International Journal of Digital Information and Wireless Communications (IJDIWC) 5(3): 203-209
The Society of Digital Information and Wireless Communications, 2015 (ISSN: 2225-658X)

Decision Trees Algorithms and Classification with Incremental Neural Network

John Tsiligaridis
Math and Computer Science Dept, Heritage University
Toppenish, WA, USA
tsiligaridis_j@heritage.edu

ABSTRACT

The purpose of this paper is to present a set of algorithms for rule extraction using direct and indirect methods and the use of Neural Network for classification. A direct method with simplification (DMS) to extract rules from data using a hash table is also developed. Apart from the rules, DMS can also produce the corresponding decision tree for each subset with predefined class value. The merge of the paths of all call values can produce the decision tree of the data. Therefore, DMS works as both direct and indirect method.
For the indirect method a Decision Tree algorithm (DTA) is created from data using probabilities, aiming at creating on-demand an accurate decision tree (DT) from either data or a stable set of rules.
The advantages of both of them to avoid redundant branches are presented. DMS is the ability to produce rules with a maximum number of conditions and prevention of subtree overlapping. Moreover, an incremental Backpropagation Neural Network (IBNN) is created using the instances of data grouping according to their predefined class. Comparisons show that IBNN outperforms DMS. Simulation results are provided.

KEYWORDS

Data Mining, Neural Networks, Rule Extraction, In cremental Neural Network, Backpropagation.

1 INTRODUCTION

Decision Trees are one of the most popular techniques of data mining [3],[4],[5].
In decision trees the input data set has one attribute called class C that takes a value from K discrete values 1,..,K and a set of numeric and categorical attributes A1,..,Ap. [8].
Numerous techniques [1],[2] have been developed for the decision tree learning including parallelization and data partitioning.
There are two types of DTs: the complete and the incomplete ones. In the incomplete type, there are subtrees where the repetition and replication are included [3]. Repetition is where an attribute is repeatedly tested along a given branch of three , e.i. age, and replication where duplicate subtrees exist within a tree , such as the subtree headed by the node “credit_rating” [3].
The number of attributes corresponds to the depth of the DT, which is an important factor contributing to the computational cost for tree-growing. With the direct methods the algorithms extract the rules, one class at a time, for data sets that contain more than two classes.
On the contrary, for the indirect methods, the DT is created first and the rules afterwards. Theorems that discover the possibility of a complete DT and the identification of don’t care attributes are developed.
DMS can discover the rules with a number of conditions at most equal to the number of the attributes. It differs from the sequential covering algorithm [3] since it mainly works with attributes and their values. In addition, DMS can create a DT for each predefined class value. The merge of all the paths of different class values define the final DT.
A Decision Tree algorithm is created from data using probabilities (DTA), and the goal is to create on-demand an accurate decision tree (DT). To avoid repetition or replication by discovering the do n’t care attributes, the criterion of elimination o branch (CEB) is also applied. CEB examines the cases of extensions for all probable partitions of an attribute. A Neural Network (NN) is a collection of units that are connected in some pattern to allow communication between units. When many

of these units are connected together, the complete network is capable of performing a complicated task. The training of a network by backpropagation involves three stages: the feedforword of the input training pattern, the calculation and the back propagation of the associated error, and the adjustment of the weights [6]. In [7] a constructive algorithm for binary classification problems has been developed. In this work an incremental back propagation NN (IBNN) is presented using the same learning method as DMS, by processing instances that belong to the same desired class.

The article is organized as follows. Section 2, 3 include the DT and set and subsets with instruction to terms for DMS. Section 4 deals with DMS. Section 5 presents the DTA. Section 6, 7 includes the NN and the IBNN. Simulation results appear in Section 8.

2 DT

In decision trees [3],[4],[5] the input data set has one attribute called class C that takes a value from K discrete values 1, . . . , K, and a set of numeric and categorical attributesA1, .. . , Ap. The goal is to predict C given A1, . . . , Ap. A DT consists of nodes that are the attributes to be tested.

Decision tree algorithms automatically split numeric attributes Ai into two ranges and they split categorical attributes Aj into two subsets at each node. Decision tree is used to find predictive rules combining numeric and categorical attributes. The splitting process is recursively repeated until the end of data. Split of records is based on an attribute test that optimizes certain criterion (Greedy strategy). The multi-way split is used, where there are as many partitions as distinct values. Nodes with homogeneous class distribution are preferred. In the incomplete DT there are subtrees where both repetition and replication are included. Repetition happens when an attribute is repeatedly tested along a given branch.

3 SET, SUBSETS

Some definitions are needed for a set n of instances (records) that belong to different classes. There are two cases of any subset; the complete and incomplete.

It is assumed that each combination of conditions has a unique class label

Subset of instances: is the subset of instances for a definite value of class decision.

a. Complete subset of values for an i attribute is when it contains all the values for all the other ($j \neq i$) attributes for a subset of instances with the same predefined class. The complete subset also includes the case where a value of an attribute has the same predefined class for all the instances (*the one condition rule)*. For the one condition rule the accuracy is 100%. Then the dataset for the DMS is reduced with all the instances that contain the attribute with the maximum accuracy.

b. Incomplete subset of values for *an i attribute* is when it contains only some of the values for all the other ($j \neq i$) attributes for a subset of instances with the same predefined class.

For the complete case, the other attributes are eliminated and rules with only some conditions from a set of k attributes with k conditions only m, m<k remain in the rules.

The records are sequentially divided into smaller subsets based on their outcomes.

Predefined set of values of an i attribute (pset i) is the set of all the values of i attribute

A complete set for a value of an attribute eliminate all the others and the rule is: ai=vi when class =ck.

Sub-branch : is the branch that comes from the same attribute value and ends up to a predefined outcome.

Example 3: Let's consider a set of 14 instances for the triangles and squares dataset (with categorical and binary attributes). In this set there are two classes labels : (triangle and square) and three attributes: color(green, yellow, red), outline(dashed, solid), dot (yes. no). The set is as follows:

	color	outline	dot	shape
1	green	dashed	no	triangle
2	green	dashed	yes	triangle
3	yellow	dashed	no	square
4	red	dashed	no	square
5	red	solid	no	square
6	red	solid	yes	triangle

7 green solid no square
8 green dashed no triangle
9 yellow solid yes square
10 red solid no square
11 green solid yes square
12 yellow dashed yes square
13 yellow solid no square
14 red dashed yes triangle

The color: yellow create a complete subset of instances with a predefined class value "square". These instances contain "complete" set of values for the rest of the attributes outline (dashed, solid) and dot (yes,no).

The other values of the color attribute (green, red) create incomplete subsets can create one condition rules.

One sub-branch is the branch that comes from the same attribute value (color=green) and ends up to predefined outcome.(shape=triangle). Another sub-branch with the same attribute value (color=green) and ends up other predefined outcome (shape=square). Both of them can create the branch of the (color=green)

4 DMS

The direct method with simplification (DMS) extracts rules directly from data as the sequential covering algorithm [1]. The criterion for deciding which class should be generated first (*basic attribute*) depends on the number of factors, the class prevalence or the cost of misclassifying records from a given class. A hash table is used with m-1 hashes each for every attribute (except for the basic one).

For the initialization the hi (i=1..n-1) have zero values for the i attribute with m indexes [0..0]. In the process when a record is read and a change to a value of an attribute (with index k) happens, this index will take the value 1 and the vector becomes : $[0_1..1_k..0_m]$. If an attribute takes all the predefined values, the vector becomes [1…1] and this attribute is eliminated from the rule. Otherwise, the attributes with their values remain in the rule. The DMS can also create a DT with updated nodes after the creation of the rule.

Example 4: From the triangles and squares dataset, having the color as basic attribute, the vector has zero value before the start of the process. For:h1 = h(outline)=v1[dashed, solid] = v1[0,0] (initial value). If a record contains dashed outline then h1 will become v1[1,0]. When all indexes have been referred (here: solid) then h1 vector becomes v1[1,1]. In that case the outline attribute is eliminated from the rule.

The yellow node of the DT will be created and connected with the color (root).

The DMS pseudocode is as follows:

DMS Input : k: # of records(instances), m: # of i attributes, ni = # values for i attributes , seti : the set of variables of i attribute from the subset records , pseti: is the predefined set of variables for the i attribute from a subset of records

Output: rules for each class ,and the DT

define the basic attribute and the order of the attributes that will be processed.
define the subset of records of the desired outcome
sort the values of the basic attribute (i.e. color)
//test if there is *the one condition rule).*
for each class
if (for the desired outcome (ck) there is an attribute with the same value(attri =valm)
{ there is one condition rule
create node with attri
connect with root}
consider a hash table with m-1 hash functions
initialization of the vectors of hash functions (one for each attribute)
while (not end of the subset records)
{ read records
update vectors of (m-1) attributes
//for (values of i attribute = ni)
if ((pseti = seti) || (vi = 1s)) //for all the attributes except the basic
{ the attri is eliminated
update the rules –complete rule-
create the new node and the path
}
if ((pseti = seti) || (vi = 1s) -for i attributes-) ∧ ((psetj ≠ setj) (for j ≠ i)
{update the hash vectors with the new values of attributes
update the rules -non complete rule-
create the new node and the path (the conditions)
}
}

print the rules
}

Example 5: Let us consider the set of records for "Triangles and Squares". The attributes: color (green, red, yellow), outline (dashed, solid), dot(yes,no) and decision for shape is triangle or square. After sorting in regards of color attribute and picking up the instances with triangle, the list with the three nodes will be:
Color outline dot shape, Green dashed no triangle, Green dashed yes triangle
From the two instances after the dot simplification, the rule becomes: green, dashed -> triangle

Example 6: The main attribute "color" has been picked up. Two are the subsets of the data : the shape = triangle and the shape = square. The first subset: shape = triangle is:
color=green outline= dashed dot=no => shape=triangle (1)
color=green outline= dashed dot=yes => shape=triangle (2)
color=green outline= dashed dot=no => shape=triangle (3)
from (1), (2) : color=green and outline = dashed => shape=triangle (a)
The (3) remain as it is. This is the first sub-branch for (color=green) and shape= triangle)
For the second subset (shape= square) after sorting of color, with color=green there are complete set of the values of the other attributes (outline and dot). So the rule is eliminated to: if color=yellow then shape =square
color=yellow outline= dashed dot=no => shape=square
color=yellow outline= solid dot=yes => shape=square
color=yellow outline= dashed dot=yes=> shape=square
color=yellow outline= solid dot=no => shape=square
From all the above instances the rule is eliminated to: if color=yellow then shape =square.
If the work continues with the second outcome(shape=square) then form the square subset with color= green we can similarly get :
color=green outline= solid dot=no => shape=square (4)
color=green outline= solid dot=yes => shape=square (5)
from (4),(5) color=green and outline= solid => shape=square (b)
From (1),..,(5) means that a node with both root =color and value =green, a new node that has three triangles and four squares can be created. Hence, the decision subtree T: root and attribute value = green can be created. Therefore, using the root =color the branches ((a),(b)) for each attribute value can be created from each subset of the desired outcome. This is the second sub-branch for (color=green) and shape=square)

Theorem 1: With DMS it is not possible to have maximum size of conditions greater than the number of attribute. DMS also provide DT with no repetitions.

Proof: This is because for each attribute only one value can be selected from any record. In addition it is not possible to have repetitions. •

For the final DT merge of rules is needed. This is achieved by adding all the other rules in the initial DT (the DT from the first subset). If the node exists and there is a new condition then a new subpath for another class value is created.

5 DTA

The DTA is based on the "most informative" attribute. It uses the criterion of maximum probability and can be created in the following phases:

Phase 1: Discover the root (i) (from all the attributes)

$$P(E_{Ai}) = \sum_{Ci} \sum_{Ai} p(Ai) * p(Ci / Ai) \quad ,$$

where Ai : the attributes of the tuples and Ci the classes (attribute test). MP = max (P(EAi)) //max attribute test criterion

Phase 2: split the data into smaller subsets, so that the partition to be as pure as possible using the same formula. The measure of nodes impurity is the MP. Continue until the end of the attributes.
There is a stopping criterion for expanding a node when all records belong to the same class.

DTA : Input : training data
Output: decision tree
1. define root node (phase 1)

2. discover the branches from root
while (! end of the attributes)
{3. splitting the attribute (phase 2) }

Example 7:

Weather	parents	money	decision
Sunny	yes	rich	cinema
Sunny	no	rich	Tennis
…..			
Windy	no	rich	cinema
…..			

P(E) = (5/10)*(5/10) + (5/10)*(1/10) = 0.3 (phase 1)
Weather: class, P(E) = (3/10)*(1/10) + (4/10)*(3/10) + (3/10)*(2/10) = 0.21
The CEB, is used to eliminate redundant branches. It is a prepruning approach [1]. It can be used also in phase2 (splitting the attribute) of DTA in order to avoid the repetitions or replications.
For an attribute (attr1) with value v1, if there are tuples from attr2 that have all the values in relation with v1 (of attr1) then the attr2 is named as: *do n't care* attribute (see Example8).
The way of discovering the existence of don't care attributes is developed by using conditional probabilities.
P_{CEB} = P(A_1 =a_1,…, $A_{|A|}$ = $a_{|A|}$ | C=ci) = $\prod_{i=1}^{|A|} p(A_i = a_i \mid C = c_j)$
A branch is eliminated when the $P_{CEB} \neq 0$. The CEB criterion can determine the existence of a small DT avoiding repetitions and replications.
The criterion of Elimination of Branch (CEB): states that if the $P_{CEB}= 0$, between two attributes (A1, A2) then A2 is don't care attribute. The CEB criterion is valid when $P_{CEB} \neq 0$. A branch is eliminated when the $P_{CEB} \neq 0$. If P_{CEB} =0 new partitions have to be included in the DT. Considering all the above the CEB is used to develop the DT so that repetition or replication is avoided. The next theorem can provide the possibility of the existence of a complete DT in advance given the data or the rules.
Theorem 2: The CEB criterion can determine the existence of a small DT with the best accuracy (100%, or complete) avoiding repetitions and replications.
Proof: Because, if CEB criterion is valid discourage the repetition •
Example 8:

Age	Has_job	Own-house	Credit-rating	class
Young	false	False	fair	No
Young	false	False	Good	No
Young	True	False	Good	Yes
Young	True	True	fair	Yes
Middle	True	True	Good	Yes
Old	false	true	Excellent	Yes
……..				

Of the DT (with own_house as a root) it is not necessary to have more extension with the attribute of age for all the probable partitions ("young"," middle", "old"). P_{CEB} = P(A1 =a1,…, A|A| = a|A| | C=ci) = P(age =young , own_house="y" | C="yes") =P(age =young | C="yes") * P(own_house="y" | C="yes") = 2/5 * 6/9 $\neq 0$.
From the above since $P_{CEB} \neq 0$ there is a direct path from root to decision (Yes), without considering the age attribute.
Both DMS and DTA try to discover complete trees avoiding repetitions, by using pseti (predefined set of variables for an i attribute) for DMS and P_{CEB} for DTA.

6 NEURAL NETWORKS

DT is a very popular and practical approach for pattern classification. DTs have many pros (comprehensible, easy to design and implement) but also some cons (many become very large for complex problems, difficult to know the true concept, and too many rules to be understood by human users).
To avoid the cons of DT and for classification reasons, a Neural Network can be used.
For the NN, the test data set and the training data set should be disjoint (so that test data are not used during training). The knowledge of the solved task resides in the network's weight. The number of neurons on the input layer corresponds to the number of attributes that represent a sample. One output unit, on the output layer, may be used to represent exactly one class. The back propagation algorithm is in wide use because it learns by adapting its weight using the generalized delta rule

which attempts to minimize the squared error between what is the desired network output and the actual network output. During learning it continually cycles through the data until the error is at a low enough value for the task to be considered solved. In this work a backpropagation NN with a single hidden layer is used since this architecture can represent any continuous mapping of the form y=f(x). The weights are updated after each pattern presentation [5].

7 INCREMANTAL BACKPROPAGATION NEURAL NETWORK

For each rule created from the DMS a NN can be created. In knowledge extraction using DMS, the network is initially trained with the training data set of the preselected output. From these data the network is created with another data set which only contains input samples. For the constructive algorithm, an initial structure is considered by having one unit in the hidden layer. A training technique, working with a group based on their class, is introduced in order to avoid time and space complexity. It learns by adapting the weight using the delta rule which attempts to minimize the squared error. Following training, the network's weights are fixed and the network can then be used to classify an unknown pattern. The training for the new network is based on the newly added instances and freezing the previous weight. By working with class groups' instances the rules created by DMS are tried to be used in NN. In case that there is no improvement with the weight change, a new unit (neuron) is added into the hidden layer.

8 SIMULATION

Four experiments are provided as follows:

1. *ID3 vs DMS :* DMS extract rules for car data faster than ID3 (Figure 1). ID3 needs to create the tree first and then to produce the rules.

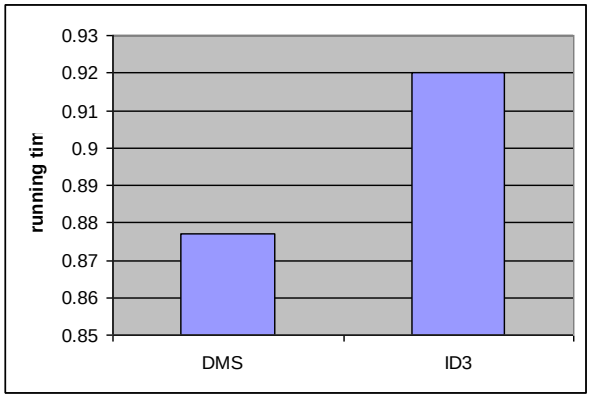

Figure 1. ID3 vs DMS

2. *ID3 vs DTA:* using the data set "iris". DTA results are slightly better than the ones of ID3. (Figure 2).

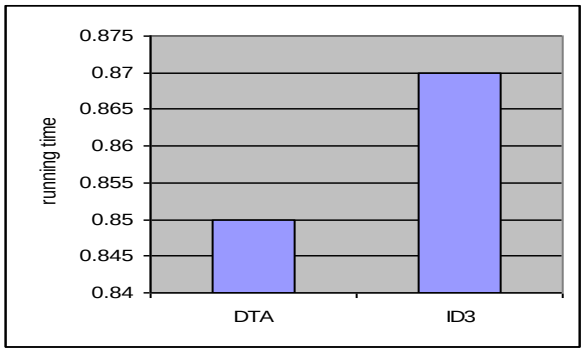

Figure 2. ID3 vs DTA

3. *DMS vs IBNN: DMS* first creates the rules for the wine dataset, then the DT, and after that reads the tree to decide for classification (Figure 3). The IBNN can get answer for classification faster.

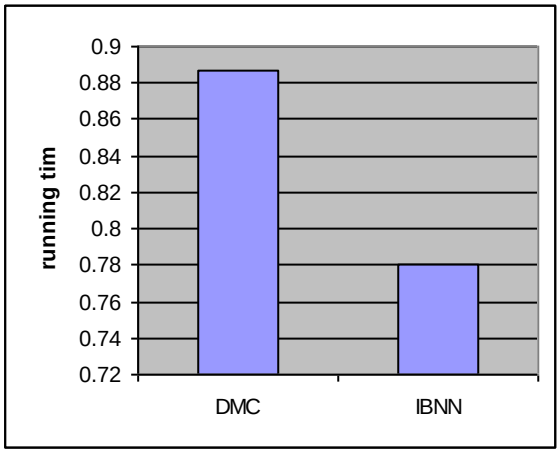

Figure 3. DMS vs IBNN for speed

4. *DMS vs IBNN:* IBNN has better accuracy than DMS (Figure 4).

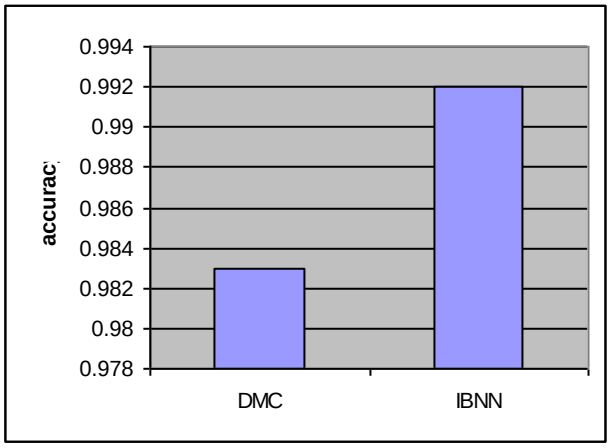

Figure 4. DMS vs IBNN for accuracy

9 CONCLUSION

The purpose of this project is threefold. First, there is a new algorithm (DMS) for extraction rules from data and creation of a DT . Second, DTA a direct method for DTs based on probabilities . The P_{CEB} criterion can minimize the height of the DT. Third, there is construction of an incremental backpropagation neural network (IBNN) algorithm for classification. The IBNN works like DMS with grouped instances, according to the desired output. DMS outperforms ID3. DMS provides rules with conditions at most equal to the number of attributes and it can also create a DT. For DTA, a criterion for avoiding repetition of branches has been developed. IBNN, having an initial architecture, changes the weights and adds new units when processing the instances until the final desired structure and output is reached. Future work will focus on Neural Networks and SVM.

REFERENCES

[1] J. Shafer, R. Agrawal, M. Mehta , "SPRINT, A scalable parallel classifier for data mining", In Proc. of 22nd Intern. Conf. on Very Large Databases, Morgan Kaufmann, p544-555
[2] C. Perlish , F. Provost, J. Simonoff, " Tree Induction vs. logistic regression: A learning-curve analysis , Machine Learning research 4, p211-255
[3] J. Han, ,M. Kamber, J. Pei, " Data Mining Concepts and Techniques", Morgan Kaufman, 3 ed, 2012
[4] U. Fayyad and G. Piateski-Shapiro, " From Data Mining to Knowledge Discovery", MIT Press, 1995.
[5] M. Karntardzic, "Data Mining: Concepts, Models, Methods, and Algorithms", IEEE Press, 2003.
[6] L. Fausett, "Fundamentals of Neural Networks : Architectures, Algorithms, and Applications, Prentice Hall, NJ, 1994
[7] M. Frean, "The upstart algorithm: a method for constructing and training feedforward neural networks", IEEE Transaction on Neural Networks, vol.2, pp.198-209,1990
[8] C. Ordonez, Comparing Association rules and Decision Trees for Disease Prediction, *HIKM 2006*, No. 11, 2011, Virginia