., & Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research, 2*(1), 1–32.

Pomorski, D., & Perche, P. (2001). Inductive learning of decision trees: Application to fault isolation of an induction motor. *Engineering Applications of Artificial Intelligence, 14*(2), 155–166.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5*(2), 197–227<http://dx.doi.org/10.1023/A:1022648800760>.

Setiono, R., & Liu, H. (1999). A connectionist approach to generating oblique decision trees. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 29*(3), 440–444.

Shah, S., & Sastry, P. (1999). New algorithms for learning and pruning oblique decision trees. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 29*(4), 494–505.

Shali, A., Kangavari, M. & Bina, B. (2007). Using genetic programming for the induction of oblique decision trees. In *6th International conference on machine learning and applications ICMLA 2007* (pp. 38–43).

Utgoff, P.E. & Brodley, C.E. (1991). Linear machine decision trees. Tech. rep., Amherst, MA, USA.

Vadera, S. (2005). Inducing safer oblique trees without costs. *Expert Systems, 22*(4), 206–221.

Vadera, S. (2010). Csnl: A cost-sensitive non-linear decision tree algorithm. *ACM Transactions on Knowledge Discovery from Data, 4*(2), 6:1–6:25<http://doi.acm.org/10.1145/1754428.1754429>.

Wu, M.-C., Lin, S.-Y., & Lin, C.-H. (2006). An effective application of decision tree to stock trading. *Expert Systems with Applications, 31*(2), 270–274.

Xiaohu, W., Lele, W., & Nianfeng, L. (2012). An application of decision tree based on id3. *Physics Procedia, 25*, 1017–1021.

Yildiz, O. T., & Alpaydin, E. (2000). Linear discriminant trees. In *Proceedings of the 17th international conference on machine learning. ICML '00* (pp. 1175–1182). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.<http://dl.acm.org/citation.cfm?id=645529.657979>.