IJCSMS International Journal of Computer Science & Management Studies, Vol. 12, Issue 03, Sept 2012
ISSN (Online): 2231 –5268
www.ijcsms.com

A Hybrid Approach Using C Mean and CART for Classification in Data Mining

Jasbir Malik[1], Rajkumar[2]

[1]M Tech (scholar), CSE Department, JIET, JIND, Kurukshetra University, Kurukshetra
malik_jasbir@ymail.com

[2]Assistant Professor, CSE Department, JIET, JIND, Kurukshetra University, Kurukshetra
rajshira@gmail.com

Abstract

Data Mining is a field of search and researches of data. Mining the data means fetching out a piece of data from a huge data block. The basic work in the data mining can be categorized in two subsequent ways. One is called classification and the other is called clustering. Although both refers to some kind of same region but still there are differences in both the terms. The classification of the data is only possible if you have modified and identified the clusters. In the presented research paper, our aim is to find out the maximum number of clusters in a specified region by applying the area searching algorithms. Classification is always based on two things. a)The area which you choose for the classification that is the cluster region .b)The kind of dataset which you are going to apply on the selected region .To increase the accuracy of the searching technique, any one would need to focus on two things . a)Whether the data set has been cauterized in proper manner or not .b)If the clusters are defined , whether they fit into the appropriate classified area or not .

Keywords: *Data Mining, C-mean, CART, KDD, SVM-Algorithm.*

1. Introduction

With the enormous amount of data stored in files, databases, and other repositories, it is increasingly important, if not necessary, to develop powerful means for analysis and perhaps interpretation of such data and for the extraction of interesting knowledge that could help in decision-making. Data Mining, also popularly known as Knowledge Discovery in Databases (KDD), refers to the nontrivial extraction of implicit, previously unknown and potentially useful information from data in databases. Data mining refers to extracting or "mining" knowledge from large amounts of data.

Classification (technique to analyses the *frequent item sets*) is one of the major fields in the area of extracting knowledge from vast data. A *frequent item set* typically refers to a set of items that frequently appear together in a transactional data set, such as milk and bread (Han & Kamber, 2001). In this chapter, we will briefly review about data mining, its architecture, functionalities, classification, methods of classification etc. We are in an age often referred to as the information age. In this information age; because we believe that information leads to power and success, and thanks to sophisticated technologies such as computers, satellites, etc., we have been collecting tremendous amounts of information. Initially, with the advent of computers and means for mass digital storage, we started collecting and storing all sorts of data, counting on the power of computers to help sort through this amalgam of information. Unfortunately, these massive collections of data stored on disparate structures very rapidly became overwhelming. This initial chaos has led to the creation of structured databases and database management systems (DBMS). The efficient database management systems have been very important assets for management of a large corpus of data and especially for effective and efficient retrieval of particular information from a large collection whenever needed. The proliferation of database management systems has also contributed to recent massive gathering of all sorts of information. Today, we have far more information than we can handle: from business transactions and scientific data, to satellite pictures, text reports and military intelligence. Information retrieval is simply not enough anymore for decision-making. Confronted with huge collections of data, we

have now created new needs to help us make better managerial choices. These needs are automatic summarization of data, extraction of the "essence" of information stored, and the discovery of patterns in raw data.

2. C- Mean Algorithm

C 4.5 builds decision trees from a set of training data in the same way as ID3, using the concept of information entropy. The training data is a set *S* = *s*1,*s*2,... of already classified samples. Each sample *si* = *x*1,*x*2,... is a vector where *x*1,*x*2,... represent attributes or features of the sample. The training data is augmented with a vector *C* = *c*1,*c*2,... where *c*1,*c*2,... represent the class to which each sample belongs. C4.5 belongs to a succession of decision tree learners that trace their origins back to the work of Hunt and others in the late 1950s and early 1960s (Hunt 1962). Its immediate predecessors were ID3 (Quinlan 1979), a simple system consisting initially of about 600 lines of Pascal, and C4 (Quinlan 1987). C4.5 has grown to about 9,000 lines of C that is available on diskette with Quinlan (1993). Input to C4.5 consists of a collection of training cases, each having a tuple of values for a fixed set of attributes (or independent variables) A = fA1; A2; :::; A kg and a class attribute (or dependent variable). An attribute Aa is described as continuous or discrete according to whether its values are numeric or nominal. The class attribute C is discrete and has values C1; C2; ::::; Cx. The goal is to learn from the training cases a function DOM(A1) fi DOM(A2) fi ::: fi DOM(Ak) ! DOM(C) that maps from the attribute values to a predicted class. The distinguishing characteristic of learning systems is the form in which this function is expressed. A decision tree is depicted as a recursive structure of a leaf node labelled with a class value, or a test node that has two or more outcomes, each linked to a sub tree.

2.1 Implementation of C4.5 Algorithm on Provided Data

The data obtained from the excel sheet has been used as a source of data in paper so as to predict the outcome of the student in the university exam. However a slight modification has been done in the same data for a better prediction.

2.2 Building Classification Trees

In the previous section we saw that the construction of a classification tree starts with performing good splits on the data. In this section we define what such a good split is and how we can find such a split. Three impurity measures, resubstitution-error, gini-index and the entropy, for splitting data will be discussed in . The actual splittingand tree construction according to these splits.

2.3 Impurity Measures

A split that will separate the data as much as possible in accordance with the class labels. So the objective is to obtain nodes that contain cases of a single class only as mentioned before. We define impurity as a function of the relative frequencies of the classes in that node's(t) = fi(p1, p2, ..., pJ) (1) with pj (j = 1, ..., J) as the relative frequencies of the J different classes in that node To compare all the possible splits of the data you have, a quality of a split as the reduction of impurity that the split achieves must be defined. In the example later on the following equation for the impurity reduction of split s in node t will be used: where fi(j) is the proportion of cases sent to branch j by s, and i(j) is the impurity of the node of branch j. Because different algorithms of tree construction use different impurity measures, we will discuss three of them and give a general example later on resubstitution error This is a measure for the impurity defined by the fraction of the cases in a node that is classified incorrectly if we assign every case to the majority class in that node: i(t) = 1 – max j p(j|t) (3) where p(j|t) is the relative frequency of class j in node t. The resubstitution error gives a score to a split according to the incorrectly classified cases in a node. It can recognize a better split if it has less error in that node. But the resubstitution error has one major disadvantage: it does not recognize a split as a better one if one of its resulting nodes is pure. So it does not prefer Split 2 over Split 1 in Figure 2. In such a case we want a split with a pure node to be preferred. Gini index The Gini index does recognize such a split. Its impurity measure is defined as follows: i(t) =X j p(j|t)(1 − p(j|t)) (4) Entropy Finally we have the entropy measure which is used in well-known classification tree algorithms like ID3 and C4.5. The advantage of

the entropy measure over the gini-index is that it will reach minimum faster if more instances of the child nodes belong to the same class. Splits to consider we have looked at different impurity criteria for computing the quality of a split. In this section we look at which splits are considered and how we select the best split (for binary splits only). The attributes can have numerical or categorical values. In the case of numerical values, all the values of the attributes occurring in the training set are considered. The possible splits are made between two consecutive numerical values occurring in the training set. If the attribute is categorical with N categories, then 2N−1−1 splits are considered. There are 2N−2 non-empty proper subsets of a set of N elements. The empty set and the complete set do not count. Furthermore a split of the N categories into S and Sc.

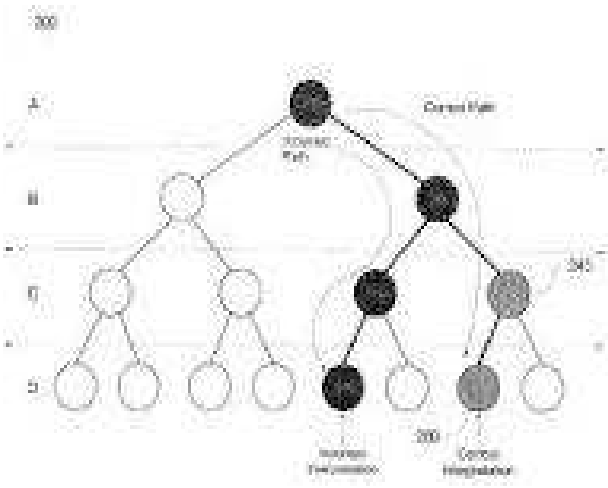
2.4 Tree Construction

Building a classification tree starts at the top of the tree with all the data. For all the attributes the best split of the data must be computed. Then the best splits for each of the attributes are compared. The attribute with the best split wins. The split will be executed on the attribute with the best value of the best split (again we consider binary trees). The data is now separated to the corresponding branches and from here the computation on the rest of the nodes will continue in the same manner. Tree construction will finish when there is no more data to separate or no more attributes to separate them by Over fitting and Pruning.
If possible we continue splitting until all leaf nodes of the tree contain examples of a single class. But unless the problem is deterministic, this will not result in a good tree for prediction. We call this overfitting. The tree will be focused too much on the training data. To prevent overfitting we can use stopping rules; stop expanding nodes if the impurity reduction of the best split is below some threshold. A major disadvantage of stopping rules is that sometimes, first a weak (not weaker) split is needed to be able to follow up with a good split. This can be seen in building a tree for the XOR problem practDM. Another solution is pruning. First grow a maximum-size tree on the training sample and then prune this large tree. The objective is to select the pruned subtree that has the lowest true error rate. The problem is, how to find this pruned subtree .There are two pruning methods we will use in the tests, cost-complexity pruning [1] and [5] and reduced-error pruning [3]. In the next two paragraphs we will explain how the two pruning methods work and finish with a concrete example of the pruning process. Cost-complexity pruning the basic idea of cost-complexity pruning is not to consider all pruned subtrees, but only those that are the “best of their kind” in a sense to be defined below. Let R(T) (T stands for the complete tree) denote the fraction of cases in the training sample that is misclassified by the tree T (R(T) is the weighted summed error of the leafs of tree T). Total cost Cfi(T) of tree T is defined as: Cfi(T) = R(T) + fi| ˜ T| (7) The total cost of tree T then consists of two components: summed error of the leafs R(T), and a penalty for the complexity of the tree fi| ˜ T|. In this expression ˜ T stands for the set of leaf nodes of T, | ˜ T| the number of leaf nodes and fi is the parameter that determines the complexity penalty: when the number of leaf nodes increases by one (one additional split in a binary tree), then the total cost (if R remains equal) increases with fi [5]. The value of fi can make a complex tree with no errors have a higher total cost than a small tree making a number of errors. For every value of fi there is a smallest minimizing subtree. We state the complete tree by Tmax. For a fixed value of fi there is a unique smallest minimizing subtree T(fi) of Tmax.

3. CART algorithm

Classification and Regression Trees is a classification method which uses historical data to construct so-called decision trees. Decision trees are then used to classify new data. In order to use CART we need to know number of classes a priori. CART methodology was developed in 80s by Breiman, Freidman, Olshen, Stone in their paper ”Classification and Regression Trees” (1984). For building decision trees, CART uses so-called learning sample - a set of historical data with pre-assigned classes for all observations. For example, learning sample for credit scoring system would be fundamental information about previous borrows (variables) matched with actual payoff results (classes). Decision trees are represented by a set of questions which splits the learning sample into smaller and smaller parts. CART asks only yes/no questions. A possible question could be: ”Is age greater than 50?” or ”Is sex male?”.

CART algorithm will search for all possible variables and all possible values in order to find the best split – the question that splits the data into two parts with maximum homogeneity. The processes then repeated for each of the resulting data fragments. Here is an example of simple classification tree, used by San Diego Medical Center for classification of their patients to different levels of risk:



In practice there can be much more complicated decision trees which can include dozens of levels and hundreds of variables. As it can be seen from figure 1.1, CART can easily handle both numerical and categorical variables. Among other advantages of CART method is its robustness to outliers. Usually the splitting algorithm will isolate outliers in individual node or nodes. An important practical property of CART is that the structure of its classification or regression trees is invariant with respect to monotone transformations of independent variables. One can replace any variable with its logarithm or square root value, the structure of the tree will not change.

CART methodology consists of tree parts:

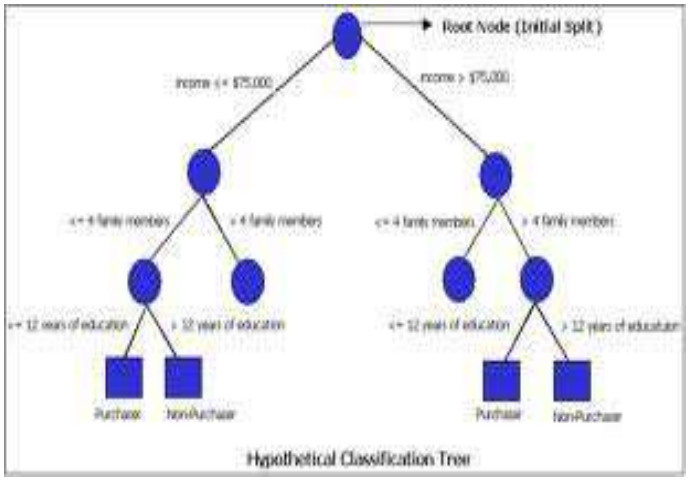
1. Construction of maximum tree
2. Choice of the right tree size
3. Classification of new data using constructed tree

Construction of Maximum Tree

This part is most time consuming. Building the maximum tree implies splitting the learning sample up to last observations, i.e. when terminal nodes contain observations only of one class. Splitting algorithms are different for classification and regression trees. Let us first consider the construction of classification trees.

3.1 Classification Tree

Classification trees are used when for each observation of learning sample we know the class in advance. Classes in learning sample may be provided by user or calculated in accordance with some exogenous rule. For example, for stocks trading project, the class can be computed as a subject to real change of asset price. Let tp be a parent node and tl,tr - respectively left and tight child nodes of parent node tp. Consider the learning sample with variable matrix X with M number of variables xj and N observations. Let class vector Y consist of N observations with total amount of K classes. Classification tree is built in accordance with splitting rule - the rule that performs the splitting of learning sample into smaller parts. We already know that each time data have to be divided into two parts with maximum homogeneity:



where tp, tl, tr - parent, left and right nodes; xj - variable j; xRj best splitting value of variable xj . Maximum homogeneity of child nodes is defined by so-called impurity function i(t). Since the impurity of parent node tp is constant for any of the possible splits xj fi xRj , j = 1, . . . ,M, the maximum homogeneity of left and right child nodes will be equivalent to the maximization of change of impurity function fii(t): i(t) = i(tp) − E[i(tc)] where tc - left and right child nodes of the parent node tp. Assuming that the Pl, Pr - probabilities of right and left nodes, we get i(t) =

i(tp) − Pli(tl) − Pri(tr) Therefore, at each node CART solves the following maximization problem: arg max xjfixRj, j=1,...,M [i(tp) − Pli(tl) − Pri(tr)] Equation implies that CART will search through all possible values of all variables in matrix X for the best split question xj < xRj which will maximize the change of impurity measure fii(t). The next important question is how to define the impurity function i(t). In theory there are several impurity functions, but only two of them are widely used in practice: Gini splitting rule and Twoing splitting rule. Gini splitting rule (or Gini index) is most broadly used rule. It uses the following impurity function i(t): i(t) =Xk6=l p(k|t)p(l|t) (2.2) where k, l 1, . . . ,K - index of the class; p(k|t) - conditional probability of class k
provided we are in node t. Applying the Gini impurity function 2.2 to maximization problem 2.1 we will get the following change of impurity measure fii(t): **I**n a classification problem, we have a training sample of *n* observations on a class variable *Y* that takes values 1, 2, . . . , *k*, and *p* predictor variables, *X*1, . . . , *Xp*. Our goal is to find a model for predicting the values of *Y* from new *X* values. In theory, the solution is simply a partition of the *X* space into *k* disjoint sets, *A*1, *A*2, . . . , *Ak*, such that the predicted value of *Y* is *j* if *X* belongs to *Aj*, for *j* = 1, 2, . . . , *k*. If the *X* variables take ordered values, two classical solutions are linear discriminant analysis1 and nearest neighbor classification.2 These methods yield sets *Aj* with piecewise linear and nonlinear, respectively, boundaries that are not easy to interpret if *p* is large. Classification tree methods yield rectangular sets *Aj* by recursively partitioning the data set one *X* variable at a time. This makes the sets easier to interpret. For example, Figure 1 gives an example wherein there are three classes and two *X* variables.
The left panel plots the data points and partitions and the right panel shows the corresponding decision tree structure. A key advantage of the tree structure is its applicability to any number of variables, whereas the plot on its left is limited to at most two. The first published classification tree algorithm is THAID.3,4 Employing a measure of node impurity based on the distribution of the observed *Y* values in the node, THAID splits a node by exhaustively searching
X takes ordered values, the set *S* is an interval of the form (−∞, *c*]. Otherwise, *S* is a subset of the values taken by *X*. The process is applied recursively on the data in each child node. Splitting stops if the relative decrease in impurity is below a prespecified threshold. Algorithm 1 gives the pseudo code for the basic steps.

Algorithm 1

Pseudo code for tree construction by exhaustive search
1. Start at the root node.
2. For each X, find the set S that minimizes the sum of the node impurities in the two child nodes and choose the split {X∗ ∈ S∗} that gives the minimum overall X and S.
3. If a stopping criterion is reached, exit. Otherwise, apply step 2 to each child node in Turn.
C4.55 and CART6 are two later classification tree algorithms that follow this approach. C4.5 uses entropy for its impurity function, whereas CART uses a generalization of the binomial variance called the Gini index. Unlike THAID, however, they first grow an overly large tree and then prune it to a smaller size to minimize an estimate of the misclassification error. CART employs 10-fold (default) cross validation, whereas C4.5 uses a heuristic formula to estimate error rates. CART is implemented in the R system7 as RPART,8 which we use in the examples below.
Despite its simplicity and elegance, the exhaustive search approach has an undesirable property. Note that an ordered variable with *m* distinct values has (*m*− 1) splits of the form $X \leq c$.

4. Proposed Work

In the current Industry of the data mining, the efficiency to increase the identification area has been a major task which has been getting implemented using different types of algorithm. According to the law of cycle, hundred percent efficiency is not possible but still the researchers have been trying to take maximum out of it. Different scientist has implemented their different types of parameters and criteria to figure out how effective the classification could be. Few names has left their mark in such kind of work with their popular proposed algorithms like CART, SVM C MEAN and many more. Our aim is to find a hybrid algorithm which can implement better result than the proposed algorithms till now. For this aim, we are going to combine two most effective results like CART

AND CMEAN to implement a sophisticated architecture which has both the features like tree architecture and a mean square average algorithm. By combining two algorithms we would be definitely be able to use both the features and the results would be definitely better.

5. Conclusions

This paper concludes that the accuracy of classifying data can be improved by hybrid the feature of C mean and CART algorithm and their implementation on the MATLAB shows their resuls improvement in terms of accuracy compared to existing classifying algorithm

References

[1] Al-Kilidar, H., Cox, K., Kitchenham, B.: The Use and Usefulness of the ISO/IEC 9126 Quality Standard. International Symposium on Empirical Software Engineering, 7 p. (2005) .

[2] Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., Merritt, M.: Characteristics of Software Quality. North Holland (1978) .

[3] Bøegh, J.: A New Standard for Quality Requirements. IEEE Software 25(2), 57--63 (2008) .

[4] Dromey, R.G.: Concerning the Chimera. IEEE Software 13 (1), pp. 33--43 (1996).

[5] ISO, International Organization for Standardization: ISO 9126-1:2001, Software Engineering – Product Quality, Part 1: Quality model (2001) .

[6] Lindland, O.I., Sindre, G., Solvberg, A.: Understanding Quality in Conceptual Modeling. IEEE Software 11(2), pp. 42--49 (1994) .

[7] McCall, J. A., Richards, P. K., Walters, G. F.: Factors in Software Quality. Nat'l Tech. Information Service, Vol. 1, 2 and 3 (1977).

[8] Mohagheghi, P., Aagedal, J.Ø.: Evaluating Quality in Model-Driven Engineering. In: Workshop on Modeling in Software Engineering (MISE'07), In: Proc. of ICSE'07, 6. p (2007) .

[9] Wagner, S., Deissenboeck, F.: An Integrated Approach to Quality Modeling. Fifth International Workshop on Software Quality, In: Proc. of ICSE'07, 6 p. (2007).

[10] Joc Sanders and Eugene Curran. *Software Quality, a Framework for success in software Development and Support*. Addison - Wesley Publishing Company, 1995.

[11] Safa, L. The practice of deploying DSM, report from a Japanese appliance maker trenches. In Proceedings of the 6th OOPSLA Workshop on Domain Specific Modeling (DSM'06), 2006, 12p.