ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012

A Comparison of Arbiter and Combiner Trees

Dr.M. Usha Rani, G.T. Prasanna Kumari

Abstract--- In data mining, there is no learning algorithm which attains the highest accuracy on any dataset. Arbiter and Combiner as techniques to integrate classifiers induced from partitioned data, having as optimization criterion the accuracy of a given dataset. We have explored an approach called meta-learning that is related to the traditional approaches of data reduction commonly employed in distributed query processing systems. Here we seek efficient means to learn how to combine a number of base classifiers, which are learned from subsets of the data, so that we scale efficiently to larger learning problems, and boost the accuracy of the constituent classifiers if possible. In this paper we compare the arbiter tree strategy to a new but related approach called the combiner tree strategy. The one-level meta-learning learning techniques may not produce highly accurate classifiers. Here, we explore hierarchical techniques by applying meta-learning strategies recursively.

Index Terms--- **Arbiter, Combiner, Hierarchical Meta-learning Techniques.**

I. INTRODUCTION

Financial institutions and market analysis firms have for years attempted to learn simple categorical classifications of their potential customer base, i.e., relevant patterns of attribute values of consumer data that predict a low-risk (high profit) customer versus a high-risk (low profit) customer. Many corporations seeking similar added value from their databases are already dealing with overwhelming amounts of global information that in time will likely grow in size faster than available improvements in machine resources. Furthermore, many existing learning algorithms require all the data to be resident in main memory, which is clearly untenable in many realistic databases. In certain cases, data are inherently distributed and cannot be localized on any one machine (even by a trusted third party) for competitive business reasons, as well as statutory constraints imposed by government. In such situations, it may not be possible, nor feasible, to inspect all of the data at one processing site to compute one primary "global" classifier.

Incremental learning algorithms and windowing techniques aim to solve the scaling problem by piecemeal processing of a large data set. Others have studied approaches based upon direct parallelization of a learning algorithm run on a multiprocessor. A review of such approaches has appeared elsewhere (Chan & Stolfo 1995). An alternative approach we study here is to apply *data reduction* techniques common in distributed query processing where cluster of computers can be profitably employed to learn from large databases. This means one may partition the data into a number of smaller *disjoint* training subsets, apply some learning algorithm on each subset (perhaps all in parallel), followed by a phase that combines the learned results in some principled fashion. In the case of inherently distributed databases, each constituent fixed partition constitutes the training set for one instance of a machine learning program that generates one distinct classifier (far smaller in size than the data). The classifiers so generated may be a distributed set of rules, a number of C programs (e.g. "black box" neural net programs), or a set of "intelligent agents" that may be readily exchanged between processors in a network. Notice, however, that as the size of the data set and the number of its partitions increase, the size of each partition relative to the entire database decreases. This implies that the accuracy of each base classifier will likely degrade. Thus, we also seek to boost the accuracy of the distinct classifiers by combining their collective knowledge. In this paper we study more sophisticated techniques for combining predictions generated by a set of *base classifiers*, each of which is computed by a learning algorithm applied to a distinct data subset.

II. META-LEARNING

Meta-learning (Chan & Stolfo, 1993b) is loosely defined as learning of meta-knowledge about learned knowledge. In our work we concentrate on learning from the output of concept learning systems. In this case meta-learning means learning from the predictions of these classifiers on common training data. A classifier (or concept) is the output of a concept learning system and a prediction (or class generated by a classifier when an instance is supplied. Thus, we are interested in the output of the classifiers, not the internal structure and strategies of the learning algorithms themselves. Moreover, in several of the schemes we define, the training data presented to the learning algorithms initially are also available to the meta-learner under certain circumstances. Figure 1 depicts the different stages in a simplified meta-learning scenario:

1. The classifiers (base classifiers) are trained from the initial (base-level) training sets.
2. Predictions are generated by the learned classifiers on the training sets.
3. A meta-level training set is composed from the predictions generated by the Classifiers.
4. The final classifier meta-classifier is trained from the meta-level training set.

In meta-learning, a learning algorithm is used to learn how to integrate the learned classifiers. That is, rather than having a predetermined and fixed integration rule (for example, voting), the integration rule is learned based on the behavior of the trained classifier.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012

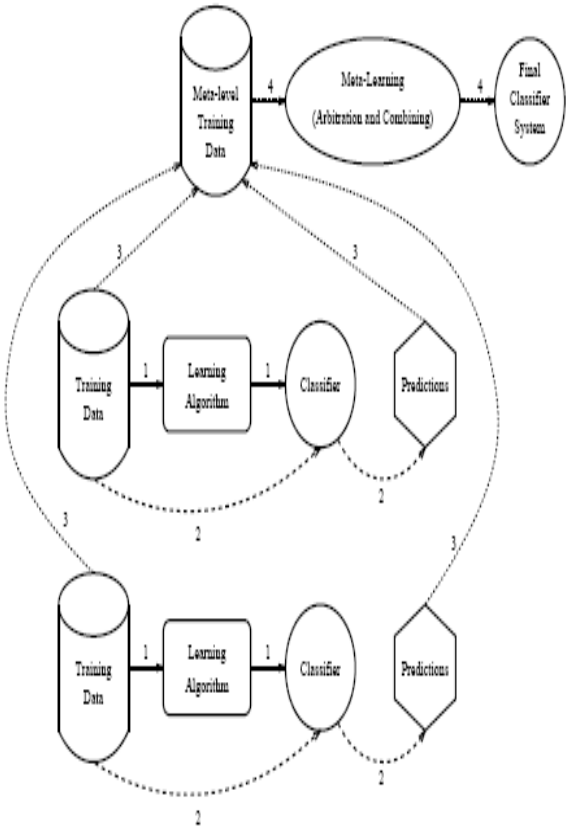

Fig 1: Meta-learning

III. META-LEARNING TECHNIQUES

Our approach is to Meta-learn a set of new classifiers (or meta-classifiers) whose training data are sets of predictions generated by a set of base classifiers. Arbiters and combiners are the two types of meta-classifiers studied here.

We distinguish between base classifiers and arbiters/combiners as follows. A base classifier is the outcome of applying a learning algorithm directly to "raw" training data. The base classifier is a program that given a test datum provides a prediction of its unknown class. An arbiter or combiner, as detailed below, is a program generated by a learning algorithm that is trained on the predictions produced by a set of base classifiers and sometimes the raw training data. The arbiter/combiner is also a classifier, and hence other arbiters or combiners can be computed from the set of predictions of other arbiters/combiners.

A. Arbiter

An arbiter (Chan&Stolfo, 1993d) is learned by some learning algorithm to arbitrate among predictions generated by different base classifiers. That is, its purpose is to provide an alternate and more educated prediction when the base classifiers present diverse predictions. This arbiter, together with an arbitration rule, decides a final classification outcome based upon the base predictions. In the arbiter strategy, the training set for the meta-learner is a subset of the training set for the base learners; i.e. the meta-level training instances are a particular distribution of the raw training set. The predictions of the learned base classifiers for the training set and a selection rule, which varies in different schemes, determines which subset will constitute the meta-learner's training set. (This contrasts with the combiner strategy, which has the same number of examples for the base classifier as for the combiner. Also, the meta-level training instances for a combiner incorporate additional information than just the raw training data.) Based on this training set, the meta-learner generates a meta-classifier, in this case called an arbiter. In classifying an instance, the base classifiers first generate their predictions. These predictions, together with the arbiter's prediction and a corresponding arbitration rule, generate the final prediction (see Figure 2). In this strategy one learns to arbitrate among the potentially different predictions from the base classifiers, instead of learning to coalesce the predictions as in the combiner strategy. The details of how the final decision is made follow. Let x be an instance whose classification we seek, $C_1(x), C_2(x), \ldots\ldots C_k(x)$ are the predicted classifications of x from k base classifiers, $C_1, C_2, \ldots C_k$, and A(x) is the classification of x predicted by the arbiter. One arbitration rule studied and reported here is as follows:

- Return the class with a plurality of votes in $C_1(x)$, $C_2(x)$ $\ldots C_k(x)$, and A(x), with preference given to the arbiter's choice in case of a tie.

We now detail how an arbiter is learned. The training set of an arbiter is generated in a way that it contains the raw training examples whose classifications the base classifiers cannot predict consistently. Formally, a training set T for the arbiter is generated by picking examples is dictated by a selection rule.

One version of a selection rule studied here is as follows:

An instance is selected if none of the classes in the k base predictions gathers a majority vote (> k/2 votes); i.e., $T = \{x \in E \mid \text{no majority } (C_1(x), C2(x), .C_k(x))\}$. The purpose of this rule is to choose examples that are confusing; i.e., the majority of classifiers do not agree. Figure2 presents a sample training set for the arbiter strategy. Once the training set is formed, an arbiter is generated by the same learning algorithm used to train the base classifiers. Together with an arbitration rule, the learned arbiter resolves conflicts among the classifiers when necessary.

B. Combiner

The aim of the combiner strategy (Chan & Stolfo, 1993a) is to coalesce the predictions from the base classifiers by learning the relationship between these predictions and the correct prediction. For example, a base classifier might consistently make the correct predictions for class c i.e., when this base classifier predicts class c, it is probably correct regardless of the predictions made by the other base classifiers. In the combiner strategy the predictions of the learned base

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012

classifiers on the training set form the basis of the meta-learner's training set. A composition rule, which varies in different schemes, determines the content of training examples for the meta-learner. From these examples, the meta-learner generates meta-classifiers, that we call a combiner. In classifying an instance, the base classifiers first generate their predictions. Based on the same composition rule, a new instance is generated from the predictions, which is then classified by the combiner (see Figure 2). We note that a combiner computes a prediction that may be entirely different from any proposed by a base classifier, whereas an arbiter chooses one of the predictions from the base classifiers and the arbiter itself.

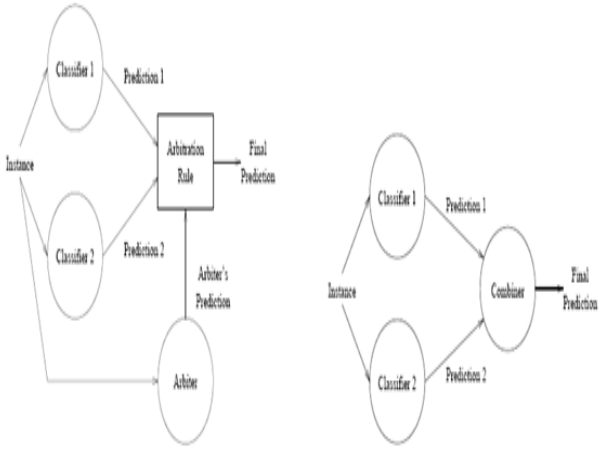

Fig 2: An Arbiter and A Combiner With Two Classifiers

We experimented with two schemes for the composition rule. First, the predictions, $C_1(x)$, $C_2(x)...C_k(x)$, for each example x in the validation set of examples, E, are generated by the k base classifiers. These predicted classifications are used to form a new set of "meta-level training instances", T, which is used as input to a learning algorithm that computes a combiner. The manner in which T is computed varies as defined below. In the following definitions, class(x) and attribute_vector(x) denote the correct classification and attribute vector of example x as specified in the validation set, E.

- Return meta-level training instances with the correct classification and the predictions; i.e., T = {(class(x), $C_1(x)$, $C_2(x)$... Ck(x)) | x € E}. This scheme was also used by Wolpert (1992). (This scheme is denoted as class-combiner)
- Return meta-level training instances as in class-combiner with the addition of the attribute vectors; i.e., T = {(class(x), $C_1(x)$, $C_2(x)$,….$C_k(x)$, attribute_vector(x)) | x € E}. (This scheme is denoted as class-attribute-combiner)

Note the difference in training data. Arbiters are computed from a distinguished and biased subset of data selected from the input database used to train the base classifiers. Combiners, however, are trained on the predicted classifications of that data generated by the base classifiers, as well as the data itself.

C. Issues

Several issues arise from our meta-learning strategies and are detailed as follows:

(1) Number and size of training subsets: The number of initially partitioned training data subsets largely depends on the number of processor available, the inherent distribution of data across multiple platforms(some possibly mobile and periodically disconnected), the total size of the available training set, and the complexity of the learning algorithms. The available resources at each processing sites naturally defines an upper bound on the size of each subset. If the number of subsets exceeds the number of processors available, each processor can simulate the work of multiple one by serially executing the task of each processor. Another consideration is the desired accuracy we wish to achieve. As we will see in our experimental results, there may be a trade-off between the number of subsets and the final accuracy of a meta-learning system. Moreover, the size of each subset cannot be too small because sufficient data must be available for each learning process to produce an effective base classifier in the initial stage of training.

(2)Distribution of examples disjoint or replicated: Since a totally random distribution of examples may result in the absence of one or more classes in the partitioned data subsets, the classifiers formed from those subsets will be ignorant about those classes. That is, more "disagreements" may occur between classifier, which leads to larger arbiter training sets. Maintaining the class distribution in each subset as in the total available training set may alleviate this problem. The classifiers generated from these subsets, may be closer in behaviour to the global classifier produced from the entire training set than those trained on random class distributions. In addition, disjoint data subsets promote the maximum amount of parallelism and hence are more desirable. Yet partial replication may mitigate the problem of extreme bias potentially introduced by disjoint data.

(3)Strategies: There are indeed many strategies for arbitration and combining as detailed here, each impacting the size of training data required to implement them effectively. Several experiments were run to determine the relative effectiveness of some of these strategies. They vary in the type of information or biased distributions of training data the arbiter is allowed to see. Thus far, the meta-learning strategies we discussed are applied solely to a single collection of base classifiers. These are called "one-level" meta-learners. We also studied building hierarchical structures in a recursive fashion, i.e., meta-learning arbiters and combiners from a collection of "lower level" arbiters and combiners. These hierarchical classifiers attempt to improve the prediction

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012

accuracy that may be achieved by one-level meta-learned classifiers.

IV. HIERARCHICAL META-LEARNING TECHNIQUES

The one-level meta-learning learning techniques may not produce highly accurate classifiers. Here, we explore hierarchical techniques by applying meta-learning strategies recursively.

A. Arbiter Trees

An arbiter tree is a hierarchical structure composed of arbiters that are computed in a bottom-up, binary-tree fashion. An arbiter is initially learned from the output of a pair of base classifiers and recursively, an arbiter is learned from the output of two arbiters. For k subsets and k classifiers, there are log2 (k) levels generated.

When an instance is classified by the arbiter tree, predictions flow from the leaves to the root. First, each of the leaf classifiers produces an initial classification of the test instance. From a pair of predictions and the parent arbiter's prediction, another prediction is produced by an arbitration rule. This process is applied at each level until a final prediction is produced at the root of the tree. We now proceed to describe how to build an arbiter tree in detail.

Suppose there are initially four training data subsets (T1 – T4), processed by some learning algorithm, L. First, four classifiers (C1 – C4) are generated from four instances of L applied to T1 – T4. The union of the subsets T1 and T2, U12, is then classified by C1 and C2, which generates two sets of predictions, P1 and P2. A selection rule as detailed earlier generates a training set (T12) for the arbiter from the predictions P1 and P2, and the subset U12. The arbiter (A12) is then trained from the set T12 by algorithm L. Similarly, arbiter A34 is generated from T3 and T4 and hence all the first-level arbiters are produced. Then U14 is formed by the union of subsets T1 through T4 and is classified by the arbiter trees rooted with A12 and A34. Similarly, T14 and A14 (root arbiter) are generated and the arbiter tree is complete. The resultant tree is depicted in Figure 3.

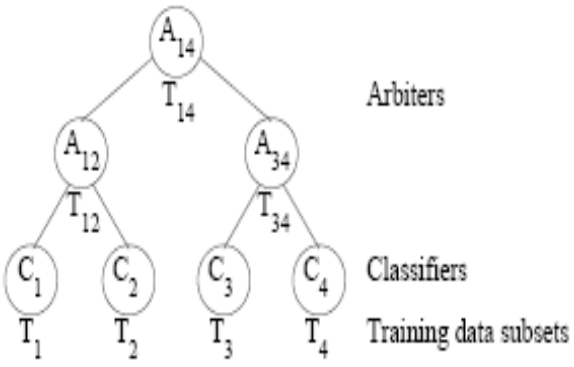

Fig 3: A Sample Arbiter tree

This process can be generalized to arbiter trees of higher order. The higher the order is, the shallower the tree becomes. In a parallel environment this translates to faster execution. However, there will logically be an increase in the number of disagreements (and hence data items selected for training) and higher communication overhead at each level in the tree due to the arbitration of many more predictions at a single arbitration site.

In a distributed computing environment, the union sets need not be formed at one processing site. Rather, we can classify each subset by transmitting each learned classifier to each site which is used to scan the local data set that is labelled with the classifier's predictions. Each classifier is a computational object far smaller in size than the training sets from which they are derived. For example, in a network computing environment each classifier may be encapsulated as an "agent" that is communicated among sites.

Since an arbiter training set is constructed from the results of the arbiter's two subtrees, each node in the arbiter tree is a synchronization point. That is, arbitrary subtrees can be run asynchronously with no communication until a pair of subtrees joins at the same parent. The time to learn an arbiter tree is proportional to the longest path in the tree, which is bounded by the path with the most training data. To reduce the complexity of learning arbiter trees, the size of the training sets for arbiters is purposefully restricted to be no larger than the training sets used to compute base classifiers. Thus, the parallel processing time at each level of the tree is relatively equal throughout the tree. However, in several of our experiments, this restriction on the allowable size of the training sets for arbiters was removed to explore two key issues: whether higher accuracy could be achieved by providing more information for each arbiter, and what might be the number of disagreements so generated, and hence the size of training data that would naturally be formed by our selection rules.

Notice that the maximum training set size doubles as one moves up one level in the tree and is equal to the size of the entire training set when the root is reached. Obviously, we do not desire forming a training set at the root as large as the original training set. Indeed, meta-learning in this case is of no use, and at great expense. Therefore, we derive a means to control the size of the arbiter training sets as we move up the tree without a significant reduction in accuracy of the final result.

Since the training sets selected at an arbiter node depends on the classification results from the two descendant sub trees during run time, the configuration of an arbiter tree cannot be optimized during compile time. The size of these set (i.e., the number of disagreements) is not known until the base classifiers are first computed. However, we may optimize the configuration of a tree during run time by clever paring of classifiers. The configuration of the resulting tree depends upon the manner in which the classifiers and arbiters are paired

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012

and ordered at each level. Our goal here is to devise a pairing strategy that favours smaller training sets near the root.

Our strategy we may consider is to pair the classifiers and arbiters at each level that would produce the fewest disagreements and hence the smallest arbiter training sets (denoted as min-size). Another possible strategy is to pair those classifiers that produce the highest number of disagreements (max-size). At first glance the first strategy would seem to be more attractive. However, if the disagreements between classifiers are not resolved at the bottom of the tree, the data that are not commonly classified will surface near the root of the tree, which is also where there are fewer choices of pairings of classifiers to control the growth of the training sets. Hence, it may be advantageous to resolve the disagreements near the leaves producing fewer disagreements near the root. That is, it may be more desirable to pair classifiers and arbiters that produce the largest sets lower in the tree, which is perhaps counter intuitive. These sophisticated pairing schemes might decrease the arbiter training set size, but they might also increase the communication overhead in a distributed computing environment. They also create synchronization points at each level, instead of at each node when no special parings are performed. A compromise strategy might be to perform pairing only at the leaf level. This indirectly affects the subsequent training sets at each level, but synchronization occurs only at each node and not at each level.

B. Combiner Trees

The way combiner trees are learned and used is very similar to arbiter trees. A combiner tree is trained bottom-up. A combiner, instead of an arbiter, is computed at each non-leaf node of a combiner tree. To classify an instance, each of the leaf classifiers produces an initial prediction. From a pair of predictions, the composition rule is used to generate a meta-level instance, which is then classified by the parent combiner. This process is applied at each level until a final prediction is produced at the root of the tree.

Another significant departure from arbiter trees is that for combiner trees, a random set of examples (a validation set) is selected at each level of learning in generating a combiner tree instead of choosing a set from the union of the underlying data subsets. Before learning commences, a random set of examples is picked from the underlying subsets for each level of the combiner tree. To ensure efficient processing, the size of these random training sets is limited to the size of the initial subsets used to train base classifiers. Base classifiers are learned at the leaf level from disjoint training data. Each pair of base classifiers produce predictions for the random training set at the first level. Following the composition rule, a meta-level training set is generated from the predictions and training examples. A combiner is then learned from the meta-level training set by applying a learning algorithm. This process is repeated at each level until the root combiner is created. Again, in a network computing environment classifiers may be represented as remote agent processes to distribute the meta-learning process.

The arbiter and combiner tree strategies have different impact on efficiency. The arbiter tree approach we have implemented requires the classification of, possible, the entire data set at the root level. Significant speed up might not be easily obtained. The combiner tree approach, however, always classifies a set of data that is bounded by the size of a relatively small validation set. Therefore, combiner trees can be generated more efficiently than arbiter trees. However, it remains to be seen what impact on accuracy either scheme may exhibit.

V. CONCLUSION

In the arbiter approach, an arbiter is meta-learned from the predictions of the local data mining algorithm. The arbiter accepts as input a data set containing "confusing" tuples, i.e., tuples whose value of the goal attribute is predicted in an inconsistent manner by the different local data mining algorithms. Note that an arbiter learns to choose among the conflicting predictions made by different local data mining algorithms, while a combiner can make a prediction completely different from the predictions made by any local data mining algorithm. Once an arbiter is learned, the final prediction is determined by taking into account the prediction made by the local data mining algorithms and the prediction made by the arbiter. These predictions are combined by using some kind of arbitration rule - such as returning the prediction with the majority of occurrences, with preference given to the arbiter's prediction in the case of a tie. Both the combiner and the arbiter approach can be extended to a form of hierarchical meta-learning [Chan & Stolfo 97], [Chan & Stolfo 95a].

In this case an arbiter tree or a combiner tree is built in a bottom up fashion. The tree leaves are associated with the local data mining algorithms. The predictions of these algorithms are sent up the tree, being used as input data for the combiners or arbiters in the next higher tree level. This process is recursively applied until a root node outputs the final prediction. It has been observed that this hierarchical meta-learning scheme improves prediction accuracy over the simpler (and faster) one-level meta-learning scheme. Furthermore, hierarchical meta-learning can often lead to a prediction accuracy equivalent to the one achieved with a global data mining algorithm (mining the entire original data set). Also, sometimes hierarchical meta-learning can lead to a prediction accuracy even higher than the one achieved with a global data mining algorithm. However, Hierarchical meta-learning schemes are often able to sustain the same level of accuracy as a global classifier trained on the entire data set.

ISSN: 2277-3754
ISO 9001:2008 Certified
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 6, June 2012

REFERENCES

[1] Lior Rokach, Pattern Classification Using Ensemble Methods, 2010.

[2] Alex A. Freitas, Simon Hugh Lavington, Mining Very Large Databases with Parallel Processing, 2000.

[3] Oded Z. Maimon, Lior Rokach, Data Mining and Knowledge Discovery Handbook, 2005.

[4] Nong Ye, the Handbook of Data Mining, 2003.

[5] P. Chan and S. Stolfo. Meta-learning for multistrategy and parallel learning. In Proc. Second Intl. Work. On Multistrategy Learning, pages 150–165, 1993.

[6] Chan, P., and Stolfo, S. 1993a. Experiments on multistrategy learning by meta-learning. In Proc. Second Intl. Conf.Info. Know.Manag. 314–323.

[7] Chan, P., and Stolfo, S. 1993b. Toward parallel and distributed learning by meta-learning. In Working Notes AAAI Work. Know. Disc. Databases, 227–240.

[8] Chan, P., and Stolfo, S. 1995. A comparative evaluation of voting and meta-learning on partitioned data. In Proc.Twelfth Intl. Conf. Machine Learning.

[9] A. Prodromidis, P. Chan and S. Stolfo. "Meta-learning in distributed data mining systems: Issues and approaches", Advances in distributed data mining, AAAI Press, 2000.

[10] P. Chan and S. Stolfo. "Learning arbiter and combiner trees from partitioned data for scaling machine learning", Proceedings of the First International Conference on Knowledge Discovery and Data Mining, 1995.

AUTHOR BIOGRAPHY

Dr.M.Usha Rani is an Associate Professor in the Department of Computer Science and HOD for MCA, Sri Padmavati Mahila Viswavidyalayam (SPMVV Women's' University), Tirupati. She did her Ph.D. in Computer Science in the area of Artificial Intelligence and Expert Systems. She is in teaching since 1992. She presented many papers at National and Internal Conferences and published articles in national & international journals. She also written 4 books like Data Mining - Applications: Opportunities and Challenges, Superficial Overview of Data Mining Tools, Data Warehousing & Data Mining and Intelligent Systems & Communications. She is guiding M.Phil. and Ph.D. in the areas like Artificial Intelligence, Data Warehousing and Data Mining, Computer Networks and Network Security etc.

Mrs G.T.Prasanna Kumari is an Associate Professor in the Department of Computer Science and Engineering, Gokula Krishna College of Engineering, Sullurpet. She is pursuing Ph.D., in Computer Science in the area of Distributed Data Mining Systems. She is in teaching since 1999. She presented papers at National and International Conferences and published articles in national & international journals.