Chi-Square Test Based Decision Trees Induction in Distributed Environment

Jie Ouyang  
Department of Computer Science and Engineering  
Intelligent Information Engineering Lab  
Oakland University, Rochester, MI 48309  
jouyang@oakland.edu

Nilesh Patel  
Department of Computer Science and Engineering  
Intelligent Information Engineering Lab  
Oakland University, Rochester, MI 48309  
npatel@oakland.edu

Ishwar K. Sethi  
Department of Computer Science and Engineering  
Intelligent Information Engineering Lab  
Oakland University, Rochester, MI 48309  
isethi@oakland.edu

Abstract

The decision tree-based classification is a popular approach for pattern recognition and data mining. Most decision tree induction methods assume training data being present at one central location. Given the growth in distributed databases at geographically dispersed locations, the methods for decision tree induction in distributed settings are gaining importance. This paper describes one distributed learning algorithm which extends the original (centralized) CHAID algorithm to its distributed version. This distributed algorithm generates exactly the same results as its centralized counterpart. For completeness, a distributed quantization method is proposed so that continuous data can be processed by our algorithm. Experimental results for several well known data sets are presented and compared with decision trees generated using CHAID with centrally stored data.

1 Introduction

The decision tree-based classification is a popular approach for pattern recognition and data mining. Methods for the induction of decision trees have been studied for past several decades. Once built, a decision tree can be used to classify previously unseen instances of patterns or to characterize patterns of different classes in the form of rules to constitute a knowledge base for decision support. The decision tree methodology has been applied to numerous applications in different domains. A comprehensive survey of decision tree induction methods from multiple disciplines and their applications is provided by Murthy[1]. Almost all methods reviewed in this survey are based on the assumption that the training data for building the decision tree is present at one central site.

With increasing globalization of businesses, advances in technology, and increasing concerns about the loss of privacy due to data mining, the interest in data mining methods that can operate in a distributed data environment has been growing. While it is always possible to move data to one central location for mining, the costs for such a move can be high. Consequently, methods for decision tree induction in a distributed data environment without large scale movement of data have started receiving attention. A distributed data environment can be homogeneous or heterogeneous. In a homogeneous environment, different sites record similar information albeit for different objects. The distributed data in such situations is also termed as horizontally partitioned data. In a heterogeneous environment, all sites record information for the same set of objects albeit different aspects of the information. Such data is referred as vertically parti-
tioned data. Of course in reality, data may be both horizontally and vertically partitioned.

Two basic approaches to distributed decision tree induction are possible. One approach is to have an ensemble of decision trees with each data site contributing its own local decision tree to the ensemble. Examples of some popular ensemble methods are boosting[2], bagging[3], and random forests[4]. Any of these methods can be employed in a distributed environment. One problem with ensemble approach is the difficulty of converting the ensemble decision to rules to form a knowledge base. Another drawback is that it applies only to the horizontally partitioned data scenario. The other approach to distributed decision tree induction is to develop only one single decision tree by organizing the computation for a traditional decision tree induction method at one site in such a way that traffic between different data sites is minimized. The resulting decision tree then can be used by each site independently if so desired. This approach is applicable to both the horizontally and the vertically partitioned data scenarios. Furthermore, the resulting single decision tree is useful for a knowledge base across the entire enterprise. The existing methods following this approach build single feature decision trees for individual features using entropy[5] or Gini index[6] as splitting criteria. Since building single feature split decision trees needs only different counts to evaluate individual features as potential split candidates, the methods are easily extensible to the distributed data settings.

In this paper, we extend the original decision tree induction algorithm based on chi-squared automatic interaction detection (CHAID) proposed by Kass[7] to distributed data. The organization of the paper is as follows. Section 2 reviews the related work in this area. Section 3 presents the induction of distributed CHAID and performance tuning parameters. In section 4 we provide empirical proof and validate the accuracy and performance of proposed method. Finally, we summarize the work and provide future directions in Section 5.

2 Related Work

The most common approach for building a decision tree classifier in a distributed setting uses the tree ensembling technique which combines the locally trained classifiers into a single decision. Boosting is a typical ensembling technique and its extension to distributed environment can be found in[8][9]. Boosting algorithms are suitable for distributed environments in that, each weak learner is trained on a local site using local data only. The final classification is weighted average of weak classifiers. Thus, theoretically, no raw data transfer is needed and the communication overhead is small. The work described in [8] focused on generating unskewed sub(local) data set for weak learner training. Two sampling methods, r-sampling and d-sampling, were proposed. For each iteration of boosting, r-sampling randomly picks a fixed number of instances without replacement while d-sampling picks one of the disjoint subsets. The d-sampling is suitable for distributed environment where weak learners are trained on different local sites. The work presented in [9] explicitly describes the framework of distributed boosting. The framework requires each site keep local distribution, local weight and a copy of the global distribution of samples so that the sampling at each run can be similar to the centralized case. At each iteration, the sampling happens on each local site based on the global distribution. Only those sampled local instances are used to train local weak learners. Let $L_{j,t}$ denote the weak learner on site $j$ at iteration $t$. $L_{j,t}$ are exchanged among all sites, and each site aggregates $L_{j,t}$ to build a hypothesis $h_{j,t}$. The local weight is updated based on the performance of $h_{j,t}$. The updated local weight is then used to update the global distribution of each site. However it is expensive to exchange local weight among sites. The solution proposed here is to broadcast only the sum of the local distribution from every local site to every other site. After $T$ iterations, all the hypothesis $h_{j,t}$ is combined to form a final hypothesis. So far, the work on distributed boosting only handles homogeneously distributed data while heterogeneously distributed data remains a challenge.

Another important ensembling technique is distributed meta-learning which takes the advantage of the inherent parallelism and distributed nature of meta-learning. The extension of meta-learning to distributed data is pretty straightforward as base classifiers are trained using local data. Prodromidis et al[10] described the distributed meta-learning framework and their implementation. Java Agents for Meta-learning (JAM). The big part of their work is identifying practical issues of distributed learners such as scalability, efficiency, portability and compatibility and explaining how these issues were addressed in JAM. In this work, meta-learning is defined recursively as collections of classifiers structured in multilevel trees, and pruning methods were used to avoid the efficiency issue. Three composition rules were suggested to generate the Meta learning data. As stated earlier, ensemble learning has advantages in distributed environments. However, the base classifiers are trained on local data and no information about their global performance is available. This introduces issues while combining the predications of base classifiers. While it is possible to get superior classification accuracy compared to single classifiers for both boosting and meta-learning as ensembling approaches, the final classifier can hardly be translated to knowledge base.

The difficulty of transforming the combined classifier to knowledge was addressed in the work of Park et al[11]. The contribution of [11] is twofold: (1) efficient decision tree
induction in heterogeneous environments and (2) combination
of local decision trees in the form of a single tree which
can be easily translated to rules. The idea of decision tree
induction in heterogeneous environment classifies the good
instances locally using available features, while the bad in-
stances are classified using cross-site features. The goodness
of instances are assessed using their local classification
confidence. Bad instances are gathered at the central site
to train the decision tree using all features. At local sites,
decision trees are trained by AdaBoost[12] used with
C4.5[5]. Take a binary classification of an instance for
example, the confidence of predicting 0 and 1 at site \( S_i \)
is respectively defined by
\[
\begin{align*}
c_i^0 &= \sum_{2p_{ij} = 0} a_i 2p_{ij} - 1 \quad (1) \\
c_i^1 &= \sum_{2p_{ij} = 1} a_i 2p_{ij} - 1 \quad (2)
\end{align*}
\]
where \( a_i \) is the boosting weight and \( p_{ij} \) is the predication
given by the \( j \)th weak decision tree of \( S_i \). The confidence
of the final predication at \( S_i \) is defined by \( c_i = |c_i^0 - c_i^1| \).
Intuitively, the smaller \( c_i \) is, the more difficult that \( x \) can
be correctly classified. If \( c_i < \gamma_1 \) for all site \( i \) and a pre-
defined threshold \( \gamma_1 \), the features of the instance will be
send to a central site. Regarding classifying unseen in-
stances, the work compared three techniques, weighted av-
erage, ordinary least-squares (OLS) scheme, and Tourn-
ament scheme, with Tournament scheme providing the best
predication accuracy. The Tournament scheme only con-
siders the predications of local sites with the highest con-
dence \( c = max_{i=1..K} c_i \). If \( c > \lambda_2 \), e.g. \( c \) is greater than
a predefined threshold, the unseen instance is assigned the
corresponding local classification label. Otherwise, the in-
stance is considered bad and classified in a centralized man-
er. The second part of this work is to preserve the simplic-
ity of decision tree. To achieve this, a Fourier analysis based
technique was used to combine multiple trees into a single
tree[15][14]. Without getting into too many mathematical
details, this procedure can be summarized in three steps: i) each
decision tree is considered as a function mapping dis-
crete feature domain to class label domain represented by
Fourier basis; ii) each site builds the Fourier representation
of the final classifier by computing as the weighted combi-
nation of Fourier representations of the weak decision
trees; iii) Fourier representations from local sites and the cen-
tral site are combined to induce a single decision tree from the
final Fourier representation. This approach only needs to
transfer Fourier representation among sites, which is com-
 pact requiring small communication overhead.

The work of Caragea et al[15][16] falls under second ap-
proach stated in the induction section. Their approach
generalizes the learning task into two components: (1) hypoth-
esis generation and (2) information extraction. The idea can
be formally expressed as \( L(D) = H(I(D)) \), where \( D, L, H, \)
and \( I \) respectively denote the data, the operators of learning,
hypothesis generation, and information extraction. This de-
composition enables the documented exact learning which
only extends the information extraction part to distributed
environment while keeping the hypothesis generation part
unchanged. The distributed exact learning method gener-
ates the same results as its centralized counterpart, which is
achieved when the following condition holds true:
\[
C(I_1(D_1), I_2(D_2), ... I_k(D_k)) = I(D). \quad (3)
\]
Eq.3 is the condition that shows the information extracted
from distributed data can be combined to form the same
information that can be extracted from the gathered data
otherwise. Of course, Eq.3 should not require migration of the raw data. Caragea suggested distributed information
extraction based on sufficient statistics. He further demon-
strated applicability of his approach on decision tree induc-
tion for both horizontally and vertically partitioned data.
The sufficient statistics for the decision tree induction based
on information gain is collected by counting number of in-
stances belonging to different classes. A slight extension of
Caragea’s work is also found in [15], where he addressed
the issue of counting examples from heterogeneous and au-
tonomous data sources by the query system INDUS[17].
The exact learning can be applied in both homogeneous
and heterogeneous environment with ease of implementa-
tion as only one part of the framework needs to be changed
for different learners. The technique based on counting is
more suitable for discretized data sets as continuous fea-
tures may lose information in discretization processing.

Giamnella et al[18] proposed a method for decision tree
induction for heterogeneous environment. Similar to the
work in [15], their algorithm is based on sufficient statistics,
particularly, counting. The main point of this work is to
reduce the coordination cost in counting the number of
instances of a tree node when the node assignment is not
available at a local site. They formulated counting as dot
product of two vectors. Take two sites, \( A \) and \( B \), for exam-
ple, the schema corresponding to a tree node is denoted as
\( \{x_A^1 = v_A^1, \ldots, x_A^k = v_A^k, x_B^1 = v_B^1, \ldots, x_B^k = v_B^k\} \), where \( x_A \) and
\( x_B \) are features on site \( A \) and \( B \) respectively. The number of
instances, denoted by \( D \), of this tree node is computed by
the dot product \( a \times b \) where \( a \) is a 0-1 vector from site \( A \) in-
dicating those rows satisfying \( \{x_A^1 = v_A^1, \ldots, x_A^k = v_A^k\} \), similar
is \( b \). Based on this notation, one approximation is used to
reduce the communication cost. Assuming site \( A \) has \( m \in-
stances(so \ does \ site \ B) \), site \( A \) and \( B \) cooperatively generate
\( k \times m \) random matrix \( R \) independently and identically from
a zero mean and unit variance distribution, where \( k \ll m \).
Site \( A \) and \( B \) exchange \( Ra \) and \( Rb \), and \( D \) is approximated by
\[
\frac{\sum_{i=1}^{k} v_B^i Rb}{k}.
\]
Note that the vectors exchanged is \( k \times 1 \) instead of \( m \times 1 \), which reduces the bandwidth required. The infor-
mation loss caused by the approximation is also analyzed. In a similar work from Baik et al.[19] suggested a different method for reducing the intra node communication. He proposed encoding of node assignment bit vectors. Both these approaches are more suitable for categorical featured data.

Bar-Or et al.[20] suggested a distributed decision tree algorithm for data sites organized in a hierarchical structure. The hierarchical structure usually corresponds to structure of management or abstraction levels. In such structure, statistics information is moved through a path from the leaf data site to the root data site. The root is responsible for building the decision tree. Their algorithm is based on building a feature cross table for computing information gain. The lower and upper bounds of information gain for a data set are defined in a recursive manner, defined as:

\[ G(P) \leq \frac{\sum_{i=1}^{k} n_i G(P_i)}{\sum_{i=1}^{k} n_i} \]

\[ G(P) \geq \frac{G(P_1)}{[1 + \frac{n_1}{n_1}][1 + \frac{n_2}{\min(n_1, n_2)}]} \]

Eq.4 holds when the data \( P = \bigcup [P_i | i = 1..k], P_i \cap P_j = \emptyset, i \neq j \). The cross-tables collected from lower levels are combined to higher level. Eq.5 holds when \( P = P_1 \cup P_2 \), and \( P_1 \) is partitioned into \( P_1^{left} \) and \( P_1^{right} \) by splitting candidates. Eq.4 and Eq.5 is used to filter out attributes whose gain cannot be large enough to change the result, hence reducing the size of the cross table needed to be transferred. This approach is well suitable for high dimension data in a homogeneous environment, provided that the correlations are sparse.

Our work presented in this paper is one distributed exact learning algorithm based on \( \chi^2 \) dependence test. Based on the nature of \( \chi^2 \) test, the generated decision tree can have more than two branches at each node. Multisplit trees usually have smaller sizes compared to binary trees. Since \( \chi^2 \) test is designed for categorical data, we further propose quantization method for the continuous data in a distributed setting.

3 Distributed CHAID

3.1 CHAID and Chi-Square Independence Test

The original CHAID algorithm was proposed in[7] where the CHAID tree was inducted following the standard top-down recursive split tree induction framework[21]. CHAID differs from other decision tree induction methods in that it evaluates splitting feature candidates by using the \( \chi^2 \) independence test at each split of the tree node.

The \( \chi^2 \) independence tests the null hypothesis that two variables, say \( V_1 \) and \( V_2 \), are independent. The test is performed on the contingency table(cross-table) of \( V_1 \) and \( V_2 \). A cross-table is a two dimensional table where each row and column corresponds to a distinct category of \( V_1 \) and \( V_2 \) respectively. Given a cross-table \( A_{c2} \) of \( c_1 \) rows and \( c_2 \) columns, the value of cell \( a_{i,j} \), denoted by \( O_{i,j} \), is the number of observations with \( V_1 = C(A_i) \) and \( V_2 = C(A_j) \), where \( C() \) is an operator which returns the category represented by rows(\( A_i \)) or columns(\( A_j \)) of cross-table A.

The \( \chi^2 \) value is then calculated using:

\[ \chi^2 = \sum_{i,j} \frac{(O_{i,j} - E_{i,j})^2}{E_{i,j}} \]

where \( E_{i,j} \) is the expectation value of cell \( a_{i,j} \) under the null hypothesis. For computing \( E_{i,j} \), refer to any statistics textbooks including \( \chi^2 \) test for detail. Intuitively, Eq.6 measures how far the actual observations are from the expectations. The values of Eq.6 follow a \( \chi^2 \) distribution with degree of freedom(\( df \)) of \((c_1 - 1)(c_2 - 1)\). The significance value \( p \) of a \( \chi^2 \) value is calculated by \( p = 1 - \chi^2 cdf(\chi^2, df) \), where \( \chi^2 cdf(\chi^2, df) \) returns the value of \( \chi^2 \) under the \( \chi^2 \) cumulative distribution of degree of freedom equals \( df \). \( p \) measures the possibility of the actual observations corresponding to a certain \( \chi^2 \) values or greater under the independence hypothesis. With a defined threshold \( \lambda \), if \( p < \lambda \), the null hypothesis is rejected and the \( V_1 \) and \( V_2 \) are considered dependent.

The biggest strength of CHAID algorithm is its ability to handle nominal, ordinal and missing values. Nominal values are order-less values such as zip codes while the ordinal values has order, i.e. discrete gradation medals such as gold, silver and bronze representing the position. Missing values are treated as a particular data type. Note that the original work used different terms: monotonic for ordinal, free for nominal, and floating for missing values. We will use the terms monotonic, free, and floating in the rest of this paper. The data type determines the manner in which the merge and split operation(see algorithm1) can be carried out. Only consecutive monotonic categories can be merged while any two free categories can be merged together. The floating data can be merged with any non-floating categories. The procedure of evaluating the splitting feature can be summarized as follow(refer to [7] for detailed description):

...
Algorithm 1 CHAID Algorithm
1. Cross-tabulate every predictor and the dependent variable, and do
   1.1. Find the allowable pair of categories of the predictor whose $2 \times d$ (d is the number of classes) sub-table is least significantly different. If this significance does not reach a threshold $\lambda_1$, the pair of predictor categories are combined. This step is repeated until no insignificantly different pairs can be found.
   1.2. For each combined category composed of three or more original categories, find the allowable binary split that generates two sub (combined) categories which are most significantly different. If the significance reaches the threshold $\lambda_2$, split the combined category and repeat step 1.1.
   2. For all the splitting feature candidates, calculate the significance of $\chi^2$ independence test. If the greatest significance reaches the threshold $\lambda_3$, the corresponding predictor is selected as the splitting feature.
3. Current dataset is partitioned based on the selected splitting feature and its combined categories. The subsets are recursively processed by the same procedure starting from step 1.

The difference between two predictor categories is measured by $\chi^2$ goodness-of-fit test which has the same form as Eq.6. The smaller $p$ is, the more significantly different are the two predictor categories. One issue that is not explicitly described in the original CHAID algorithm is the stopping criteria for the tree, which we will address in the later section.

3.2 CHAID in Distributed Environments

The distributed CHAID algorithm follows the standard decision tree induction framework. It differs from the original (centralized) CHAID by which the splitting feature evaluation, algorithm1, is performed in a distributed settings. For heterogeneous distributions, assume $n$ observations containing $m$ feature dimensions is vertically partitioned and distributed among many local sites. The extension of original CHAID is described in the pseudo code2.

Algorithm 2 CHAID Algorithm in Heterogeneous Environment
1. Local site $S_k$ performs algorithm1 and calculates $p_k$ for the most dependent local feature, $f_k$.
2. Master site collects $p_k$, chooses the greatest one $p_j$ and notices the local site $S_j$.
3. Site $S_j$ splits its data based on the combined categories of $f_j$ and broadcasts the subsets assignment $I_j$ to all other local sites, which accordingly split their local data.

For a setting of $K$ sites, the communication cost is measured to be $O(K^2)$ by calculating the cells:

$$CT_{i,j} = \sum_k CT^{k,i}_a, C(C(T^{k,i}_a) = C(C(T^{k,i}_a) \land C(C(T^{k,i}_j) = C(C(T^{k,i}_a)).$$

Algorithm 3 CHAID Algorithm in Homogeneous Environment
1. Every local site $S_k$ cross-tabulated every predictor $l$ with the dependent variable and sent all local cross-tables $CT^{K,l}_k$ to the master site along with the row and column category labels.
2. The master site forms the global cross-table $CT^l$ for feature $l$ by calculating the cells:

$$CT^{l}_{i,j} = \sum_k CT^{k,l}_{a,b}, C(C(T^{k,l}_a) = C(C(T^{k,l}_a) \land C(C(T^{k,l}_b) = C(C(T^{k,l}_a)).$$

3. Master site performs steps from 1.1 to 3 of algorithm1.
4. Master site broadcasts the splitting feature and its combined groups to all local sites which then accordingly split the local data.

For a setting of $K$ sites, the communication cost in this case is $O(KMN)$, where $M$ is the total number of distinct categories of all features, $N$ is the number of classes. For a grown tree of $h$ nodes, the total communication cost is measured to be $O(KMNh)$ which arises from collecting local cross-tables. The communication cost for growing a tree depends on the number of nodes the final tree has, which in most cases is very small compared to the cost of transferring data between nodes.

Unlike the tree induction using Gini or Entropy, the merge and split operations usually generate two or more branches out of one tree node. This multisplit characteristic of CHAID provides additional parallel processing advantage over the binary split trees where the tree branches can be processed concurrently, further reducing the training time.

One limitation of the $\chi^2$ test is its inability to process continuous feature data. To make our algorithm complete, we need to perform distributed quantization for continuous data as a preprocessing step. Adaptive partition, which tries to get equal number of observations in every bin, is usually better than equal span partition when dependence between two variables needs to be calculated[22]. The adaptive partition recursively splits the dataset into two subsets using the median points as pivots. We adopt the distributed median point estimation proposed in[23]. The idea of that work is to approximate the global dataset based on the representa-
3.3 Tunable Parameters of CHAID

The original CHAID offers no discussion in this area. However the statistical hypothesis test is sensitive to significance threshold. In addition, the quantization process also loses information. This can result in poor tree induction with lower performance. Here we suggest parameters that can be used to tune the performance of the final tree. The impact of our tuning is discussed in the experiment section.

The original CHAID algorithm performs merge and split iterations on all features in order to determine the splitting feature. This could be computationally intensive if the features have many categories or the feature dimensionality is large. One alternative to this approach is to calculate the dependence between each predictor and the dependent variable using the original categories and select the predictor with the most significance as the splitting feature. Once the splitting feature is picked, the merge and split operations are performed only on this feature to determine the combined categories. To simplify expression, we call the modified version pre-selection method while the original version post-selection method. The computation cost of pre-selection is \( n_m \) of the cost of post-selection, where \( m \) is the number of predictors.

The merge and split operations in the original CHAID use fixed significance thresholds for all level of the tree. A tree node is stopped from splitting if no merge can be done by the fixed merge threshold. We consider the merge threshold as a tunable parameter because the model of the data can be complex or simple. For simple models, the trees can achieve good performance while retaining low tree depth. For complex models, a shallow tree means a prematurely terminated tree which may provide inferior accuracy. We set the merge threshold as a function of the tree level. The function can be expressed as:

\[
MT = 0.0001 + (NL - 1)^2 \times 0.001,
\]

where \( MT \) denotes merge threshold where as \( NL \) denotes the node level. The function sets a small significance level for merge at lower level tree nodes so that the number of branches at lower level is small, which in turn preserves the correlation between predictors and the dependent variable in the sub-tree nodes. Also this function sets a high significance level for higher level tree nodes to keep the trees from growing too tall. We use adaptive merge threshold (AMT) to denote the modified version and fixed merge threshold (FMT) to denote the original method. Same as the merge threshold, the split threshold is also adaptive, which is set to be \( MT \times 0.8 \).

Our third modification is based on the observation that the leaves in the original CHAID trees usually have high degree of class mixture which results in low testing accuracy. This is due to the categorical predictors values have multiple class labels with low dominance. Hence we connect the merge threshold of leaves to their class dominance. E.g. if a tree node is considered a leaf and the class dominance is lower than a threshold, merge threshold is increased by \( NL^2 \times 0.001 \) so that this node can be further split. As a summary, a tree node is considered as a leaf if any of the following conditions is true:

- The node size is small.
- The class dominance reaches the threshold.
- No split can be found with all possible merge thresholds for all predictors.

To distinguish, we refer this method as class dominance (CD) based while the original method is referred as merge threshold (MT) based. The impact of these parameters in discussed in the experiment section.

4 Experimental Results

To assess the performance of our distributed CHAID algorithm, we compared its accuracy against a centrally build decision tree. The experiments are conducted over 13 UCI datasets. The data sizes range from couple hundred to several thousands. The class numbers of these datasets range...
from 2 to 7. In addition, the complexity of the datasets varies as the accuracy of some popular decision tree algorithms on them ranges from around 50% to around 90%. All accuracy reported is obtained by two runs of five folds cross validation on each dataset. Since the experimental results for heterogeneous environment is theoretically proved to be exactly the same as the centralized settings, we only report the experiments done for homogeneous settings. To simulate a homogeneous distribution of 3 sites, we first permuted the data instances and then partitioned it in 3 random sets. Each set is then distributed to three virtual processing sites. Note that our distributed CHAID algorithm is only compared to its centralized counterpart as we are not proposing a new decision tree induction method. The comparison of CHAID with other popular decision tree induction methods can be found in other literatures, for example [24].

The first experiment tests the impact of adaptive merge threshold on the accuracy performance. This experiment is performed on each dataset for class dominance. The impact is represented by ratio of accuracy difference against the accuracy obtained by using the fixed merge threshold, e.g. $\frac{\text{accuracy(AMT)} - \text{accuracy(FMT)}}{\text{accuracy(FMT)}}$. The results are shown in Fig.1 and Fig.2. Fig.1 shows the results of experiments done using the post-selection. Each line shows the results corresponding to the class dominance of stop criteria set to different values. Similarly, Fig.2 shows results of experiments done using the pre-selection. These two figures show some interesting observations. First, for both pre-selection and post-selection methods, the AMT boosts the accuracy performance for most datasets especially when the class dominance for stop is set to high values (0.8, 0.9). In some cases the performance gain reaches 20% to 40%. Second, the lines in Fig.1 follow a much obvious pattern than the lines in Fig.2. This can be explained by the fact that the post-selection method has more options for the splitting feature than the pre-selection method. The post-selection method can select the most correlated feature to the dependent variable after merge and split operations, while the pre-selection has only one option of the splitting feature which is determined before merge and split. This feature may not be the best choice for split which in turn results premature or overfitting trees. This has more impact for high class dominance. Hence the pattern in Fig.2 is pretty random. The same reason may explain why the performance difference is bigger for post-selection method than pre-selection method.

A visual inspection shows that when the class dominance is 0.8, we had the performance gain for most datasets by using the adaptive merge threshold. Hence we think 0.8 is a reasonable value for the class dominance parameter.

The second experiment tests performance difference resulted by using pre-selection against post-selection, which is represented by the ratio of accuracy difference against the accuracy obtained by using the post-selection, e.g. $\frac{\text{accuracy(Pre)}}{\text{accuracy(Post)}}$. Fig.3 shows the mean accuracy change obtained when adaptive merge threshold is used and class dominance is set for different values. We can observe

<table>
<thead>
<tr>
<th>CD</th>
<th>Mean Accuracy</th>
<th>Accuracy Change (Adaptive vs. Fixed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>0.2</td>
<td>-0.1</td>
</tr>
<tr>
<td>0.7</td>
<td>0.4</td>
<td>-0.2</td>
</tr>
<tr>
<td>0.8</td>
<td>0.6</td>
<td>-0.3</td>
</tr>
</tbody>
</table>

Figure 1. Fixed vs Adaptive Merge Threshold(post-selection)

Figure 2. Fixed vs Adaptive Merge Threshold(pre-selection)

Figure 3. Accuracy of Pre-selection vs Post-selection
the accuracy of this two splitting feature selection schema is similar with one out performs the other one for different datasets. However, the pre-selection schema is preferred to the post-selection schema as it saves a substantial portion of computation time.

The third experiment compares the performance of the distributed version with the centralized version of CHAID. The result is shown in Table 1. The data types are in short forms: M-Monotonic, F-Free, R-Real. As an initial work, we didn’t include floating type in our experiments. The columns titled "Cen" in Table 1 is the accuracy of the original CHAID without tunable parameters. The columns titled "Dis(pre)" and "Dis(post)" present accuracy of our distributed CHAID with class dominance set to be 0.8 and 0.2 respectively. It is shown that the distributed CHAID without tunable parameters, our extension can reimburse the information loss caused in the quantization process in distributed environment to achieve similar or better performance than its centralized version.

5 Conclusion

In this paper, we proposed an extension of the original CHAID algorithm to distributed environment. We extended the algorithm, which was designed for categorical data, for continuous data. This is accomplished by extending adaptive quantization algorithm to the distributed settings. Our extension can be theoretically proved to get the exact results as its centralized counterpart. We also proposed tunable parameters for the distributed CHAID algorithm. Our experimental results confirm that these parameters can reimbursed the information loss caused in the quantization process in distributed environment to achieve similar or better performance than its centralized version.

Table 1. Performance of Centralized and Distributed CHAID

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classes</th>
<th>Instances</th>
<th>Data Type</th>
<th>Cen</th>
<th>Dis(pre)</th>
<th>Dis(post)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast_Cancer</td>
<td>2</td>
<td>286</td>
<td>M,F</td>
<td>75.45±2.60</td>
<td>74.74±1.16</td>
<td>75.46±1.47</td>
</tr>
<tr>
<td>Breast_Cancer(cancer)</td>
<td>2</td>
<td>603</td>
<td>M</td>
<td>94.15±2.12</td>
<td>90.76±2.20</td>
<td>92.68±0.87</td>
</tr>
<tr>
<td>Bank</td>
<td>2</td>
<td>345</td>
<td>R</td>
<td>61.74±6.16</td>
<td>68.07±3.71</td>
<td>60.29±5.52</td>
</tr>
<tr>
<td>Car</td>
<td>2</td>
<td>1728</td>
<td>M</td>
<td>95.43±1.35</td>
<td>92.19±3.36</td>
<td>90.92±3.32</td>
</tr>
<tr>
<td>Dermatology</td>
<td>6</td>
<td>358</td>
<td>M,R,F</td>
<td>87.43±6.54</td>
<td>91.63±2.43</td>
<td>87.72±5.87</td>
</tr>
<tr>
<td>Glass</td>
<td>7</td>
<td>214</td>
<td>R</td>
<td>50.44±7.09</td>
<td>64.50±6.53</td>
<td>56.29±8.54</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>2</td>
<td>351</td>
<td>M,R</td>
<td>79.74±3.12</td>
<td>83.03±6.68</td>
<td>82.63±2.03</td>
</tr>
<tr>
<td>Iris</td>
<td>2</td>
<td>150</td>
<td>R</td>
<td>86.67±4.97</td>
<td>91.34±5.02</td>
<td>83.04±2.22</td>
</tr>
<tr>
<td>Monks</td>
<td>2</td>
<td>432</td>
<td>F</td>
<td>74.04±4.63</td>
<td>79.04±3.25</td>
<td>74.94±3.04</td>
</tr>
<tr>
<td>Segment</td>
<td>7</td>
<td>2310</td>
<td>R</td>
<td>92.38±2.37</td>
<td>86.84±1.28</td>
<td>86.65±2.18</td>
</tr>
<tr>
<td>Vote</td>
<td>2</td>
<td>455</td>
<td>F</td>
<td>94.94±2.55</td>
<td>86.21±3.54</td>
<td>86.67±3.12</td>
</tr>
<tr>
<td>Wine</td>
<td>3</td>
<td>178</td>
<td>R</td>
<td>79.94±6.2</td>
<td>87.37±3.01</td>
<td>83.01±2.71</td>
</tr>
<tr>
<td>Zoo</td>
<td>7</td>
<td>108</td>
<td>M,F</td>
<td>88.05±6.56</td>
<td>79.14±11.34</td>
<td>82.05±10.39</td>
</tr>
</tbody>
</table>

References


