I, Dipayan Chattopadhyay, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Science.

It is entitled:
Distributed Decision Tree Induction Using Multi-agent Based Negotiation Protocol

Student's name: Dipayan Chattopadhyay

This work and its defense approved by:

Committee chair: Raj Bhatnagar, Ph.D.
Committee member: Karen Davis, Ph.D.
Committee member: Chia Han, Ph.D.
Committee member: Paul Talaga, Ph.D.
Distributed Decision Tree Induction Using Multi-agent Based Negotiation Protocol

A thesis submitted to the
Graduate School
of the University of Cincinnati
in partial fulfillment of the
requirements for the degree of

Master of Science
in the Department of Electrical Engineering And Computing Systems
of the College of Engineering and Applied Sciences

by
Dipayan Chattopadhyay
BTech. SRM University
May 2012

Committee Chair: Raj Bhatnagar, Ph.D.
Abstract

Exponential growth of data and its distributed nature has made data analysis tasks difficult. Most distributed databases are also sensitive to moving data between different data sites due to privacy and security concerns. Learning of decision trees from distributed databases has been implemented as cases of cooperative multi-agent systems that provide summaries needed from the participating datasets. Most such algorithms also fail to create intermediate models, classify minority instances in skewed databases, and provide classification accuracies that are not at par with accuracies from centralized data. This work proposes a novel approach, based on competing multiple agents that engage in game-theoretic bidding, for inducing decision trees from distributed databases without exchanging any data summaries with other databases. We demonstrate the efficacy and validity of our approach with a number of datasets.
Acknowledgement

This work wouldn’t have been possible without the support and guidance of my advisor Dr. Raj Bhatnagar. He has been a great source of inspiration and has always been available to me when in need. His constant support has immensely helped me improve the quality of this thesis and has provided me the ability to conduct good research work. I would like to thank my committee members to take out time and review my thesis and serve in the thesis committee. Lastly I would like to thank my parents for their eternal love and blessings.


**Contents:**

1. Introduction ................................................................................................................................. 1
   1.1 Motivation ................................................................................................................................. 3
       1.1.1 Security and privacy concerns in centralizing data ......................................................... 4
       1.1.2 Inability to detect instances belonging to minority instances ........................................ 4
       1.1.3 Unavailability of local interpretable models .................................................................... 5
   1.2 Approach ................................................................................................................................... 5
   1.3 Outline ...................................................................................................................................... 6

2. Background ..................................................................................................................................... 8
   2.1 Classification ............................................................................................................................ 8
   2.2 Decision tree as a Classifier ...................................................................................................... 9
   2.3 Game theory and decision theory .............................................................................................. 12
   2.4 Distributed decision tree induction mechanisms ....................................................................... 17
   2.5 Multi-agent systems .................................................................................................................. 19
   2.6 Negotiation in multi-agent systems .......................................................................................... 20

3. Decision tree induction .................................................................................................................. 23
   3.1 Hunts algorithm .......................................................................................................................... 23
   3.2 Splitting criteria .......................................................................................................................... 25
       3.2.1 Entropy ............................................................................................................................... 25
       3.2.2 Gini Index ........................................................................................................................... 26
       3.2.3 Classification error .............................................................................................................. 26
       3.2.4 Information gain ............................................................................................................... 27
       3.2.5 Gain ratio .......................................................................................................................... 28
   3.3 Established algorithms ................................................................................................................. 29
       3.3.1 CART ................................................................................................................................. 29
       3.3.2 ID3 algorithm .................................................................................................................... 30
       3.3.3 C 4.5 algorithm ............................................................................................................... 31
   3.4 Agent based decision tree induction ........................................................................................... 31
       3.4.1 Decision tree learning in distributed environment ............................................................ 32
       3.4.2 Negotiation among MAS for quality improvements .......................................................... 34

4. Our Approach ................................................................................................................................. 36
List of Figures:

Fig. 2. 1: Rule Based Classifier ........................................................................................................... 9
Fig. 2. 2: Perceptron ..............................................................................................................................
Fig. 2. 3: Sample Decision Tree  Fig. 2. 4: Graphical Representation ................................................. 10
Fig. 2. 5: Traffic Network .................................................................................................................... 15
Fig. 2. 6: Regulated Market Scenario .................................................................................................. 16
Fig. 3. 1: Comparison of impurity measures .................................................................................... 27
Fig. 3. 2: Sequential Learning ........................................................................................................ 32
Fig. 3. 3: Parallel Learning Methodology .......................................................................................... 33
Fig. 3. 4: Centralized negotiation mechanism ................................................................................ 34
Fig. 4. 1: System Design ................................................................................................................... 41
Fig. 4. 2: Sample decision tree .......................................................................................................... 42
Fig. 4. 3: Rule set derived from decision tree .................................................................................. 43
Fig. 4. 4: Master rule set creation ..................................................................................................... 48
Fig. 4. 5: Negotiation Process ........................................................................................................... 52
Fig. 4. 6: Rule growth for a single rule .............................................................................................. 54
Fig. 5. 1: Classification accuracy comparison in seeds dataset ....................................................... 62
Fig. 5. 2: Error comparison on seeds dataset ................................................................................... 62
Fig. 5. 3: Classification accuracy comparison in wine dataset ......................................................... 63
Fig. 5. 4: Misclassification in wine dataset ....................................................................................... 64
Fig. 5. 5: Classification accuracy in Pima Indian dataset ................................................................. 65
Fig. 5. 6: Classification error in Pima Indian dataset ....................................................................... 65
Fig. 5. 7: Classification accuracy in banknote authentication dataset ........................................... 66
Fig. 5.8: Classification error in banknote authentication dataset........................................67
Fig. 5.9: Classification accuracy comparison in skewed dataset.....................................68
Fig. 5.10: Classification error comparison for skewed seeds dataset..............................69
Fig. 5.11: Classification accuracies in skewed wine dataset........................................70
Fig. 5.12: Misclassification comparison for skewed wine dataset..................................70
Fig. 5.13: Classification accuracies in skewed Pima Indian dataset...............................71
Fig. 5.14: Misclassification comparison for skewed Pima Indian dataset.......................72
Fig. 5.15: Classification accuracies in banknote authentication skewed dataset..............73
Fig. 5.16: Misclassification comparison for banknote authentication skewed dataset.......73
Fig. 5.17: Difference in classification accuracies before and after negotiation..............75
Fig. 5.18: Scalability Analysis.....................................................................................76
List of Tables:

Table 3.1: Example dataset showing occurrence of rain ........................................ 24
Table 4.1: Snapshot of dataset captured by a rule .................................................. 39
Table 5.1: Dataset Summary .................................................................................. 60
Table 5.2: Scalability Analysis .............................................................................. 75
List of Algorithms:

Algorithm 3. 1: CART ................................................................. 27
Algorithm 3. 2: ID3 ................................................................. 28
Algorithm 4. 1: Master Rule set Creation ........................................... 44
Algorithm 4. 2: Rule Quality Evaluation ............................................. 45
Algorithm 4. 3: Negotiator Operations for Negotiation .............................. 47
Algorithm 4.4: Agent Operations for Negotiation .................................... 48
Chapter 1

1. Introduction

Remarkable growth of the internet coupled with cheap memory storage devices has led to an exponential growth in the amount of data being generated and stored. It is very common to see data storage devices of the order of terabytes and petabytes. Most of the data being stored are in the form of logs, transactions, and experiment observations. The data is also being generated very quickly and in many different formats. One of the main reasons of storing so much data is to create process models for prediction and forecasting. The science of data analytics and mining is being utilized by a variety of businesses and academic researchers. Businesses often store data to monitor customer purchases and weblogs to learn about customer behaviors and thus improve profitability. Medical data is being stored to predict the causes of diseases. Data is also being used for weather forecasting, decision making, engineering design and analysis, and social behavior monitoring.

Classification is one of the fundamental operations performed in data-mining. Classification aims at learning process models from observed data for predicting the outcome of a future instance of the same process. Classification assigns instances to predefined categories. Classifiers learn models in terms of patterns from training datasets. These models are then used on the test data to predict their class assignments. Classifiers have widely been used for medical diagnosis, business intelligence, science, engineering, and in many other fields. There are many popular and well-known classifier systems such as the decision trees, perceptron, and rule-based classifiers etc.
Decision trees generate rules that are conditions, or conjunctions of conditions involving several attributes and cutoff values. Decision trees are very popular, fast to learn, and reasonably accurate classifiers that provide intuitive and comprehensible insights into the data and the classification decisions.

Decision trees learning algorithms have historically been designed with an assumption that all the data required to induce a tree are located at one common location. This assumption is not true in most of the modern distributed applications. Most of the data is now scattered over geographically different locations and combining this data at one common data location is a huge task that has many critical implications. Some of the most severe problems that emerge out of combining this data are related to security, availability of resource to store huge volumes of data and privacy concerns. There is a need therefore to create decision trees by keeping in mind all these issues involved. Most of the distributed decision tree induction algorithms proposed fail to provide an intermediate interpretable model which can be used to check validity and often suffer when dealing with skewed datasets. The accuracy provided by these algorithms in classification has not been at par with decision tree algorithms grown on centralized data.

Game theory is the study of mathematical models that aims to resolve conflicts and improve cooperation among intelligent decision making agents to provide rational solutions to difficult coordination problems. The games played in game theory must have well defined players, actions available to the players, and payoffs for each outcome to each agent. A game should also have a strategy for reaching a state of stable equilibrium. Much of the earlier research in building decision trees from distributed databases uses the paradigm of cooperative agents that are willing to share any parts of their local datasets that may be needed to centrally build a decision tree.
representing all the datasets. This process may lead to some of the security and privacy issues discussed previously.

We propose a novel approach to deal with the problem of learning from distributed data using a game theoretic framework. We present a paradigm of competing agents, each representing its own dataset, and each attempting to help create a decision tree closest to its own dataset. These agents provide their local decision trees to the central coordinator, the central coordinator converts each decision tree into paths, representing rules, and presents these rules to agents for their bids and/or improvement suggestions. Each agent bids on the suitability of each such rule depending on its own dataset, trying to minimize its information loss if the rule were to be included in the consensus decision tree. These agents also suggest modifications to existing rules for which they may incur lower losses. Eventually, with such repeated negotiations and iterative bidding on rules the coordinator evolves a set of rules that minimizes each agent’s perceived information loss. During this process no real data is ever exchanged, thus removing the possibility of a security and privacy leak. The main objective of this work is to create decision trees out of distributed data using multiple agents that compete with each other to create rules that are acceptable to all. It is observed from experiments conducted on various datasets that the classification accuracies of the trees built using our approach are very close to those of the trees built from unified and centralized data.

1.1 Motivation

In this section we deal with some of the factors that have motivated us to pursue this research work.
1.1.1 Security and privacy concerns in centralizing data

To understand the impending dangers in merging databases and scenarios where a decision tree creation might be needed, consider an intelligent credit card authorization application. For an application to reach at any decision on awarding a hike in credit limit or awarding a new credit card, it must have information about applicant’s name, social security, date of birth, bank balance and pin numbers. All this information in most circumstances will not be available at one particular location much less in one system. The task of moving such critical sensitive information from different systems to a centralized location would demand high security and encryption. If the data is huge, the processes of encryption and decryption is going to be time consuming and may also add to the overhead. In addition to this, if data is structured and represented as tables, then combining data would mean doing complex database joins that would add to the already high time consumption. Merging of a large volume of data would require huge infrastructure for data storage which in itself becomes a very costly affair.

The work presented here tries to remove any privacy or security risks by creating decision trees on each participating data site and does not require transfer of any data. Trees are converted to rule sets and moved to a centralized arbitrator component. This central arbitrator then builds a tree acceptable to all data sites. Since the rule sets occupy a lot less space in comparison to actual data, our methodology removes the requirement of huge storage facilities.

1.1.2 Inability to detect instances belonging to minority instances

One of the challenges that current distributed decision tree algorithms face is failure to classify instances belonging to classes with fewer records. This is found in skewed databases where the number of records belonging to one class are way more than that belonging to other classes. One such instance where we might be interested in finding instances belonging to minority classes
is when dealing with income distribution data. It is often observed that there are overwhelmingly large number of records near the mean and very few records towards the higher and lower bounds. It will be of great interest when tasked to find traits about these sections of people. Inability to detect and predict such characteristics leaves a lot to be desired.

This work tries to come up with a class sensitive metric that enables the rules to classify correctly those instances that have a small representation. This results in reduction of the number of misclassification errors and detection of classes with very little representation.

1.1.3 Unavailability of local interpretable models

Current distributed decision tree induction algorithms lack in providing any intermediate data models that are interpretable before the complete decision tree is built. Such kind of an algorithm is always very helpful to provide a rough insight onto to the data before the whole model is built. This work creates intermediate interpretable data models that can be easily understood and can provide an insight into the final model before it is built.

1.2 Approach

This work aims at solving the problems discussed above with the current methods of decision tree induction from distributed databases. Some of these issues are resolved in the following way:

- Initial decision trees are built at individual data nodes/sites. Data is never shared or transferred between two data nodes or in between a node and a central controller. Only the local decision trees are shared between a node and a centralized negotiator component, thereby removing the problems related to privacy and security of the underlying data.

- No central repository of all the data needs to be created, thus removing the requirement of having huge hardware and other resources for storage and maintenance of this data.
• Raw data tuples from different data nodes are never combined or modified, thus removing the need of performing joins, which saves a lot of computational resources. Also, due to the parallel nature of the distributed algorithm, it is much faster in comparison to any sequential algorithm.

• A class sensitive metric is created to handle the problem of detection of instances belonging to classes having very few instances. This metric provides more weightage to instances belonging to classes with fewer instances than to classes with huge number of instances.

• An easily interpretable model is first created at every data node and then these models are iteratively combined to form a model that suits all the nodes. Our method successfully creates intermediate models at every step of this induction process, thereby providing a rough estimate of the final model before the complete induction process finishes.

• A negotiation process ensures that the model finally created from the intermediate models submitted by the individual data nodes is acceptable to all the data nodes. The negotiation process ensures that the final model created is much more acceptable to all the nodes and therefor reflects the information embedded within each one of them.

Our results show that the tree built from the proposed algorithm has a classification accuracy very close to that of the tree built using well established algorithms applied on all the data integrated in a common data repository. In addition, the work in this thesis shows the improvement in the decision tree accuracies as a result of the data nodes engaging in the negotiation process.

1.3 Outline
The remainder of the thesis is organized as follows: Chapter 2 provides a background on decision trees, multi-agent systems, and distributed decision tree induction algorithms. Chapter 3 discusses in detail the decision tree induction process, well established algorithms for decision tree building, and agent based decision tree induction. Chapter 4 details our approach and Chapter 5 presents the results obtained by our approach and compares them to alternate approaches. Chapter 6 concludes this thesis by providing the summary of work done and possible extensions.
Chapter 2

2. Background

The previous chapter discusses distributed decision tree induction process and outlines the problems associated with the current methodologies. It also provides a brief introduction to the work accomplished in this thesis and provides a road map to solving these problems. This chapter gives a brief introduction to classification, decision tree induction and game theory. We discuss in detail on how multi-agent systems have been used in the past for dealing with problems in distributed environment. We also discuss how negotiation can be used as a means to achieve consensus among agents in conflict in a multi-agent scenario. Some of the previously explored methods of distributed decision tree induction are also discussed.

2.1 Classification

Classification is the process of categorizing the data space based on already existing knowledge, gained in terms of the observed instances and their class labels. The classification algorithms process a training set containing a set of labeled training instances and then seek to predict labels for a set of testing instances. Each of the attributes included in an instance can be considered as a feature. Based on how features vary for different classes it is possible to predict the class labels for the instances in the training data set [21].

Any algorithm that learns and implements classification in a data space is called as a classifier. In machine learning terms, a classification algorithm is considered to be an instance of supervised learning. Classifiers can be divided into two types, namely, of “Black-box” and “White-box” types.
A classifier is considered a black-box type if the requirement is to only find the label of the test data, and its examples are Support vector machines and neural networks. If the requirement is to get an insight into the rules that lead to the classification decisions then it is called a “white-box” type of classifier, such as the decision trees and the rule-based systems. [21]. Figure 2.1 shows a “White-Box” classifier in the form of a rule-based classifier and figure 2.2 shows a perceptron classifier which is “Black-Box”.

2.2 Decision tree as a Classifier

Decision trees are classifiers that utilize tree like structures as predictive models to map the regions of data space to class labels. The main goal is to predict the class label of an instance based on as few of instance’s features as possible [34]. In a typical decision tree, each tree node can be considered as a decision location based on attributes and split values along the path from the root of the tree to the leaf node. A conjunction of tree nodes and their conditions leads to a particular class label for an instance. The class label is always assigned at the leaf node. A decision tree is one of the most popular techniques of supervised learning. There are generally two phases to a Decision tree induction process, viz. growth phase and pruning phase [18].
A decision tree induction process usually involves splitting the whole dataset into subsets based on the conditions tested at nodes which act as split-points. The process of selecting the best attribute for splitting is determined by using an attribute value test. This process is repeated in an iterative fashion at each node of the tree. The recursive process ends when all the data points corresponding to a node (or a data space region) have the same class label i.e. the node has pure class or when splitting of the dataset doesn’t result in any information gain.

![Sample Decision Tree](http://www.geocomputation.org/1998/61/gc_61.htm)

**Fig. 2.3: Sample Decision Tree**

A path in a tree from the root to the leaf can be considered as conjunction of attribute based decisions that lead to a particular class label. And each such path of the decision tree also demarcates a region in the data space as belonging to a particular class. A decision tree can thus be interpreted in the form of a set of regional boundaries in the data space as shown in Fig. 2.4. It shows an interpretation of a decision tree having 2 classes separated by horizontal and vertical lines that act as decision-test induced boundaries. These lines try to create regions which are pure i.e. have instances belonging to only one class. Newer lines are generated for every test in an iterative manner and the lines get generalized to axis-parallel hyper planes in a large dimensional data space. The tree-induction iterations stop when the regions in data space becomes pure or when
drawing axis-parallel planes can no longer help in increasing the class purity of the resulting regions.

Decision trees create models that are very easy to comprehend and are very intuitive in nature. Decision trees have some drawbacks too. Some of the drawbacks are:

- **Over-fitting:** Decision trees try to split the whole data-space using horizontal and vertical decision lines. The decision tree induction algorithms sometimes tend to split the data space many times during the iterative process to attain the goal of pure classes. This leads to over-fitting or overlearning the model which results in bad predictions.

- **Greedy algorithm while choosing attributes:** Decision tree induction algorithms based on ID3 [2] or C4.5 [3] split the data space iteratively by using a greedy algorithm. The split points leads to creation of sub-regions having instances belonging to one particular class [2] [3].

- **Restricting one rule to one instance:** Decision trees often restrict a data tuple in a dataset to help define only one rule. This is because the data is portioned such that that the sub-regions formed are mutually exclusive and do-not share boundaries. This is done by most of the decision tree induction algorithms to find the smallest set of rules for most of the data. By doing this classifiers often overlook certain useful information and patterns that may have good support [4].

It is often observed that the decision trees often don’t fare well when dealing with databases that are skewed in other words databases having most of the instances belonging to one class [26]. Work done in this thesis work tries to address this problem by creating a cost sensitive metric.
2.3 Game theory and decision theory

Game theory is a mathematical model concerned with study of making decisions where there are multiple agents or rational opponents involved under condition of competition or conflicting interests. Game theory assumes that each participant will act in a rational manner and try to resolve the conflicts of interests in its favor [34].

Game theory and decision theory are very much related with each other as discussed in work by Parsons and Wooldridge [35]. Decision theory [36] is way by which analysis can be performed on choosing the best option from a series of options when the result of any of the options is uncertain. Decision theory works by identifying the “best” solution. “Best” can have multiple meanings, most common of them is something that can maximize the utility of the decision maker [35]. The concept of game theory was first coined by Von Neumann and O, Morgenstern [37] by studying the games of chess and checkers. Game theory studies the interactions among agents that are self-interested. Game theory studies the strategies that maximize welfare/utility of an agent in a multi-agent encounter and designs protocols that have some desirable properties.

The main objective of game theory is to maximize the overall gain and reach a stable state of equilibrium. Any game is governed by “solution concepts”. Solution concepts describe strategies that will be adopted by players that will result in a game. An ideal game theoretic framework should have certain properties as outlined by Sandholm [40]. Some of these properties are:

- **Guarantee of success**

  The solution concept should ensure that the agents should eventually reach an agreement and should converge.
• **Maximizing Utility**
  The solution concept should ensure that sum of utilities of the participating agents should get maximized.

• **Pareto Efficiency**
  A solution concept should ensure that there exists no other solution that will result in gain of utility of one of the agents without affecting the gains of all other participating agents.

• **Rationality of individual solution concept**
  A solution concept is called “rational” if it’s in the best interests of every agent to follow it. Without this solution concept there is no incentive for individual agents to participate in the game.

• **Stability**
  A solution concept is stable when all the agents playing the game behave in same way. A solution concept must be in a position to guide every participating agent to an equilibrium.

• **Simplicity**
  The solution concept should be simple enough for the participating agents to determine the optimal strategy.

We try to achieve these properties while designing our game theoretic framework. We in this work, show that the participating agents do reach an agreement and converge, by testing our algorithm on various datasets. We also show that the protocol established tries to maximize the utility of every individual agent thereby increasing the overall gain. The protocol designed clearly defines an incentive to every agent and all agents work rationally to increase their individual utility.
The protocol established is also simple enough for the agents to determine the most optimal solution and come to a consensus.

Most common solution concepts adopted aim to guide participating agents to a state of equilibrium. Equilibrium is a state where the net loss or gain is zero. In other words to attain equilibrium, one agent’s loss is other’s gain and the net loss and gain results in a “zero sum”. Duxon et al. [41] proposes some of the general properties of an equilibrium:

- **Equilibrium property P1**: Agents must have a consistent behavior.
- **Equilibrium property P2**: There should not be any incentive available to an agent to change its behavior.
- **Equilibrium property P3**: An equilibrium is achieved as an outcome of some dynamic processes.

There are two most commonly adopted types of equilibrium. They are as follows:

- **Nash Equilibrium**

  Nash equilibrium is a solution concept involving multiple agents where each agent has knowledge of the strategy being used by other agents, and no agent can gain utility by just changing their own strategy. If all the agents involved have chosen a strategy and no player can make a gain without any other player also changing their strategy then the set of strategies from all the agents involved are said to be in Nash Equilibrium [42].

  One of the classic examples of Nash equilibrium is a “traffic network” problem. Let us consider the following traffic network graph below in fig. 2.5. Let the source be A and destination be D. x represents the number of cars that pass through each of the paths.
There are 3 possible paths to reach D from A namely: ABCD, ABD and ACD. Let us consider there are 100 cars that need to be routed. If 25 cars pass through ABD, 25 cars pass through ACD and 50 cars pass through ABCD then each of these routes take 3.75 time units for smooth passage of all cars and the system is said to be in Nash equilibrium.

Fig. 2. 5: Traffic Network

- Competitive Equilibrium

Competitive equilibrium is used in a multi agent setting where every agent is focused on increasing their own utility. Agents participating in a competitive environment may form a temporary coalition with other agents but conflicting goals makes this kind of coalition a very difficult task. Most common mechanisms of reaching a competitive equilibrium is through the process of bidding and auction. Most of the automatic agents in the e-commerce platform follow this kind of mechanism to reach at a consensus. Every participating agent submits its own bid in an attempt to bring the consensus towards itself. One agents’ gain becomes another’s loss, this resembles the “zero sum game”. It is often seen that individual local gains of the agents lead to a global goodness and this keeps the
system stable. Let us consider a very realistic example of how the markets work based on the competitive equilibrium.

Let us consider a market to be an exchange and let us have 2 agents competing namely demand and supply. Demand and supply are responsible for regulating the price and the quantity demand. In a properly regulated market the price and quantity is regulated. Equilibrium property p1 is satisfied since at equilibrium price, the supply becomes equal to demand. Equilibrium property p2 also is satisfied since demand doesn’t have any incentive to change its behavior at equilibrium price. Similarly we can see the dynamic process of equilibrium in an unbalanced case, i.e. when equilibrium is not reached. The price goes up when the demand is more and supply is less, this puts pressure on supply. When supply increases, though demand is high the price reaches an equilibrium. This behavior uphold the third property of equilibrium. This is demonstrated in the regulated market scenario figure as depicted in figure 2.6.

![Regulated Market Scenario](image)

**Fig. 2.6: Regulated Market Scenario**

Game theory has two main branches: cooperative and non-cooperative game theory. In cooperative game theory agents come together as coalitions in a multi-agent setting to work together to minimize their losses. Non cooperative game theory deals with the study of agents that
interact with each other to achieve their own goals and there is no coalition. In this work we deal with non-cooperative agents that compete with each other and try to reach at a competitive equilibrium.

2.4 Distributed decision tree induction mechanisms

Classification, especially decision tree induction is a well-researched problem. Some of the most famous classification algorithms are: Bayesian Models [5], Support vector machines [6], decision tree induction mechanisms like ID3 [2] and C4.5 [3]. Majority of these algorithms differ in choosing the objective function and the size of the tree created.

Stolfo et el. [7] propose a classification mechanism where the algorithm learns a weak classifier on each partition. The classifiers and samples of data are then centralized. This method however does require transfer of real data thus compromising on security, privacy and data ownership concerns. Another class of famous classifier systems are the Meta classifiers. Boosting [8] and Bagging [9] are two famous meta classifiers that first create a model on samples of Data and then these models are aggregated using a second algorithm which can be as simple as just taking the majority. Bar-Or et al. [10] propose a new methodology to hierarchically execute the ID3 algorithm by centralizing, i.e. collecting the best attribute and cutoff to split the data space at every level of the tree induction process. In this paper, the authors use a higher and lower bound for attribute values at each level of the tree instead of one fixed erroneous value. This methodology however can work well only in a close knit environment and doesn’t provide an intermediate interpretable model for early evaluation.

Merugu and Ghosh [11] discuss using probabilistic classification on heterogeneous data. This type of classification has however been found to be inept for large data. Caragea et al. [12]
propose using an effective splitting criterion that can be used in a distributed fashion. The authors prove that by providing statistical summaries from different sites to a central component a lot of communication costs associated with brute force centralization can be brought down. They propose a methodology of tree building where most of the tree building work is done by a central arbitrator and involves communication between the arbitrator and other data sites that share summaries. Since most of the work is done by the arbitrator it becomes a very time consuming process and makes the process sequential. In addition this method also doesn’t provide any interpretable models. Olsen [13] and Giannella et al. [14] use Gini index as a measure for impurity and show that the Gini between two attributes can be calculated as a dot product of two binary vectors. The remote sites only communicate the projected vectors instead of transferring raw data or large binary vectors. The authors say that they reduce the communication cost by 80% and get an accuracy of 80% when compared to a centralized method where all the data is located at one central location.

Domingos [15] proposes a decision tree induction algorithm that builds a tree on overlapping subsets of original dataset. A data set is generated and added to the original dataset. The trees are used to classify these datasets and a separate individual tree is grown on the augmented training set. The authors prove that the trees produced are more stable and accurate. But the drawback associated with this method is that the training set created is larger. Provost and Hennessy [16] discuss about a rule based learning algorithm that can generate rules on each subset of training data. If the rule is satisfactory, it is passed for evaluation to other datasets. All the rules that are satisfactory for all the datasets are retained and for a part of larger superset of rules. The final rule set created has many rule conflicts and the classification accuracies of the model are not at par with the centralized algorithms.
Hall e.t. al. [17] extend the approach adopted by Provost and Hennessy [16] by introducing an approach for rule conflict resolution. The authors show that there is a lot of promise in doing a conflict resolution on the rules set created from remote data nodes. We utilize a similar approach for rule building but use a negotiation framework to make the rules better and acceptable for all. Tong e.t al. [19] propose an approach to create Decision Forests that can combine multiple decision tree models. The authors say that on combining models of similar predictive quality, the quality significantly improves over the individual model both in training and testing cases.

Andrzejak e.t al. [20] propose an approach of learning distributed data in parallel by creating interpretable models of the overall model. The authors propose an approach to merge many decision trees formed at different data sites into a single tree.

2.5 Multi-agent systems

Agent based systems (ABM’s) are a class of computational models having autonomous agents that work independently to attain individualistic goals while still having a global perspective. Multi agent systems (MAS’s) comprise of sets of intelligent agents that run within a specified framework/ environment. These agents can be divided into three different types. Passive agents that do not have any goals, active agents that have simple goals and complex agents that do complex computations. All these agents have some common inherent characteristics. All these agents are autonomous in making any decision of their own. All the agents have a local view and no agent knows the system as a whole. All these agents are also decentralized, meaning there can’t be any other agent controlling one agent. Multi agent systems are used mainly for agent oriented software engineering [22] [23]. MAS provides an excellent mechanism for implementing modular approach in a distributed computing environment. MAS’s have found their utility in decision
making systems. Caragea et al. [12] have implemented a distributed knowledge acquisition system from heterogeneous and distributed data sources called as INDUS.

Multi agent systems have previously been used in the decision tree induction process. Bhatnagar and Srinivasan [24] show a decision tree induction process for distributed environment using MAS. Each remote database is represented by an agent that is responsible for collecting the statistical summaries from the data. A centralized learner component is initiated that can be at a centralized location or reside in one of the participating data sites. The learner is authorized and responsible for collecting all the summaries from remote data sites.

2.6 Negotiation in multi-agent systems

J.S.Rosenschein [38] in his PhD thesis titled “Rational Interaction: Cooperation among intelligent agents” was the first to show the usage of game theory to solve problems in distributed systems. He showed how individual agents could work together to solve problems of common interests. Rosenschein [38] analyzed a range of multi-agent interactions using game theoretic techniques. He showed deal making and cooperation among multiple agents without communication. Agents would simply compute the outcomes and knew that the other agents involved would also do the same. A lot of work on game theory and multi agent negotiation has been done since then.

Most compelling applications of game theory in multi agent environment have been in the field of negotiation [27] [39] [40]. Negotiation is a process by which agents can agree on something that is of common interest to them [35]. In a multi-agent negotiation setting agents interact, coordinate and bargain with each other to achieve common goals while pursuing common
interests. Some of the most common ways of negotiation are: direct-negotiation, bidding/auction, contracting etc. They are elaborated as follows:

- **Contracting:** Contracting is a negotiation mechanism where an agent wants to assign some work to other agent or groups of agents and monitor the work done by them. This type of negotiation is most common in cooperative game theory where some incentive is given to agents to work for other agents.

- **Direct Negotiation:** Agents directly talk to each other to find out a common ground and reach at an acceptance.

- **Bidding/Auction:** Most common negotiation mechanism that uses a centralized auctioneer. All the agents submit bids to the auctioneer and continue bidding till no further bids can be made. Finally auctioneer chooses the agent with the highest bid.

We use an auction/bidding mechanism for negotiation in our work. This sort of approach has previously been tested for network routing, traffic networks, e-marketing, online trading etc. “contract net protocol” introduced by Smith [43] utilizes the auction based bidding process to solve issues related to network routing problem. “Contract net protocol” uses a centralized manager that manages the whole process of bidding. The manager send out eligibility, task and expiry time to the participating nodes. The participating nodes send their bids to the manager and the manger chooses the best bid based on the suitability of the node and the time consumption.” Contract net protocol” implements a simple base case of bidding process where the bidding takes place only one time. Sandholm [44] improvises this protocol for vehicle routing problem and utilizes a repetitive bidding mechanism. We also use such a mechanism where the participating agents can bid multiple times until they don’t have any better option to bid.
Kraus [27] proposes a framework for server interaction in a multi-server environment where every server puts forward a proposal. All the other servers have the liberty of accepting or rejecting this proposal. If a proposal is accepted by all the servers then the process stops. The approach adopted by us doesn’t stop the negotiation process if incase a proposal is accepted by all participating parties. All the participating agents are encouraged to modify an accepted proposal and continue to do so in an iterative fashion until there is no more proposal that can be obtained by modification of existing proposal that is better than the existing proposal, by any of the participating agents. This process is not time bound. Paliwal [25] proposes a negotiation framework for making decisions in MAS that utilizes a direct negotiation approach where repetitive bidding is done. Paliwal [25] also proposes to store and modify rules that may be rejected by many but are nearly successful rules.

All the above mentioned negotiation frameworks leave scope to development of a distributed decision tree induction process that can inculcate the negotiation aspect into decision tree making and create trees that can discover patterns more efficiently.
Chapter 3

3. Decision tree induction

The previous chapters have provided a background on classification and game theory. We discuss about decision trees and study several methodologies adapted in the past for distributed decision tree induction. We show how multi-agent based systems have been helpful in solving many distributed problems. We have seen how negotiation has helped in creating a consensus among multiple agents. This chapter aims at discussing in detail the process of induction of a decision tree. In this chapter we discuss some of the very famous decision tree induction algorithms and understand the advantages and drawbacks associated with them. Different measures used to find the best attribute and split value also will be discussed.

In a decision tree the leaf nodes are always assigned class label. The non-leaf nodes contain attribute-value pairs that lead to a class label. Most of the decision tree induction algorithms use a greedy strategy to select best attribute to split the data and create purer sub regions. One such algorithm that is used by most of the famous decision tree induction algorithms is hunts algorithm.

3.1 Hunts algorithm

Let $D_i$ be set of training databases that are associated with a data node $i$ and let $C$ be set of class labels such that $C = \{C_1, C_2, ..., C_n\}$ where $n$ is the number of classes. Following are the steps behind Hunt’s algorithm:

1. If all the instances in $D_i$ belong to the same class then $i$ is a leaf node that can be labelled at $C_i$. 


2. If $D_i$ is heterogeneous meaning it has instances from different classes then, an *attribute test condition* is used to select an attribute to split the dataset $D_i$ to smaller subsets. The instances of $D_i$ are distributed to the child subsets. This algorithm is then recursively applied to every child node. The recursive process continues until step 1 is accomplished or if no gain is achieved in further splitting of data sets.

Let us consider an example to demonstrate the hunt’s algorithm. Consider table 1 describing the possibility of rain to occur during certain conditions. Humidity, wind speed, temperature are the attributes here and occurrence of rain is the class. It can be seen that the class is not homogeneous. This implies that the rule 2 of hunt’s algorithm needs to be implemented, i.e. to split the dataset into subsets having pure class. Let us ignore the attribute test condition to split the dataset as it will be discussed later. We can split the dataset with instances 1 and 2 becoming part of a child subset and rest can be another. After the first split hunt’s algorithm is again applied recursively on the child datasets created. This splitting of data will continue until a dataset is obtained that has a pure class according to Hunt’s first rule.

<table>
<thead>
<tr>
<th>Humidity</th>
<th>Wind Speed</th>
<th>Temperature</th>
<th>Occurrence of Rain</th>
</tr>
</thead>
<tbody>
<tr>
<td>56%</td>
<td>20mph</td>
<td>80 degrees</td>
<td>Yes</td>
</tr>
<tr>
<td>73%</td>
<td>50mph</td>
<td>75 degrees</td>
<td>Yes</td>
</tr>
<tr>
<td>20%</td>
<td>15mph</td>
<td>40 degrees</td>
<td>No</td>
</tr>
<tr>
<td>45%</td>
<td>10mph</td>
<td>50 degrees</td>
<td>No</td>
</tr>
<tr>
<td>10%</td>
<td>5mph</td>
<td>40 degrees</td>
<td>No</td>
</tr>
<tr>
<td>90%</td>
<td>25mph</td>
<td>60 degrees</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 3.1: Example dataset showing occurrence of rain
The next section discusses on choosing the best attribute to split the dataset.

3.2 Splitting Criteria

There are many measures that can be used to determine the best attribute and cutoff for splitting the databases. Several famous decision tree induction algorithms in the past have used different measures for splitting. Quinlan [2] used entropy as a measure to determine the attribute to split for his famous ID3 algorithm. Breiman e.t al. [26] used Gini Index as a measure for determining the best attribute for split for his famous CART algorithm.

Measures developed to select the best split are often based on degree of impurity of the child nodes. If the degree of impurity is small then the attribute qualifies to be a good attribute for splitting and vice versa. For example, if a class distribution is (0, 1) then the class is skewed and there is no impurity, whereas a class distribution of (0.5, 0.5) has the highest possible impurity.

Let $p(i|t)$ denote the fraction of instances belonging to class $i$ at data node $t$. For example, if class 1 has 20 instances and total instances in the dataset is 100 then $p(i|t)$ would be 0.2. Based on this there are three very famous impurity measures:

- Entropy
- Gini Index
- Classification error

3.2.1 Entropy

Entropy is the measure of disorder, chaos or uncertainty. It can be formulated as:

$$
Entropy = - \sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t)
$$

(1)
To understand the equation (1), let us consider a case where there are n outcomes, to be specific n different classes \( \{C_1, C_2 \ldots C_n\} \) with equal probability \( p(C_i) = 1/n \). The uncertainty of a set of n outcomes is \( \log_2 n \). In case of a non-uniform probability distribution the uncertainty would be \( \log_2 P(C_i) \). Therefore on an average the probability distribution would be

\[
- \sum_{i=1}^{n} p(C_i) \log_2 p(C_i).
\]

3.2.2 Gini Index

Gini Index too is a measure of uncertainty or randomness like entropy. It can be calculated using the formula:

\[
Gini(t) = 1 - \sum_{i=0}^{c} [p(i|t)]^2
\]

Higher value of gini index implies more chaos and disorder. An attribute would be ideal for splitting the dataset when the gini index value is lowest for it.

3.2.3 Classification error

Classification error is a simple metric that finds out the impurity in the dataset created. It can be calculated using the formula:

\[
Classification\ Error(t) = 1 - \max[p(i|t)]
\]

Fig. 3.1 gives a good picture of how the different measures compare against each other.
It can be observed that all the measures have maximum impurity values when the class distributions is even i.e. (0.5, 0.5).

3.2.4 Information Gain

Information gain is an improvement over just using entropy to decide the best attribute to split. Information gain is the comparison of impurity of parent node (before splitting) and impurity of child nodes after splitting. Information gain is represented by $\Delta$. It provides the goodness of a split. Higher value of information gain makes a better split. It can be computed using the formula:

$$\Delta = \text{Entropy (Parent)} - \text{Entropy (Child)}$$

In other words it can be calculated as:

$$\Delta = I(\text{parent}) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j)$$
Here $I$ denotes the impurity measure. $N$ is the total number of instances and $k$ is the number of attribute values. $N(v_j)$ is number of records associated with child node $v_j$. When entropy is used as the impurity measure then the difference is called as information gain or $\Delta info$.

### 3.2.5 Gain Ratio

Impurity measures such as entropy or gini index seem to be biased towards attributes that have more instances that are distinct. For example if a customer’s database were to have attributes like gender and age, entropy would have favored age since probability to find skewed child datasets is higher when choosing an attribute with more distinct values like age. However this may not be the best case scenario always. For example if in the same customer’s table if attributes like customer ID and age are present, entropy and gini index would prefer customer ID since it is more probable to give child subsets with purer regions. The regions formed would have very less instances and this would create models that are not desirable. To overcome this problem a new impurity measure called as gain ratio was used by Quinlan e.t. al [3] in the famous C 4.5 algorithm.

Gain ration is also used to quantify the goodness of a split. It can be formulated as follows:

$$Gain \ Ratio = \frac{\Delta info}{Split \ Info}$$

(6)

Split info can be calculated as follows:

$$Split \ Info = -\sum_{l=1}^{k} p(v_l) log_2 p(v_l)$$

(7)

Here $k$ is the total number of splits. If an attribute produces a large number of splits its split info value also will be large as a result the gain ratio will be small. This will remove the problem of choosing attribute with most number of distinct values discussed previously.
3.3 Established Algorithms

Decision tree induction process is a well-researched area and there are several well established algorithms for decision tree induction process. This section provides a brief introduction to various well established decision tree induction algorithms.

3.3.1 CART

One of the first algorithms was proposed by Breiman [26] in 1984 called as “Classification and regression trees” (CART). This algorithms aims to generate data sets that are homogeneous from a dataset that has instances from multiple classes using a binary tree like approach. It is a non-parametric decision tree learning methodology that produces either decision or regression trees based on whether it’s a categorical or a numeric data. CART is also based on Hunt’s rule discussed previously.

Pseudo code:

<table>
<thead>
<tr>
<th>Algorithm: Classification and Regression trees (CART)</th>
</tr>
</thead>
</table>
| • Let $A$ be a feature with domain $A$. Ensure a finite number of binary splits from $X$ by applying the following domain partitioning rules:
  - If $A$ is nominal, choose $A' \subseteq A$ such that $0 < |A'| \leq |A \setminus A'|$.
  - If $A$ is ordinal, choose $a \in A$ such that $x_{\text{min}} < a < x_{\text{max}}$ where $x_{\text{min}}$, $x_{\text{max}}$ are the minimum and maximum values of feature $A$ in $D$.
  - If $A$ is numeric, choose $a \in A$ such that $a = (x_k + x_i) / 2$, where $x_k$, $x_i$ are consecutive elements in the ordered value list of feature $A$ in $D$.
• For node $t$ of a decision tree generate all splitting of the above type.
• Choosing a splitting from the set of splitting that maximizes the impurity reduction $\Delta I$:
  \[ \Delta I(D(t), \{D(t_l), D(t_r)\}) = I(t) - \frac{|D_l|}{|D|} \cdot I(t_l) - \frac{|D_r|}{|D|} \cdot I(t_r) \]
Here $t_l$ and $t_r$ denote the left and right successor of tree $t$.

Algorithm 3. 1: CART
3.3.2 ID3 Algorithm

ID3 algorithm was invented by Quinlan [2] in 1986. It is precursor to famous C 4.5 algorithm. The algorithm goes through every attribute in the set S and finds out the attribute having the lowest entropy. The set S is then split by using the attribute found into two subsets. This algorithm also continues to split the set until homogeneous set of data is found in the child nodes. The ID3 algorithm does not guarantee an optimal as it follows a greedy approach and gets stuck in a local optima. ID3 algorithm also can lead to over fitting as it produces small trees but not the smallest tree. ID3 also is difficult to use on continuous real valued data as for a same attribute there can be many cutoffs and searching for best value to split may be time consuming.

Pseudo code:

<table>
<thead>
<tr>
<th>Algorithm: ID3 (Iterative Dichotomiser 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID3 (D, Attributes, Target)</td>
</tr>
<tr>
<td>1. t = createNode ()</td>
</tr>
<tr>
<td>2. IF ∀x, c(x) ∈ D: c(x) = 1 THEN label(t) = '+', return(t) ENDIF</td>
</tr>
<tr>
<td>3. IF ∀x, c(x) ∈ D: c(x) = 0 THEN label(t) = '-', return(t) ENDIF</td>
</tr>
<tr>
<td>4. Label (t) = mostCommonClass (D, Target)</td>
</tr>
<tr>
<td>5. IF Attributes = ∅ THEN return (t) ENDIF</td>
</tr>
<tr>
<td>6.</td>
</tr>
<tr>
<td>7. FOREACH a ∈ A* DO</td>
</tr>
<tr>
<td>Da = {(x, c(x)) ∈ D: x</td>
</tr>
<tr>
<td>IF Da = ∅ THEN</td>
</tr>
<tr>
<td>t' = createNode ()</td>
</tr>
<tr>
<td>label (t') = mostCommonClass (D, Target)</td>
</tr>
<tr>
<td>CreateEdge (t, a, t')</td>
</tr>
<tr>
<td>ELSE</td>
</tr>
<tr>
<td>CreateEdge (t, a, ID3 (Da, Attributes \ {A*}, Target))</td>
</tr>
<tr>
<td>ENDIF</td>
</tr>
</tbody>
</table>

Algorithm 3. 2: ID3
3.3.3 C 4.5 Algorithm

C 4.5 algorithm was an improvement on ID3 algorithm by Quinlan [3] in 1994. The induction process was very similar. In case of C 4.5 gain ratio was used as the impurity measure instead of using entropy as impurity measure. This removed the drawback of ID3 where attributes with more distinct values were given more preference over other attributes. C4.5 could handle both continuous and discrete valued attributes. In addition C 4.5 could handle missing attribute values. C 4.5 also went back to the tree created and pruned the tree to make it small and concise. C 5 was developed after C 4.5 that offers better speeds and memory usage. Also trees created by C 5 are smaller than that produced by C 4.5.

3.4 Agent Based Decision Tree Induction

An increasing number of computing systems are being viewed in terms of agents. These agents are independent, autonomous and intelligent. However, in most cases these agents do not have a whole picture of the system and often work for individual gains. Some of the key characteristics of agents are:

- They are autonomous meaning they are intelligent and do processing on their own.
- They can collaborate with a master system or with other agents.
- No other agent can interfere in the processing being done by one agent.
- None of the agents have a complete picture of the system as a whole.
- The agents can perceive the environment and react to it.

Agents are often used for parallel processing, which comes in very handy when dealing with distributed systems as it provides a modular approach and also saves time.
3.4.1 Decision tree learning in distributed environment

Let us consider that we have \( n \) databases represented as \( \{D1, D2, ..., Dn\} \) located across \( n \) different sites. Let us also consider that we have an arbitrator in the form of a learner that receives data from different sites and is responsible for integration of this data and further processing if required. There are primarily two different ways in which this distributed learning process can be modelled.

**Sequential learning methodology**

In this type of a model learning is accomplished by only one agent. One agent goes sequentially to each of the \( n \) different data sites and gathers statistical data required for induction process and sends these to the negotiator/ arbitrator which can be a totally independent component or even located at one of the data nodes.

![Sequential Learning](image)

**Fig. 3.2:** Sequential Learning

Fig. 3.2 shows a sequential decision tree learning methodology where an agent traverses through all the data sites \( \{D1, D2, ..., Dn\} \), collects required data and sends this data to a learner site \( h \) where the consolidated model can be made. This would be a very time consuming process.

**Parallel learning methodology**

This model of learning is accomplished by using multiple agents. An agent is located in each of the participating data sites. The individual agents are responsible for collecting statistical data
required for the induction process and forwarding it to a centralized component. This parallelizes the whole induction process and thus reduces the overall time required.

Fig. 3.3 shows a parallel learning process where an agent is associated with each of the data sites. These agents are responsible for collection of statistical data from their own individual sites. They are also responsible for doing all the required pre-processing in the individual data sites. After collection of all this data, it is forwarded to a central arbitrator that integrates all this data and does required further processing.

![Parallel Learning Methodology](image)

Fig. 3.3: Parallel Learning Methodology

Any distributed learning algorithm has three distinct phases of processing:

- **Local processing phase:** Operations are performed on individual data sites. Data cleaning, re-organization may take place in the phase.

- **Intermediate phase:** The results obtained from the local processing phase are post-processed and anomalies if any are removed.

- **Final processing phase:** Results are aggregated from multiple data sites and further processing is done.
3.4.2 Negotiation among MAS for quality improvements

Negotiation as discussed in chapter 2 can be a useful mechanism to find a common ground when dealing with multiple agents. We use the parallel learning mechanism to construct the decision tree in a distributed environment. This means that a single tree needs to be created from the summaries of multiple agents. We utilize auction based negotiation methodology that allows an individual agent to accomplish its goal instead of worrying about the global objective. All agents try to increase their profits and the negotiator tried to improve the overall performance by checking if the individual bids made by independent agents are acceptable to other agents involved in the negotiation.

Fig. 3.4: Centralized negotiation mechanism

Fig. 3.4 shows a centralized negotiation mechanism with each of the individual data sites sending bids to the centralized learner site that negotiates with all the other agents. When a bid is proposed...
by any of the data sites, it is reviewed and sent across to all the other data sites participating in the induction process. If the submitted bid is better or doesn’t change their objective it is retained in the priority queue. However if the bid is detrimental to the objective of majority of the agents then it is rejected. Bids are prioritized based on their acceptance by various data sites and this process continues until all the data sites are out of bids i.e. a modification can no longer help improve the local objectives.

The next chapter will cover this aspect in more detail and will show how a negotiation mechanism is implemented on a distributed decision tree induction process.
Chapter 4

4. Our Approach

Chapter 1 gives an introduction to classification, decision trees and how game theory can be used in distributed environment to create a consensus and build a stable decision tree from distributed data. We discuss about the motivation of our work and outline the process that we will be adopting. Chapter 2 gives a good background on classification and specifically decision trees. We discuss about the basics in game theory and where it is headed. We talk about multi-agent system and see how they can be used for a distributed induction process. We finally review some research on negotiation and see how negotiation is being used to generate consensus amongst agents with conflicting interests. Chapter 3 details the decision tree induction process in a centralized environment. This chapter will detail our approach where competing agents will use an auction bases bidding mechanism to find a common ground and build a decision tree acceptable to all. We will elaborate on the process we adopt and then discuss the results we achieve in the next chapter.

4.1 Overview

This work aims at proving an alternative approach for building distributed decision trees. The work done creates trees that have a classification accuracy very similar to the ones that would have been created if all the data was not scattered but located centrally in one database. The approach proposed in this work has the following characteristics:

1. Real data does not need to be distributed between different data sites, thus removing the security and privacy concerns.

2. A huge central repository of data need not be created to run decision making algorithms, thus removing the requirement of having sophisticated storage hardware and software.
3. Data centralization will no longer be required, this will save a lot of time. Also running algorithm in a distributed fashion in parallel will result in a quicker induction process than building a tree on whole aggregated data.

4. Skewed data sets having majority of instances belonging to once class can be modelled correctly to learn patterns that point to instances belonging to minority classes.

5. Intermediate interpretable models can be created, that can allow for an early check on whether the rules generated “make sense”. This can also help understand the dominant features in a data site.

6. The negotiation mechanism employed leads to creation of tree that can represent a model which can classify the instances more accurately and concisely.

This algorithm proposed here uses a bottom-up approach of merging independent decision trees created at different data sites. A lot of pre-processing steps are undertaken to identify rules/patterns that can correctly classify a given instance. A process of bidding between the intelligent agents located at every remote data site ensures that the final decision tree built for all the data scattered is able to provide a good classification accuracy.

4.1.1 Objective
The main objective/aim is to build a tree $L_d$ from distributed data $\{D_1, D_2, \ldots, D_n\}$ such that it is almost as similar as possible to a tree $L_c$ obtained when a decision tree is induced from $\{D_1, D_2, \ldots, D_n\}$ combined.

4.2 Preliminaries
This subsection will define and describe some of the terms that will be used throughout this chapter.
4.2.1 Rule

A rule can be considered as a conjunction of many conditions based on attributes and their cutoffs. In other words, a set of decisions leading to a class can be considered as a rule. A rule is used to represent a pattern.

**Definition:** A classification rule is a function $h$ applied on data instances $x$ that can be defined as a conjunction of decisions $\{(a_1, v_1), (a_2, v_2), \ldots, (a_i, v_i)\}$ where $a$ is the attribute, and $v$ is the value associated with the attribute such that $h(x)$ points to a label $Y$ which is as similar as possible to the original label $Y'$.

The true class $Y'$ may not always be the same as the predicted class $Y$. Number of decisions in a rule define the length of a rule. A rule is considered good if the predicted class and actual class are the same.

4.2.2 Rule quality

Quality of a rule is measured in terms of the number of the following criterion:

- Number of instances a rule captures.
- Class distribution of the majority class in the data-set captured by the rule.
- Weight of the majority instance’s class captured in terms of availability of instance’s class.

Weight of an instance’s class can be calculated by using the following formula:

$$\text{Class Weight} = \frac{1}{\frac{\text{Number of instances belonging to the class in the data set}}{\text{Total number of instances in the data set}}}$$

(1)

Or

$$\text{Class Weight} = \frac{\text{Total number of instances in the data set}}{\text{Number of instances belonging to the class in the data set}}$$

(2)
Let total number of instances in the entire database be $N$. Let number of instances captured by rule be $D$. Let the number of instances belonging to majority class be $X$ that are captured by the rule.

The rule quality can be given by:

$$Rule\ Quality = \frac{D}{N} \times \frac{X}{D} \times Class\ Weight$$  \hspace{1cm} (3)

Or

$$Rule\ Quality = \frac{X}{N} \times Class\ Weight$$  \hspace{1cm} (4)

Let us consider an example, to understand how the rule quality is measured. Table 1, below shows a data set captured by rule $R$. Let us also consider that the total number of instances in the database is 50. Number of elements belonging to class 1 is 40 and that of class 2 is 10. The quality of the rule can be calculated by taking product of the three criterion mentioned above.

<table>
<thead>
<tr>
<th>Attribute 1</th>
<th>Attribute 2</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>26</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>15</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.1: Snapshot of dataset captured by a rule

In this case of rule $R$, the rule quality is measured in the following way:
Therefore the rule quality will be better if:

- The number of instances captured by the rule is large.
- The class distribution of the data set captured by the rule is skewed.
- The class weight of the instances captured is high.

The rule quality metric discussed here is class sensitive, thus this will be very helpful in those cases where the dataset is skewed meaning the dataset has majority of instances belonging to one class and very few instances belong to other classes. Instances of minority classes will be given more weightage thus helping in finding patterns that can also classify instances with a small representation.

### 4.3 System design

In this section we discuss the design adopted for the decision tree induction process. The algorithm uses a multi agent system design comprising of multiple intelligent autonomous agents. A parallel learning methodology as discussed in chapter 3 is used for learning the decision model.

This system is designed for horizontally partitioned data, meaning all the data sites that are scattered have the same attributes. The system is designed to handle any number of data bases to participate in the induction process. An intelligent agent is associated with every dataset.
participating in induction process. The agent is autonomous and has knowledge about its local database only and doesn’t have the picture of the system as a whole.

A centralized negotiator/arbitrator component is an intelligent agent that is responsible for aggregating data from multiple other agents and also communicating with them. No two agents communicate with each other, but all the agents only communicate with the negotiator.

Fig. 4.1 shows a simple system design, where the negotiator is the only agent that communicates with all other agents associated with each of the datasets. Agents associated with databases \{D_1, D_2…D_n\} create decision tree models at their individual sites. These decision tree models are sent to the negotiator component which then uses a negotiation based auctioning/bidding methodology to improve the rule quality. The bidding process terminates when none of the databases are able to create a new bid that increases their local gains.
4.4 Decision trees to rule sets
Decision tree is a conjunction of decisions that lead to a fixed outcome and structured in the form of a tree. Decision tree is often used to capture a pattern which can help give identity to a section of instances.

Decision trees can also be looked at as a set of rules when traversed from the root to the leaf level. All non-leaf nodes including the root node can be considered as conditions which aim to refine the data and the leaf nodes are the outcome of these decisions. Let us consider the tree as in Fig. 4.2.

![Sample decision tree](image)

**Fig. 4.2: Sample decision tree**

Here A, B, C, D, E, F and G are the attributes and values adjacent to them are the cutoff values. These attributes and cutoffs lead to classes at the leaf level. On traversing from the root A to all the leaf nodes following rule set can be extracted:
4.5 Rule set pre-processing

Once the rule set is generated, the rule set is pre-processed to remove any anomaly. Anomaly can be in the form of repeated rules or rules that are very similar to one another. In this section we will discuss the approach used to remove these anomalies. The pre-processing step takes place at the central arbitrator component once it has aggregated all the rules. The anomalies are dealt in the following way:

- **Duplicate rules**

  If the rule set created out of the decision tree built has rules that have the same attributes and the same cutoff values and point to the same class then the rule is removed. For each rule, a scan is done throughout the rule set to see if same rule exists.

- **Similar rules**

Figure 4.3: Rule set derived from decision tree

Once the rule set is generated from the decision tree, all the rules are sorted according to the attribute order. This is done to maintain an order and reduce chances of anomalies.
Rules that are very similar to each other are merged. Two rules are considered similar when following criterion matches:

- The two rules have the same attributes in their rules.
- Both the rules point to the same class.
- The cutoff values of the two rules overlap and do not have distinctly different ranges. Example 0.4<A<5 and 0.1<A<6 have overlapping ranges so they can be merged, but 0.4 < A < 5 and 7<A<9 do not have overlapping ranges and thus can’t be merged.

Let us consider two Rules R1 and R2 mentioned below:

\[ R1 = A > 2 & B < 3 & C > 10 & D < 2 \rightarrow \text{Class A} \]

\[ R2 = A > 10 & B < 0.5 & C > 10 & D < 1 \rightarrow \text{Class A} \]

R1 and R2 are dissimilar since the attributes differ in more than three attributes.

Let us consider another case:

\[ R1 = A > 2 & B < 3 & C > 10 & D < 2 \rightarrow \text{Class A} \]

\[ R2 = A > 2 & B < 3 & C > 10 & D < 2 \rightarrow \text{Class B} \]

Though R1 and R2 are identical, but they point to two different classes, hence can’t be called as similar rules.

Let us consider a third case:

\[ R1 = 2 < A < 10 & 3 < B < 7 & 0.2 < C < 1.2 \rightarrow \text{Class A} \]

\[ R2 = 1 < A < 1.5 & 2 < B < 2.5 & 3 < C < 4 \rightarrow \text{Class A} \]
Though R1 and R2 differ only at three different attributes, the cutoff values do not overlap. Since the cutoff values do not overlap, the rules can’t be termed as similar.

Let us consider one last case:

\[ R_1 = 2 < A < 10 \& 3 < B < 7 \& 0.2 < C < 1.2 \rightarrow \text{Class } A \]

\[ R_2 = 1 < A < 9 \& 1 < B < 5 \& 0.1 < C < 1.5 \rightarrow \text{Class } A \]

These two rules can be considered as similar, as it meets all the conditions of similarity defined above.

4.5.1 Rule merging

Two rules that are similar can be merged by doing a simple union operation on the cutoff values. For example: let us consider the two rules R1 and R2 such that:

\[ R_1 = 2 < A < 10 \& 3 < B < 7 \& 0.2 < C < 1.2 \rightarrow \text{Class } A \]

\[ R_2 = 1 < A < 9 \& 1 < B < 5 \& 0.1 < C < 1.5 \rightarrow \text{Class } A \]

These two rules are similar, and can be clubbed after merging to form one rule. The merged new rule would be:

\[ R = 1 < A < 10 \& 1 < B < 7 \& 0.1 < C < 1.5 \rightarrow \text{Class } A \]

After the two similar rules are merged, the old rules R1 and R2 are replaced by the new rule. It is observed that the quality of rule increases since the new rule captures more instances. One drawback of rule merging is that on merging of rules, the new rule can also take in some error in the form of instances of minority classes. A user defined error acceptability value is provided that can stop the merging process if the rule merging takes in lot of errors.

Thus it can be concluded from the above example that:
If two rules $R1$ and $R2$ are such that $R1$ is $(x_1 < A < x_2)$ & $(y_1 < B < y_2)$ & $(z_1 < C < x_2)$ $\rightarrow$ $Class$ $1$ and $R2$ is $(x_1' < A < x_2')$ & $(y_1' < B < y_2')$ & $(z_1' < C < x_2')$ $\rightarrow$ $Class$ $2$ then the combined rule will have following ranges:

$$A \rightarrow \{ (x_1, x_2) \cup (x_1', x_2') \}$$

$$B \rightarrow \{ (y_1, y_2) \cup (y_1', y_2') \}$$

$$C \rightarrow \{ (z_1, z_2) \cup (z_1', z_2') \}$$

4.6 Communication with the centralized negotiator

<table>
<thead>
<tr>
<th>Algorithm 1 Master Rule Set Creation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective:</strong> Create master rule set from independent decision tree</td>
</tr>
<tr>
<td>1: $localRuleSet \leftarrow localDecisionTree$</td>
</tr>
<tr>
<td>2: $mRuleSet \leftarrow Union(localRuleSet)$</td>
</tr>
<tr>
<td>3: while $mRuleSet \neq \emptyset$ do</td>
</tr>
<tr>
<td>4: $currentRule \leftarrow \emptyset$</td>
</tr>
<tr>
<td>5: sortRules($currentRule$)</td>
</tr>
<tr>
<td>6: if $detectDuplicate(currentRule)$ then</td>
</tr>
<tr>
<td>7: removeRule($currentRule$)</td>
</tr>
<tr>
<td>8: end</td>
</tr>
<tr>
<td>9: if $detectSimilar(currentRule)$ then</td>
</tr>
<tr>
<td>10: mergeRules($currentRule$, $similarRule$)</td>
</tr>
<tr>
<td>11: end</td>
</tr>
<tr>
<td>12: label($currentRule$)</td>
</tr>
<tr>
<td>13: $currentRule.next \leftarrow currentRule$</td>
</tr>
<tr>
<td>14: end</td>
</tr>
</tbody>
</table>

Algorithm 4.1: Master Rule set Creation

Algorithm 4.1 provides an insight into the process of master rule set creation. As discussed earlier individual decision trees are grown at remote data sites that may be distributed across different geographic locations. These decision trees are converted into local rule sets as described in rule1. The system is designed such that the agents located at each database do not communicate with each other, but only communicate with the centralized negotiator component. All these local rule sets are aggregated by the negotiator and merged to create a master rule set. The rules are first sorted according to the order in which the attributes are present in the dataset as described in line
5 in algorithm 4.1. These rules are processed to remove any conflicts. Line 6-8 says how each rule is compared with all other rules in the master rule set and checked for duplicity. If a duplicate rule is found it is removed from the master rule set. Then a rule is picked up and it is compared with all other rules to see if any similar rule exist as in line 9-11 in algorithm 4.1. If similar rules are found then those rules are merged. The merging process takes place in the same way as described in the previous sections. New merged rules created replace the old rules in the master rule set. Each rule in the master rule set is labelled for rule identification as in line 12. One drawback of rule merging is that on merging of rules, the new rule can also take in some error in the form of instances of other classes. A user defined error acceptability value is provided that can stop two rules from merging if the incoming error value exceeds the set cutoff.

Algorithm 2 RuleQuality Evaluation

Objective: Calculate rule quality of rules from master rule set

for agent in agentSet do
  for R in mRuleSet do
    pickup(agent,R)
    applyRule(agent,R)
    ruleMetric ← calculateRuleQuality(agent,R)
  end
end

Algorithm 4.2: Rule Quality Evaluation

Algorithm 4.2 shows the process after the master rule set is created. Line 3-4 shows how the intelligent agent associated with each of the data sets picks up rules from the master rule set and applies all the rules to the individual datasets and finds out the rule quality based on the rule quality metric discussed previously. Only those rule sets are picked up that are not generated by it. Every rule is associated with a rule quality value. Each of these intelligent agents creates a matrix of rule quality values for their own data sites and returns this matrix to the negotiator.
To sum up the process till now let us consider the Fig. 4.4 below. The schematic diagram represents a system with three different data sites and a central negotiator.

The negotiator on receiving the rule sets from different datasets, does a weighted sum of the rule quality scores for each individual rule. Based on the cumulative value of the rule quality, the master rule set is sorted. Rules that have a better score are given higher priority and placed at beginning of a priority queue while the ones having a lower score are placed at the very end.

Fig. 4.4: Master rule set creation

4.7 Negotiation for rule quality improvement

In the previous chapters we have discussed the usage of game theory to resolve conflicts amongst multiple agents in a distributed environment. We have seen how game theory specifically negotiation can be used as a tool to consensus building. We discuss in breadth about how multiple agents can be used for the process of distributed decision tree induction. We implement an auction based bidding mechanism with a central arbitrator for the purpose of negotiation between multiple
agents involved in the process of decision tree building. We propose an approach here where multiple agents do not transfer real data but share rules or patterns from individual data sites with a centralized arbitrator which involved in the process of negotiation.

---

**Algorithm 4.3 Negotiator Operations**

**Objective:** Central negotiator operations for negotiation

1. // Flag rules rejected by majority of rules.
2. for agent in agentSet do
3.   for R in pRuleSet do
4.     rejectedByMajority(R) = consensus(reject(agent,R))
5.   end
6. end
7. sendToAgentRejectedRule(R)
8. // Get rule metric from agents and combine modified ruleset with master ruleset.
9. ruleQuality ← ruleMetric
10. getFromAgents(modRuleSet)
11. pRuleSet ← sortByRuleQuality(mRuleSet, ruleQuality)
12. if modRuleSet ≠ ∅ then
13.   for R in pRuleSet do
14.     if rejectedByMajority(R) then
15.       remove(R)
16.   end
17. end
18. pRuleSet ← Union(pRuleSet, modRuleSet)
19. end
20. rePrioritize(pRuleSet)

---

Algorithm 4.3: Negotiator Operations for Negotiation

Algorithm 4.3 details the role of negotiator in the process of tree induction. Line 2 – 6 show how the negotiator takes the consent from all the datasets regarding the quality of a rule. Then it checks if a rule is rejected by majority of datasets. If this is the case the negotiator notifies all the agents about the rejected rules. Line 9 – 19 describe the role of negotiator once it receives rule quality metric and modified rule set from the agents. All these operations are described in this section.

All the intelligent agents associated with different datasets pick up the rules from the master rule set and try to change the cutoff i.e. either increase or decrease the cutoff values of each of the attribute present in the rule.
Each rule in the master rule set is associated with a rule label. This rule is picked up and modified by changing the cutoff values. The modified rule is associated with the same rule label as that of the original rule. The modified rule is applied on to its own local dataset and rule quality is calculated. If the modified rule has a better rule quality than that of the rule having the same rule label, then that rule is replaced by the modified rule in the master rule set. Same process is utilized by all the other agents participating in the negotiation process. Negotiator receives bids from all the agents and creates a priority queue of all the rules based on the net rule quality. The net rule quality is computed by taking the weighted average of all the rule quality values received from different data sites. This process continues until, no more modifications by any agent will lead to better rules.

The user has the ability to set the percentage of modification that can be done on the cutoff values. In addition, changing cutoff and modifying the rule may lead to creation of rules that may have some instances that do not belong to the majority class or that may lead to classification error in future. To keep a check on that the user can also set a percentage for acceptable incoming error.
Algorithm 4.4: Agent operation for negotiation

Objective: Individual agent’s operations for negotiation

1: for agent in agentSet do
2:     for R in pRuleSet do
3:         pickup(agent, R)
4:         OldMetric ← calculateRuleQuality(agent, modRule)
5:         if betterThanPrevious == False then
6:             True ← reject(agent, R)
7:         end
8:         if isRuleRejectedByMajority(R) then
9:             ignoreRule(R)
10:        end
11:     if isRuleRejectedByMajority(R) == False then
12:         modRule ← modify(agent, R)
13:         NewMetric ← calculateRuleQuality(agent, modRule)
14:         if NewMetric > OldMetric then
15:             label(modRule)
16:             modRuleSet ← modRule
17:         end
18:     end
19: end
20: if endOfIteration == True then
21:     sendToNegotiator(modRuleSet)
22: end

Algorithm 4.4: Agent operation for negotiation

Algorithm 4.4 describes the operations of an agent during the negotiation process. Line 3-7 show how all the agents pick up rules from the central negotiator and apply them on their individual datasets. The rule quality is calculated and if it is found that the rule is worse than a rule with same base level then it is labeled as rejected. This is sent to the negotiator. The negotiator notifies the agent about the decisions made by other participating agents. If a rule is rejected by majority of the agents it is neglected else it is modified by the agent and rule quality is again checked. This process of negotiation is elaborated in the section below.
Fig. 4.5 shows how negotiation process is implemented for a single rule. After creation of a master rule set, it is sorted based on the rule quality metric. The rule with best rule quality value is added to the beginning of the set. A rule is extracted from the rule set and it becomes a proposal for negotiation. The rule is picked up by all the agents and is applied on associated datasets. If the rule is not acceptable by majority of agents then it is ignored. A rule becomes unacceptable if the calculated rule quality metric for the rule has a value less than the quality of a rule with same rule label. If it is acceptable to majority of agents, then the rule is considered as a candidate for modification. If any agent participating in the process deems that a better rule can be obtained by modifying the current rule, then it modifies the rule and a new rule is proposed. A rule can be
modified by changing the cutoff value of each of the attributes present in the rule. The cutoff values can either be increased or decreased by a fixed percentage set by the user. The new proposed rule is stored in a modified rule buffer. Next rule in the master rule set becomes the candidate for next proposal. This process continues and number of newly modified rules keeps growing in the buffer. Once all the agents have finished picking up all the rules from the master rule set, the modified rule set is transferred to the master rule set.

Master rule set is updated by the modified rule buffer. Rules in the modified rule buffer replace those rules in the master rule set which have the same rule labels. The replaced rules are stored in the old rule storage buffer. All other rules remain as it is. If all the new rules added to the master rule set belonging to one base label are rejected by majority of agents then the old rule having the same base label in the old rule buffer replaces the newly added rules in the master rule set.

This process continues and the master rule set keeps on getting updated with new better rules. This process stops when the modified rule set buffer is empty, or in other words the rules cannot be modified anymore to generate better rules. The master rule set obtained at the end of the negotiation process acts as the final rule set.

There can be a possibility of having more than one rule with the same rule label. This would arise because a single rules will have to go through a series of modifications. In this due process of modification, an attribute’s value in a rule can be modified in at least two different ways. If there are k attributes in a rule then, this would mean that a single rule can be modified into 2 n-1 * k different rules where n represents the number of iterations. To elaborate this process let us consider a simplistic process of a rule having one attribute in Fig. 4.6 below.
Fig. 4. 6: Rule growth for a single rule

Fig. 4.6 shows that rule R1 present in the master rule set is modified and R1-1 and R1-2 are formed. R1-1 and R1-2 get stored in the modified rule buffer. Since both R1-1 and R1-2 have a better rule quality than R1, they replace R1 from the master rule set. R1-1 and R1-2 currently in master rule set are picked up by agents and modified into R1-1-1, R1-1-2, R1-2-1 and R1-2-2 respectively. If R1-1-1 and R1-1-2 have a better rule quality than R1-1 then R1-1 will be replaced by R1-1-1 and R1-1-2 in the master rule set. By the end of the negotiation process, we will have at most 2n-1 rules originating from R1 where n is the number of iterations. Rules originating from the same base rule i.e. R1 in this case will all be grouped together. The rule that has best rule quality value from this group will be chosen as one of the final rules that make up the combined decision tree.

4.8 Decision tree synthesis from decision rules

Once the final rule set is obtained by the negotiation process, the rule set is transformed back into a decision tree structure. We follow the AQDT-1 algorithm [46] proposed by Imam et al. and outline its main steps here. The first step in the structuring process is to find the best root attribute. The best root attribute is selected based on the following criteria: disjointness: a metric that says how effective an attribute is in discriminating rules belonging to different classes, dominance:...
relevance of an attribute calculated by counting its occurrence across all the rules in the master rule set, *extent*: provides number of different values an attribute has across the master rule set.

According to Imam et al. [46] the class disjointness for an attribute $A$ can be calculated as follows:

Disjointness $D(A, C_i)$ for a rule set pointing to class $C_i$ is sum of disjointness $D(A, C_i, C_j)$ between rule sets pointing to $C_i$ and those pointing to $C_j, j = 1,2,3 \ldots m, j \neq i$, where $m$ represents the number of classes. Let $V_1, V_2, \ldots, V_m$ denote the value sets of attribute $A$ that are present in rules pointing to classes $C_1, C_2, \ldots, C_m$. The disjointness between $C_i$ and $C_j$ can be calculated by:

$$D(A, C_i, C_j) = \begin{cases} 
0, & \text{if } v_i \subseteq v_j \\
1, & \text{if } v_i \supseteq v_j \\
2, & \text{if } v_i \cap v_j \neq \emptyset \text{ or } v_i \text{ or } v_j \\
3, & \text{if } v_i \cap v_j = \emptyset 
\end{cases}$$

$\emptyset$ represents an empty set.

Therefore the disjointness for attribute $A$ can be formulated as:

$$D(A) = \sum_{i=1}^{m} \sum_{j=1, j \neq i}^{m} D(A, C_i, C_j)$$

An attribute has highest disjointness when it occurs in every rule and has a different value in every one of them. Dominance is calculated by going through all the rules in the master rule set and counting the appearance of an attribute in each of them. Extent is calculated by traversing all the rules and finding out the number of different value ranges for an attribute. Number of branches that an attribute will lead to is determined by its extent. An attribute with higher disjointness and dominance is highly ranked, while a higher extent score negatively impacts the attribute ranking.
A combined ranking is calculated based on the “lexicographical evaluation function with tolerances” by Michalski [47]. First, the attributes are evaluated based on their disjointness, if there is a tie then the dominance is used as a criterion for choosing the best attribute. Even here if there is a tie then the extent is used to rank the attributes. An attribute with higher extent value is given a lower rank.

The above discussed criteria are combined to get a cumulative attribute rank. We illustrate the usage of the above defined criteria for best attribute selection using the following example:

\[
R_1 := A = 2 & B = 2 \rightarrow C1
\]

\[
R_2 := A = 3 & 1 < C < 3 & D = 1 \rightarrow C1
\]

\[
R_3 := 1 < A < 2 & 3 < B < 4 \rightarrow C2
\]

\[
R_4 := A = 3 & 1 < C < 2 & D = 2 \rightarrow C2
\]

Figure 8: Rules to structure into trees

For the above mentioned rules, the attribute rank is as follows:

<table>
<thead>
<tr>
<th>Class</th>
<th>Attributes</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Values</td>
<td>2,3</td>
<td>2</td>
<td>1..3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Disjointness</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C2</td>
<td>Values</td>
<td>1..3</td>
<td>3..4</td>
<td>1..2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Disjointness</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Attribute Disjointness</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Attribute Dominance</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Attribute Extent</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1: Attribute metric calculation
Based on the above table we can see that A has the highest values for disjointness and dominance, thus A is chosen as the root of the tree. Number of branches that A has will be determined by its extent. An attribute having a large extent value will lead to many branches. This might become a problem if the tree needs to be made compact and small. To avoid many branches from a single attribute we remove value ranges that subsume other value ranges. Example if attribute A has following values: \{1,5\}, \{1,3\}, \{3,5\}, \{10,15\} we remove \{1,5\} as a value since it subsumes \{1,3\} and \{3,5\}. Once the best attribute is selected the steps involved in the AQDT1 algorithm as detailed by Imam et al. [46] are as follows:

- **Step 1:** Go through each rule and select the highest ranked attribute. Let us consider that attribute to be A.

- **Step 2:** Create a tree node and assign the attribute A to be the root. Number of branches from attribute A will be the value of its extent.

- **Step 3:** Associate a rule that satisfies the branch condition to each of the branches coming out of the root. For example if there is a branch from attribute A having value $V_i$ then associate that branch with all the rules that contain attribute A and have value $V_i$ in them. Remove from the rules these conditions and choose the next best attribute in the section of the rules that satisfy the precondition of the branch. Continue this process.

**Note:** If there are rules that do not contain attribute A at all then associate those rules with all the branches originating from the root attribute, A in our case. This is justified by the consensus law: $[X = 1] \equiv \{[X = 1] \& [Y = 2] \cup [X = 1] \& [Y = 3]\}$

- **Step 4:** If all the rules in the branch point to the same class then create a leaf node and assign it to that class label.
Chapter 5

5. Results and discussions

The previous chapter detailed the approach we used for decision tree induction in a distributed environment. We showed how multiple agents implement an auction based negotiation protocol and compete with each other to reach at an equilibrium for each of the rules. These rules are then structured into a tree and this tree becomes the final tree merged out of the individual trees. This chapter evaluates the performance of the algorithm by running it on several datasets and comparing its classification accuracy with that of an algorithm that runs on a dataset formed by aggregation of all the distributed databases.

5.1 Experimental setup

The algorithm developed was tested on 4 different datasets from UC Irvine machine learning repository [29]. Matlab 7 was used for development of the algorithm. The datasets were horizontally partitioned, structured in a tabular format and distributed across 3 different data sites. 70% of the data was reserved for training while 30% of the data was reserved for testing. Two sets of experiments were done, one on randomly sampled training data and other on a skewed training data. Each of these experiments was repeated 10 times and the average classification accuracy was calculated.

A classification tree was built on the aggregated data collected from different data sites for the purpose of comparison. The tree used information gain as the impurity measure for attribute selection and was built using an open source free data mining suite “Orange Canvas” [30]. The
result of classification accuracy obtained is compared with the classification accuracy of our distributed decision tree induction algorithm.

5.2 Experimental objective

The main aim of this work was to grow multiple decision trees at different remote sites and combine them using a negotiation framework and evaluate their performance. An attempt was made to create rules that could discover patterns belonging to minority instances even in skewed datasets. The work done shows promise in combing game theoretic framework of negotiation with distributed decision tree induction process. It was observed that classification accuracy of the distributed method proposed here is very close to the accuracy of a decision tree building algorithm used on a huge dataset formed by combination of the distributed datasets. The algorithm also performs well when the distributed datasets are skewed.

5.3 Datasets Used

All the experiments were performed on multi-variate, real valued continuous data. All the class labels had numerical values. The datasets used for the purpose of experiment are as follows:

5.3.1 Seeds dataset

This dataset provides measurements for kernels belonging to three different varieties of wheat [31]. It has 7 real valued attributes and 210 continuous valued instances. This dataset has been used for purposes of classification and clustering. All the instances are labelled into 3 different classes.
5.3.2 Wine dataset

Wine dataset provides chemical analysis that can predict the origin of wine [32]. It has 13 real valued attributes and 178 continuous valued instances. All the instances are classified into 3 different classes. All the classes have numerical labels.

5.3.3 Pima Indian diabetes dataset

Pima Indian dataset was donated by national institute of diabetes and kidney disease. It aims at finding out the possibility of a person having diabetes based on several attributes. This dataset has 8 real valued attributes and 768 continuous valued instances. There are 2 class labels. There are 500 class 1 instances and 268 class 2 instances.

5.3.4 Banknote authentication dataset

Banknote authentication data was extracted from images that were taken for evaluation of the authenticity of the data [33]. This dataset has 5 different real valued attributes and 1372 continuous valued instances. It has 2 classes which have numerical labels.

Table 5.1 provides the summary of the datasets being used.

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Number of Classes</th>
<th>Attributes</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeds Dataset</td>
<td>3</td>
<td>7</td>
<td>210</td>
</tr>
<tr>
<td>Wine Dataset</td>
<td>3</td>
<td>13</td>
<td>178</td>
</tr>
<tr>
<td>Pima Indian diabetes Dataset</td>
<td>2</td>
<td>8</td>
<td>768</td>
</tr>
<tr>
<td>Banknote authentication Dataset</td>
<td>2</td>
<td>5</td>
<td>1372</td>
</tr>
</tbody>
</table>

Table 5.1: Dataset Summary
5.4 Results and analysis

In this section we show the performance of our algorithm in terms of classification accuracy it achieves with respect to a centralized algorithm running on aggregated data. The first part of our analysis deals with datasets that are created by random selection of instances from the dataset source. The second part of our analysis is on skewed datasets that have an overwhelming majority of instances belonging to one class and few instances belong to other classes.

5.4.1 Analysis on randomly selected data

We ran our algorithm on the seeds dataset. 70% of the data is utilized for the purpose of training and testing is done on the rest of 30% of data. There are totally 147 training instances distributed uniformly across 3 different data sites, each having 49 instances. 63 instances are reserved for the purpose of testing. We run the algorithm for varying percentages of purity at leaf level. The amount of change that can be made by an agent on a rule is set at 5% and the acceptable incoming error is also set at 5%. We conduct 10 such runs. Fig. 5.1 shows the percentage accuracy comparison between our method and the centralized method where data is aggregated and decision tree algorithm is built on it.
Fig. 5. 1: Classification accuracy comparison in seeds dataset

Figure 5.2 shows the behavior of our algorithm against a sequential algorithm at different leaf level purities.

Fig. 5. 2: Error comparison on seeds dataset
It can be observed from the following analysis that though our algorithm doesn’t outperform the centralized algorithm, but the accuracies obtained are very near to the ones obtained from the distributed algorithm.

We next look at the wine dataset from which instances are randomly selected and shuffled. 70% of the data is utilized for the purpose of training and testing is done on the rest of 30% of data. There are totally 126 training instances distributed uniformly across 3 different data sites, each having 42 instances. 52 instances are reserved for the purpose of testing. We run the algorithm for varying percentages of purity at leaf level. The amount of change that can be made by an agent on a rule is set at 5% and the acceptable incoming error is also set at 5%. We conduct 10 such runs. Fig. 5.3 shows the percentage accuracy comparison between our method and the centralized method where data is aggregated and decision tree algorithm is built on it.

![Wine Dataset: CAC Comparison](image)

**Fig. 5.3:** Classification accuracy comparison in wine dataset
The error comparison between the two algorithms can be better analyzed by looking at their graphical trend. Fig. 5.4 shows the comparison between the two algorithms in terms of the misclassifications.

![Graph showing misclassification in wine dataset](image)

**Fig. 5.4: Misclassification in wine dataset**

We ran our algorithm on the Pima Indian’s diabetes dataset. There are totally 537 training instances distributed uniformly across 3 different data sites, each having 179 instances. 231 instances are reserved for the purpose of testing. We run the algorithm for varying percentages of purity at leaf level. The amount of change that can be made by an agent on a rule is set at 2% and the acceptable incoming error is also set at 2%. We conduct 10 such runs. Fig. 5.5 shows the percentage accuracy comparison between our method and the centralized method where data is aggregated and decision tree algorithm is built on it.
Fig. 5.5: Classification accuracy in Pima Indian dataset

Fig. 5.6 shows the comparison in error in classification between our algorithm and the centralized algorithm.

At last we look at the banknote authentication dataset. There are totally 960 training instances distributed uniformly across 3 different data sites, each having 320 instances. 411 instances are
reserved for the purpose of testing. We run the algorithm for varying percentages of purity at leaf level. The amount of change that can be made by an agent on a rule is set at 5% and the acceptable incoming error is also set at 2%. We conduct 10 such runs. Fig. 5.7 shows the percentage accuracy comparison between our method and the centralized method where data is aggregated and decision tree algorithm is built on it.

![Banknote authentication dataset: CAC Comparison](image)

**Fig. 5.7:** Classification accuracy in banknote authentication dataset

The difference in error trends across different leaf level purities can be observed from Fig. 5.8. It can be seen that though our method doesn’t outclass the centralized method but the accuracies are more or less the same. All these observations confirm our first objective that combination of distributed decision tree induction process with a negotiation based bidding methodology leads to creation of better rules that improve the overall accuracy of a distributed induction process.
5.4.2 Analysis on skewed datasets

We examine the behavior of our algorithm on datasets that are skewed, i.e. datasets having an overwhelming number of instances belonging to one class and a few instances belonging to other classes. This is done to test our second objective of finding out patterns in datasets which have a very small representation. We create skewed datasets by having 70% representation of majority class and 30% representation of rest of the classes. Similar to the first part of analysis we run our algorithm on skewed datasets made from the above mentioned datasets.

We ran our algorithm on seeds dataset following the same model of 70% training data and 30% testing data. The training data was distributed across 3 different data sites. All the data sites had different class distributions.

Fig. 5. 8: Classification error in banknote authentication dataset
The first data set we used to test our algorithm was the seeds dataset. It had 147 training instances spread across 3 different data sites. In the first data site, class 1 was the majority class comprising of 70% of the instances and rest were equally divided between class 2 and 3. Similarly in the second data site, class 2 had the majority representation and in the third data site class 3 was the majority class with class 1 and 2 having minority representations. 63 instances were set aside for testing purposes. The amount of change permitted in the negotiation process was 5% and the acceptable incoming error was also 5%. The algorithm is run 10 times and the average classification accuracy values are presented. Fig. 5.9 compares the performance of our algorithm with that of a classical algorithm run on data aggregated from different distributed sources.

![Seeds dataset skewed: CAc Comparison](image)

**Fig. 5.9: Classification accuracy comparison in skewed dataset**

Fig. 5.10 shows the error incurred in classification by our algorithm and compares it with the error incurred by a standard centralized algorithm.
Next we apply our algorithm on wine dataset which is skewed. 70% instances are allocated for training and rest 30% for testing. There were 120 training instances and 58 testing instances. The training instances were distributed to 3 different data sites. The first data site had majority of instances (70%) belonging to class 1 and the rest of the classes had minority representations. Similarly data site 2 and 3 had class 2 and 3 as the majority classes. 2% change in rules was allowed in negotiation process and the acceptable incoming error was set at 0%. Fig. 5.11 shows the comparison of classification accuracies between our approach and the centralized technique for different purity values at leaf level.
We compare the misclassifications in the models built by our approach and the centralized approach. This is expressed in Fig. 5.12.

Fig. 5.12: Misclassification comparison for skewed wine dataset
We apply our algorithm on a skewed Pima Indian diabetes dataset. 70% instances are utilized for training and rest 30% for testing purposes. There were 426 training instances and 170 testing instances. The training instances were distributed to 3 different data sites. The first data site had majority of instances (70%) belonging to class 1 and the rest of the classes had minority representations. Data site 2 had class 2 as the majority class and class 1 acted as the minority class. In the data site 3 class 1 and class 2 were equally distributed. 2% change in rules was allowed in negotiation process and the acceptable incoming error was set at 2%. Fig. 5.13 shows the comparison of classification accuracies between our approach and the centralized technique for different purity values at leaf level.

![Pima Indian Dataset Skewed: CAC Comparison](image)

**Fig. 5.13: Classification accuracies in skewed Pima Indian dataset**

Misclassification by both our model and the centralized approach is compared in the Fig. 5.14.
Lastly we apply our algorithm to the banknote authentication dataset. There were 854 training instances and 328 testing instances. The training instances were distributed to 3 different data sites. The first data site had majority of instances (70%) belonging to class 1 and the rest of the classes had minority representations. Data site 2 had class 2 as the majority class and class 1 acted as the minority class. In the data site 3 class 1 and class 2 were equally distributed. 5% change in rules was allowed in negotiation process and the acceptable incoming error was set at 3%. Fig. 5.15 shows the comparison of classification accuracies between our approach and the centralized technique for different purity values at leaf level.
Fig. 5. 15: Classification accuracies in banknote authentication skewed dataset

The comparison of misclassifications is presented in Fig. 5.16.

Fig. 5. 16: Misclassification comparison for banknote authentication skewed dataset
It can be observed that the skewed datasets contribute to more error, but this behavior is expected. Skewed datasets create models that tend to ignore minority instances and provide rules that can classify only the majority classes. This behavior leads to increase in misclassifications. It can also be observed from these graphs that the classification error incurred by our algorithm learning the skewed dataset using is very similar to the one where all data is transported to a central data repository and a classical induction algorithm is used. This sort of behavior shows that creation of class sensitive rule quality measures have led to discovery of instances that have very less representation.

5.4.3 Effect of negotiation

It can be established from the previous section that our approach of combining distributed decision tree induction mechanism with a game theoretic process of negotiation has led to discovery of rules that have better classification accuracy. In this section we see the effect of applying negotiation algorithm and see its impact in improving the classification accuracy.

To see the effect of negotiation we run the classification algorithm before and after negotiation on a seeds dataset with 147 training set instances and 63 test set instances. All the training and test datasets are randomly selected. We run the algorithm 10 times and each time we plot the difference in classification accuracy for different percentage purity at leaf level. Fig. 5.17 shows the difference in classification accuracies before and after applying the negotiation mechanism. It can be observed that in almost all the cases there is a boost to the classification accuracy of the algorithm when negotiation mechanism is applied to the master rule set.
It is very evident from the results that negotiation between multiple agents dealing in the process of creation of decision tree induction leads to rules/patterns that can classify instances better.

5.4.4 Scalability analysis

A scalability analysis was performed using the Page blocks dataset from UCI machine learning repository [29]. It has 5 classes with 10 different real valued attributes. There are totally 5376 instances in this dataset. 70% of data was used for training and 30% was used for testing purposes. We ran our algorithm with a sample training set of 900, 1800, 2700 and 3900 instances and noted the execution time for our negotiation framework. All the samples were designed to preserve the class distribution of the original page blocks dataset.

The results are as follows:
<table>
<thead>
<tr>
<th>Number of Instances</th>
<th>Classification accuracy</th>
<th>Execution Time (in Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>900</td>
<td>98.45%</td>
<td>7</td>
</tr>
<tr>
<td>1800</td>
<td>97.6%</td>
<td>19</td>
</tr>
<tr>
<td>2700</td>
<td>98.2%</td>
<td>129</td>
</tr>
<tr>
<td>3900</td>
<td>96.8%</td>
<td>276</td>
</tr>
</tbody>
</table>

Table 5.2: Scalability Analysis

The behavior of our algorithm over time can be understood using the plot below:

Fig. 5.18: Scalability Analysis

It can be observed that there is no significant difference in classification accuracies of our algorithm with the increasing number of instances. Fig 5.18 seems to suggest that the algorithm may have a quadratic time complexity.
Chapter 6

6. Conclusion

Game theory research has increasingly found applications in the field of computer science, specifically in decision making systems. Game theory has been used for vehicle routing, network traffic routing and a host of other things. Negotiation has proved to be a tool for conflict resolution in multi-agent environments. Decision trees have been one of the most widely-used classifier systems due to their simplicity and ease of understanding. As data keeps growing more focus is placed on creating distributed algorithms for decision making. Most of the distributed decision tree induction algorithms lack intermediate interpretable models that can be used for checking the validity of rules they generate. The rules generated from distributed data by the usage of these algorithms are far less accurate than those generated from a model created out of a data source which comprises of an aggregate of all the distributed data, i.e., a model originating by centralizing all the distributed data. Many decision tree induction algorithms perform badly when the datasets are skewed. This is because they fail to learn patterns that can discover instances belonging to minority classes.

We use multi-agent systems [MAS] for distributed decision tree induction, where agents work in parallel to create individual decision trees. These trees are converted to rules and rules are aggregated in a central negotiator component. All of the participating agents bid on these rules and try to modify the rules to suit their own objectives. To resolve any conflicts between agents and build a consensus, we create an auction based bidding protocol where each of these agents compete with each other to reach a stable equilibrium. Every data site builds its own interpretable
intermediate model that can be verified to see if the rules generated are appropriate before the whole tree is built. The tree induction process also uses a class sensitive rule quality metric that helps to build a model where instances belonging to classes having smaller representation can be discovered. This metric contributes to decrease in number of misclassifications.

The results show that the process of negotiation among multiple data sites for building a common decision tree acceptable to all holds promise. It is observed that the classification accuracies of decision trees built using our approach have accuracies similar to the ones where distributed data is centralized and a classical decision tree is created. We test our algorithm on datasets that are skewed; it is able to discover rules that can classify even the minority instances thereby increasing the classification accuracy. Results for skewed datasets are also very promising.

The game theoretic framework of negotiation developed here is not Pareto efficient and also the negotiation process cannot continue without the central negotiator component. The agents involved in negotiation continue to bid on the rules and try to make the rules better until no more modifications are possible that can improve the rules. This process may take some time for all the agents to converge. Some rule pruning mechanisms can be utilized to shorten the rule set, this could lead to less time utilization. If Agents can coordinate amongst themselves and reach a consensus, this will remove the dependence on the centralized negotiator for tree building and make the framework fault tolerant. The work done as a part of this thesis is for horizontally partitioned data; similar mechanism can be developed for vertically partitioned data where the attributes are not common between different data sites. Similar methodology can be adapted to be utilized for streaming data. The concept of applying negotiation for rule improvement can be used for association rule mining and other classification methods.
Despite some of the shortcomings pointed out, the work done has significant advantages towards developing better models in a distributed data environment.
References


