Graduate Research in Engineering and Technology (GRET)

Volume 1 | Issue 1 Article 4

July 2013

LEARNING HYPERPLANES THAT CAPTURES THE GEOMETRIC STRUCTURE OF CLASS REGIONS

PRAMOD PATIL
Dept. of Computer Engineering, College of Engineering, Pune, pdpatiljune@gmail.com

ALKA LONDHE
PG, Student (ME), Dept. of Computer Engineering, DYPIET, Pune, alka_londhe@rediffmail.com

PARAG KULKARNI
Dept. of Computer Engineering, College of Engineering, Pune, parag.india@gmail.com

LEARNING HYPERPLANES THAT CAPTURES THE GEOMETRIC STRUCTURE OF CLASS REGIONS

PRAMOD PATIL[1], ALKA LONDHE[2] & PARAG KULKARNI[3]

[1,3]Dept. of Computer Engineering, College of Engineering, Pune
[2]PG, Student (ME), Dept. of Computer Engineering, DYPIET, Pune
E-mail: pdpatiljune@gmail.com, alka_londhe@rediffmail.com, parag.india@gmail.com

Abstract- Most of the decision tree algorithms rely on impurity measures to evaluate the goodness of hyperplanes at each node while learning a decision tree in a top-down fashion. These impurity measures are not differentiable with relation to the hyperplane parameters. Therefore the algorithms for decision tree learning using impurity measures need to use some search techniques for finding the best hyperplane at every node. These impurity measures don't properly capture the geometric structures of the data. In this paper a Two-Class algorithm for learning oblique decision trees is proposed. Aggravated by this, the algorithm uses a strategy, to evaluate the hyperplanes in such a way that the (linear) geometric structure in the data is taken into consideration. At each node of the decision tree, algorithm finds the clustering hyperplanes for both the classes. The clustering hyperplanes are obtained by solving the generalized Eigen-value problem. Then the data is splitted based on angle bisector and recursively learn the left and right sub-trees of the node. Since, in general, there will be two angle bisectors; one is selected which is better based on an impurity measure gini index. Thus the algorithm combines the ideas of linear tendencies in data and purity of nodes to find better decision trees. This idea leads to small decision trees and better performance.

Keywords- *Oblique decision tree; generalized eigenvalue; optimization proble; clustering hyperplanes*

I. INTRODUCTION

Decision tree is a well known and widely used method for classification. The popularity of decision tree is because of its simplicity and easy interpretability as a classification rule. In a decision tree classifier, every non-leaf node is associated with a so called split rule or a decision function which is a function of the attribute vector and is commonly binary-valued. Each leaf node in the tree is associated with a class label. To classify an attribute vector using a decision tree, at every non-leaf node that is encountered a branch is taken to one of the children of that node based on the value assumed by the split rule of that node on the given attribute vector. This process follows a path in the tree and when a leaf is reached, the class label of the leaf is assigned to that attribute vector. Decision trees can be broadly classified into two types [1]:

- Axis – Parallel Decision Trees
- Oblique Decision Trees

In an **axis parallel decision tree**, the split rule at every node is a function of only one of the components of the attribute vector. For example, at each node a test may do whether a chosen component of the attribute vector belongs to some interval. Axis-parallel decision trees are mostly attractive when all features are nominal (i.e., take only finitely many values); in such cases we can have a non-binary tree where at each node we test one feature value and the node can have as many children as the values assumed by that feature. In general, axis-parallel decision trees represent class boundaries by piecewise linear segments where each segment will be a line parallel to one of the feature axes. Axis-parallel decision trees are good when the decision boundaries are actually axis parallel. However, in more general situations, we have to approximate even arbitrary linear segments in the class boundary with many axis-parallel pieces and hence the size of the resulting tree becomes large.

The **oblique decision trees**, conversely, use a decision function that depends on a linear combination of all feature components. Thus an oblique decision tree is a binary tree where the hyperplane is associated (in the feature space) with each node. To classify a pattern, a path is followed in the tree by taking the left or right child at each node based on which side of the hyperplane (of that node) that the attribute vector falls within. Oblique decision trees represent the class margin as a general piecewise linear surface. Oblique decision trees are more adaptable (and hence are more popular) when features are real-valued.

In a decision tree, each hyperplane at a non-leaf node should split the data in such a way that it aids further classification; the hyperplane itself need not be a good classifier at that stage. In view of this, many top down decision tree learning algorithms are based on rating hyperplanes using the so called impurity measures. The core idea is as follows. Given the set of training patterns at a node and a hyperplane, the set of patterns are known that join the left and right child of this node. If each of these two sets of patterns has predominance of one class over others, then, presumably, the hyperplane can be considered to have contributed positively to further classification. At any step in the learning process, the level of purity of a

node is some measure of how skewed is the distribution of different classes within the set of patterns landing at that node. If the class distribution is nearly uniform then the node is highly impure; if number of patterns of one class is much larger than that of all others then the purity of the node is high.

The impurity measures used in the algorithms give higher rating to a hyperplane which results in higher purity of child nodes [4]. Gini index, Entropy and Towing rule are some of the frequently used impurity measures. Many of the impurity measures are not differentiable with respect to the hyperplane parameters. Thus the algorithms for decision tree learning using impurity measures need to use some search techniques for finding the best hyperplane at each node. All top down decision tree algorithms employ greedy search and a hyperplane learned at a node once, is never changed. If a bad hyperplane is learned at some node, it cannot be corrected afterword and the effect of it may be a large sub-tree at that node. To overcome such over-fitting, the learnt decision tree should be pruned.

A problem with all impurity measures is that they depend only on the number of (training) patterns of different classes on either side of the hyperplane. Thus, if the class regions are changed without changing the effective areas of class regions on either side of a hyperplane, the impurity measure of the hyperplane will not change. Thus the impurity measures do not truly capture the geometric structure of class distributions (or the class regions in the feature space). Also, all the algorithms need to optimize on some average of impurity of the child nodes and often it is not clear what kind of average is proper.

In this paper a Two-Class decision tree learning algorithm is presented, which is based on the idea of capturing, to some extent, the geometric structure of the underlying class regions. For this the ideas from some recent variants of the SVM method are used, which are quite good at capturing (linear) geometric structure of the data. The rest of this paper is organized as follows: The Mathematical Model is given in Section 2. Section 3 presents the proposed Algorithm. Experimental setup and Result Analysis are given in Section 4. We conclude this paper in Section 5.

II. MATHEMATICAL MODEL

The presented algorithm for learning oblique decision trees is based on the idea of capturing, to some extent, the (linear) geometric structure of the underlying class regions. Given a set of training patterns at a node, algorithm first finds two clustering hyperplanes, one for each class. Each hyperplane is such that it is closest to all patterns of one class and is farthest from all patterns of the other class. This formulation leads to two generalized eigenvalue problem. These are the solutions of a general optimization problem [2-3]. The mathematical model for the presented system is given here.

Let's consider the problem of classifying m points in the n–dimension-al real space $\mathbb{R}^n$. A word about some notations:

- All vectors will be column vectors unless transposed to a row vector by a prime superscript $'$.
- The scalar product of two vectors x and y within the n–dimensional real space $\mathbb{R}^n$ will be denoted by $x'y$.
- The two–norm of x will be denoted by $\|x\|$.
- For a matrix $A \in \mathbb{R}^{m \times n}$, A_i is the ith row of A which is a row vector in $\mathbb{R}^n$.
- An arbitrary dimension column vector of ones will be denoted by e.
- The arbitrary order identity matrix will be denoted by I.

Given some training data S, a set of m points of the form

$$S = \{(x_i, y_i)\} \; ; \; x_i \in \mathbb{R}^n \; ; \; y_i \in \{-1, 1\} \quad i = 1 \ldots m\}$$

where the y_i is either 1 or −1, indicating the class to which the point xi belongs. Each xi is an n-dimensional real vector.

Let, $A \in \mathbb{R}^{m_1 \times n}$ represent the data set of class -1 and $B \in \mathbb{R}^{m_2 \times n}$ represent the data set of class 1. Then, by referring [3], get

$$G := [A - e]'[A - e] + \delta I$$
$$H := [B - e]'[B - e]$$

By solving generalized eigenvalue equation $Gz = \lambda Hz$, get

$$\widetilde{w_1} := x'\omega^1 - \gamma^1 = 0 \qquad (1)$$

Similarly,

$$L := [B - e]'[B - e] + \delta I$$
$$M := [A - e]'[A - e]$$

By solving generalized eigenvalue equation $Lz = \lambda Mz$, get

$$\widetilde{w_2} := x'\omega^2 - \gamma^2 = 0 \qquad (2)$$

Once clustering hyperplanes are found, the hyperplane that associate with the current node will be one of the angle bisectors of these two hyperplanes.

Let $\widetilde{w_3} := x'\omega^3 - \gamma^3 = 0$ and $\widetilde{w_4} := x'\omega^4 - \gamma^4 = 0$ be the angle bisectors of $x'\omega^1 - \gamma^1 = 0$ and $x'\omega^2 - \gamma^2 = 0$. It is easily shown that these hyperplanes are given by (in the case $\omega^1 \neq \omega^2$)

$$\widetilde{w_3} = \widetilde{w_1} + \widetilde{w_2} \qquad (3)$$

$$\widetilde{w_3} = \widetilde{w_1} - \widetilde{w_2} \quad (4)$$

As mentioned earlier, the angle bisector is chosen which has lower impurity. Here the gini index is used to measure impurity. Let $\widetilde{w_t}$ be a hyperplane which is used for dividing the set of patterns St in two parts S_l^t and S_r^t . Let m_{-1}^{tl} and m_1^{tl} denote the number of patterns of the two classes in the set S_l^t and m_{-1}^{tr} and m_1^{tr} denote the number of patterns of the two classes in the set S_r^t . Then gini index of hyperplane $\widetilde{w_t}$ is given by,

$$Gini(\widetilde{w_t}) = \frac{m^{tl}}{m^t}\left[1 - \left(\frac{m_1^{tl}}{m^{tl}}\right)^2 - \left(\frac{m_{-1}^{tl}}{m^{tl}}\right)^2\right] + \frac{m^{tr}}{m^t}\left[1 - \left(\frac{m_1^{tr}}{m^{tr}}\right)^2 - \left(\frac{m_{-1}^{tr}}{m^{tr}}\right)^2\right] \quad (5)$$

The algorithm chooses $\widetilde{w_3}$ or $\widetilde{w_4}$ to be the split rule for S^t based on which of the two gives lesser value of gini index given by (5).

When the clustering hyperplanes are parallel (that is, when $\omega^1 = \omega^2$) the hyperplane is chosen midway between them as the splitting hyperplane. It is given by $\widetilde{w} = (\omega, \gamma) = (\omega^1, (\gamma^1 + \gamma^2)/2)$.

As is easy to see, in this method, the optimization problem of finding the best hyperplane at each node is solved exactly rather than by relying on a search technique based on local perturbations of the hyperplane parameters. These clustering hyperplanes are obtained by solving the generalized eigenvalue problem. Afterward, to find the hyperplane at the node, it is needed to compare only two hyperplanes based on gini index.

III. PROPOSED ALGORITHM

The performance of any top down decision tree algorithm depends on the measure used to rate different hyperplanes at every node. The accuracy of the learned tree depends on the size of the tree (depth and the number of leaf nodes) which is also a function of how the example set is splitted at every non-leaf node of the decision tree. If the split rule is such that it misses the right geometry of the classification boundary then the resulting decision tree is likely to be of large size. The issue of having a suitable algorithm to find the hyperplane that optimizes the chosen measure or rating function at each node is also vital. For example, for all impurity measures, the optimization is difficult because finding the gradient of the impurity function with respect to the parameters of the hyperplane is impossible. Motivated by these considerations, here a new criterion function is presented to assess suitability of a hyperplane at a node that can capture the geometric structure of the class regions. For the criterion function, the optimization problem can also be solved more easily. The method of finding the best hyperplane at each node, by considering a 2-class problem is first discussed. Given the set of training patterns at a node, algorithm first finds two hyperplanes, one for each class. Each hyperplane is such that it is closest to all patterns of one class and is farthest from all patterns of the other class. These hyperplanes are called as the clustering hyperplanes (for the two classes).

Algorithm: Two Class Oblique Decision Tree

Input :$S = \{(x_i, y_i)\}$ $i = 1 \ldots m$, Max-Depth, Min-Threshold
Output : Pointer to root of the decision tree

```
Begin
        Root:=GrowTree_TwoClass(S);
        Return Root;
End
GrowTree_TwoClass(S^t)
Input   : S^t
Output: Pointer to a subtree

Begin
        Find matrices A and B of the set S^t of
        patterns at node t;;
        Find clustering hyperplanes (solutions of
        optimization problems);
        Computer angle bisectors ;
        Choose the angle bisector which gives
        lesser value of gini index;
        Let w_t  denote the split rule at node t;
        Divide the set S^t. Such that S^tl = {w_t'x <
        0} and S^tr = {w_t'x ≥ 0} ;
        Define Threshold = min(m_1^t, m_-1^t) / m^t for S_l^t and
        S_r^t;
        If (Threshold^L < Min_Threshold) or
        (Max_TreeDepth =TreeDepth) then
                Prune_Tree
                Get a node t_l and make it leaf
        node;
                Allot class label associated to the
                majority class to t_l;
                Make t_l left child of t;
        Else
                t_l = GrowTree_TwoClass(S_l^t);
                Make t_l left child of t;
        End
        If (Threshold^R < Min_Threshold) or
        (Max_TreeDepth =TreeDepth) then
                Prune_Tree
                Get a node t_r and make it leaf
        node;
                Allot class label associated to the
                majority class to t_r;
                Make t_r right child of t;
        Else
                t_r = GrowTree_TwoClass(S_r^t);
                Make t_r right child of t;
        End
        Return t;
End
```

Because of the way they are outlines, these clustering hyperplanes capture the dominant linear tendencies in the examples of each class that are useful for discriminating between the classes. Hence a hyperplane that passes in between them could be good for splitting the feature space. So, the hyperplane is taken that bisects the angle between the clustering hyperplanes as the split rule at this node. Since, normally, there would be two angle bisectors; the bisector is chosen which is better, based on an impurity measure, i.e. the gini index. If the two clustering hyperplanes happen to be parallel to each other, then a hyperplane is taken midway between the two clustering hyperplanes as the split rule.

A complete description of this is provided as **Algorithm-Two Class Oblique Decision Tree**. Algorithm recursively calls the procedure GrowTree_TwoClass(St) which will return a subtree at node t.

At any given node, given the set of patterns St, the two clustering hyperplanes (by solving the generalized eigenvalue problem) are found. Choose one of the two angle bisectors, based on the gini index, as the hyperplane to be associated with this node. This hyperplane is used to split St into two sets, i.e. the ones that go into the left and right child nodes of this node. Then recursively did the same at the two child nodes. The method starts at the root node of the tree where St is the complete set of example patterns.

The recursion stops when the set of patterns at a node are such that the fraction of patterns belonging to the minority class of this set are below a user-specified threshold or the depth of the tree reaches a pre-specified maximum limit.

IV. EXPERIMENTAL SETUP AND RESULT ANALYSIS

In this section the empirical results are presented to show the effectiveness of the proposed algorithm. The performance of algorithm is tested on synthetic and several real datasets.

The performance of algorithm is compared with OC1 [4], CART [1], C4.5 [8] which are among the standard decision tree algorithms at present. The performance is also compared with SVM classifier which is among the best generic classifiers today. The experimental comparisons are presented on one synthetic datasets and six 'real' datasets from UCI ML repository [6].

A. Dataset Description

One synthetic piecewise-linearly-separable dataset is generated in two dimensions which is described below:

2 × 2 checkerboard data set:

2000 points are sampled uniformly from $[-1\ 1] \times [-1\ 1]$. A point is labeled +1 if it is in $([-1\ 0] \times [0\ 1]) \cup ([0\ 1] \times [0\ -1])$; otherwise, it is labeled −1. Out of 2000 sampled points, 990 points are labeled +1, and 1010 points are labeled −1. Now, all the points are rotated by an angle of $\pi/6$ with respect to the first axis in anti-clockwise direction. That is the final training set.

Apart from this 2x2 checkerboard synthetic dataset, the algorithm is also tested on several bench-mark datasets downloaded from UCI ML repository [6]. The datasets available on UCI ML repository has many observations with missing values of some features. For this experiment those observations are chosen for which there are no missing values for any feature.

TABLE-1 DETAIL OF REAL-WORLD DATA SETS USED FROM UCI ML REPOSITORY

Data Set	Dimensions	# Points	Class Distribution
Breast-Cancer	10	683	444, 239
Bupa Liver Disorder	6	345	200, 145
Pima Indian (diabetes)	8	768	500, 268
Heart	13	270	150, 120
Magic	10	19020	12332, 6688
Votes	16	232	124, 108

B. Experimental Setup

The proposed algorithm is implemented in Java. For OC1, the downloadable package available on Internet is used [4]. For CART, C4.5 the weka package available on Internet is used [7]. Weka is a collection of machine learning algorithms for solving real-world data mining problems. It is written in Java and runs on nearly all platforms. The algorithms can either be applied on to a dataset or called from Java code.

To learn SVM classifiers, the libsvm code is used [5]. Libsvm-2.84 uses the one-versus-rest approach for multiclass classification.

Proposed Algorithm has only two user-defined parameters which are Max-Depth, Min-Threshold (the threshold on fraction of points of minority class to decide any node a leaf node). For all experiments ε value is chosen between 0.05 and 0.2. This range of ε appears quite robust on all the datasets.

SVM has two user defined parameters: penalty parameter C and the width parameter σ for Gaussian kernel. Best values for these parameters are found

using 5-fold cross-validation and the results reported are with these parameters. Both OC1 and CART use 90% of the total number of points for training and 10% points for pruning. OC1 needs two more user-defined parameters. These parameters are number of restarts (R) and number of random jumps (J). For experiments it is set R = 20 and J = 5 which are the default values suggested in the package.

C. Results and Discussion

In this section the performance of proposed algorithm for learning oblique decision tree with other approaches on different datasets is discussed. The results provided are based on 10-fold cross validation. Table 2 shows average values of the performance parameters such as accuracy, time taken and the number of leaf nodes and depth of the tree, for each of the algorithms for each of the problems.

TABLE-2: COMPARISON RESULTS BETWEEN PROPOSED ALGORITHM AND OTHER DECISION TREE APPROACHES

Data Set	Method	Accuracy	Time (Sec)	# Leaves	Tree depth
2x2 Checker-board	2-Class	**98.20**	0.031	4.0	2.0
	OC1	97.70	2.730	13.5	8.0
	C4.5	96.70	0.041	35.6	7.0
	CAR	96.65	0.254	41.0	13.
Breast-Cancer	2-Class	**97.07**	0.014	2.0	1.0
	OC1	94.44	1.520	5.4	3.0
	C4.5	95.16	0.016	12.9	5.0
	CAR	94.29	0.070	5.0	3.0
Bupa – Liver Disorder	2-Class	**71.01**	0.014	6.0	5.0
	OC1	70.43	3.190	5.2	2.8
	C4.5	68.13	0.008	27.1	8.0
	CAR	67.82	0.050	3.0	2.0
Pima – Indian	2-Class	**76.30**	0.030	5.0	5.0
	OC1	71.61	3.880	8.2	5.2
	C4.5	73.70	0.017	22.6	9.0
	CAR	74.87	0.148	6.0	5.0
Heart	2-Class	**86.30**	0.014	2.0	1.0
	OC1	73.70	0.077	7.4	3.4
	C4.5	77.41	0.008	18.3	7.0
	CAR	82.59	0.033	6.0	3.0
Magic	2-Class	82.92	0.655	24.0	6.0
	OC1	-	-	-	-
	C4.5	85.06	3.242	320.2	21.
	CAR	**85.23**	13.157	128	12.
Votes	2-Class	**96.98**	0.013	2.0	1.0
	OC1	94.83	0.170	2.0	1.0
	C4.5	96.97	0.003	2.0	1.0
	CAR	96.97	0.017	2.0	1.0

From Table 2 it can be seen that, over all the problems, proposed algorithm performs significantly better than all the other decision tree approaches in accuracy except on Magic and Pima – Indian datasets. On these datasets, the performance of the proposed algorithm is comparable with the best other decision tree method. The average accuracy of proposed algorithm is better than other decision tree approaches on Breast-Cancer dataset, Bupa–Liver Disorder Dataset, Heart Dataset and Votes dataset. Thus, overall, performance of proposed algorithm is better than or comparable to any other decision tree approach in terms of accuracy.

In majority of the cases proposed algorithm generates trees with smaller depth with lesser number of leaves as compared to other decision tree approaches. This supports idea that proposed algorithm better exploits geometric structure of the dataset while generating decision trees. Normally if smaller trees are learnt, the generalization error of the decision tree is likely to be small.

Time wise proposed algorithm is much faster than other decision tree approaches in all cases as can be seen from the results in the table. In all cases, the time taken by proposed algorithm is less by at least a factor of five to ten. This is because the problem of obtaining the best split rule at each node is solved using an efficient linear algebra algorithm in case of proposed algorithm while the other approaches have to resort to search techniques because optimizing impurity-based measures is tough.

TABLE 3: COMPARISON RESULTS OF PROPOSED ALGORITHM WITH SVM

Data Set	Method	Accuracy	Time (Sec)	Kernel
2x2 checkerboard	2-Class	**98.20**	0.031	-
	SVM	97.35	0.218	Gauss ian
Breast-Cancer	2-Class	**97.07**	0.014	-
	SVM	65.59	0.159	Gauss ian
Bupa Liver Disorder	2-Class	**71.01**	0.014	-
	SVM	59.74	0.040	Gauss ian
Pima Indian	2-Class	**75.39**	0.030	-
	SVM	65.11	0.169	Gauss ian
Heart	2-Class	**86.30**	0.014	-
	SVM	55.93	0.042	Gauss ian
Magic	2-Class	**82.92**	0.655	-
	SVM	65.87	285.139	Gauss ian
Votes	2-Class	**96.98**	0.013	-
	SVM	96.97	0.0232	Gauss ian

Now next consider comparisons of proposed algorithm with SVM. Table 3 shows comparison results of proposed algorithm with SVM.

The performance of proposed algorithm is comparable to that of SVM in terms of accuracy. Proposed algorithm performs comparable to SVM on 2×2-checkerboard dataset, Breast Cancer Dataset, Bupa–Liver dataset, Pima Indian dataset, Magic dataset, Heart dataset and Votes dataset.

In terms of the time taken to learn the classifier, proposed algorithm is faster than SVM on all cases. At every node of the tree, a generalized Eigen-value problem is solved which takes time of the order of (d +1)3, where d is the dimension of the feature space. On the other hand, SVM solves a quadratic program whose time complexity is $O(n^k)$, where k is between 2 and 3 and n is the number of points. So in general, when number of points is large compare to the dimension of the feature space, proposed algorithm learns faster than SVM. Thus, overall, proposed algorithm performs better than SVM.

V. CONCLUSION

In this paper a Two-Class algorithm for learning oblique decision trees is presented. The novelty is in learning hyperplanes (at each node in the top-down induction of a decision tree) that captures the geometric structure of the class regions. At each node of the decision tree, algorithm finds the clustering hyperplanes for both the classes and chooses one of the angle bisector as the split rule. Classification accuracy results of proposed algorithm are comparable to those of classical decision tree induction algorithms and support vector classification algorithms and, in some cases, they are better. As well, the method performs better than the other classic decision tree approaches in terms of size of the tree and time. The simple program formulation, computational efficiency, and accuracy of proposed algorithm on real world data indicate that it is an effective algorithm for classification. Analysis of the statistical properties of proposed algorithm and extensions to Multi–Class classification are promising areas of future research.

REFERENCES

[1] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. “Classification and Regression Trees.” Statistics / Probability Series. Wadsworth and Brooks, Belmont, California, U.S.A., 1984.

[2] Naresh Manwani, P. S. Sastry, Senior Member, IEEE, “Geometric Decision Tree”, Sept. 2010.

[3] O. L. Mangasarian and E.W.Wild, “Multisurface proximal support vector machine classification via generalized eigenvalues,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 28, no. 1, pp. 69–74, Jan. 2006.

[4] Sreerama K. Murthy, Simon Kasif, Steven Salzberg, “A System for Induction of Oblique Decision Trees”, Jmynal of Artificial Intelligence Research2 (1994)

[5] Chih-Chung Chang and Chih-Jen Lin. LIBSVM : a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[6] D.J. Newman A. Asuncion. ”UCI Machine Learning Repository”. University of California, Irvine, School of Information and Computer Sciences, 2007. http://www.ics.uci.edu/~mlearn/ MLRepository.html.

[7] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1. http://www.cs.waikato.ac.nz/ml/weka/

[8] Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993