UNIVERSITY OF THE WITWATERSRAND

WITS
UNIVERSITY

School of Computer Science and Applied Mathematics
Faculty of Science
MASTERS RESEARCH REPORT

An Empirical Comparison of Decision Trees and Decision Graphs on Supervised Learning Problems

Lavesh Vijay Patil

Supervisor(s): Prof Turgay Celik

October 2019, Johannesburg

Declaration

University of the Witwatersrand, Johannesburg
School of Computer Science and Applied Mathematics

SENATE PLAGIARISM POLICY

I, Lavesh Vijay Patil, (Student number: 1036697) am a student registered for COMS7009 - MSc(Computer Science) Research Report FT in the year 2018.

I hereby declare the following:

- I am aware that plagiarism (the use of someone else's work without their permission and/or without acknowledging the original source) is wrong.
- I confirm that ALL the work submitted for assessment for the above course is my own unaided work except where I have explicitly indicated otherwise.
- I have followed the required conventions in referencing the thoughts and ideas of others.
- I understand that the University of the Witwatersrand may take disciplinary action against me if there is a belief that this is not my own unaided work or that I have failed to acknowledge the source of the ideas or words in my writing.

Signature:

Signed on 29^{th} day of October, 2019 in Johannesburg.

Abstract

Recently interpretable machine learning algorithms have gained interest due to the transparency of their models. Some machine learning domains like healthcare and medical diagnosis expect the rationality and interpretability of decision models used for predictions. These domains need clear understanding of the decision rules, as it has a direct impact on human life. Currently, there are a few established algorithms like decision trees and decision graphs which support such explainable decision models. This study explores one of the efficient, but less known decision graph algorithms - Attribute-based Decision Graph (AbDG). In-built flexibility of this algorithm has led to its two variants, based on the edge creation approach, called as p-Partite and c-Partite. It also supports multiple interval methods to split an attribute. Recommended interval methods in the original algorithm are MDLPC, ESAD and EDADB. This study compares the performance of all such variants with commonly used decision tree CART (Classification And Regression Tree). In addition, it also investigates a few enhancements to AbDG such as optimization of finding interval range for values which were not seen during the testing phase and using additional K-Bins interval method. These empirical analysis results reveal that, decision tree models are more concise in certain scenarios, but are less accurate when compared to AbDG's highest performing variants. Also, it shows that, there are variants like MDLPC which produce more concise but less accurate models when compared to decision tree. Thus, AbDG and its variants can be further explored as an alternative to decision tree, when more accurate and interpretable models are required.

Acknowledgments

I am grateful to Prof. Turgay Celik for offering me this topic and for all the advice and guidance given.

I would also like to thank my family who always stood by me.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Attribute-based Decision Graph

Machine learning helps to solve various problems, especially with current trends, it has spread its wings wider. It has been helping many application domains to unpack and understand their problem and also solve these to a great extent. These problems are broadly categorized into four types such as supervised, unsupervised, semi-supervised and reinforcement. In supervised learning an algorithm devises a generalized model by analyzing provided input (typically vector data) and its corresponding output, which can be used for predictions. Supervised machine learning problems are also further divided into broader types based on the type of an output, i.e., if an output is a categorical attribute then it is called as classification problem and if an output is a real value then it is called as regression problem. Attribute-based Decision Graph [5; 6] is one of such machine learning algorithms, which solves specifically supervised classification problems.

Generally, most of the machine learning tools look at each attribute of the dataset as one full criterion in a bigger picture, which contains large sets of decision rules. However, some of the datasets have multiple characteristics hidden in each attribute space. Attribute-based Decision Graph is an attempt to unpack such hidden aspects of each attribute by splitting its full space in set of intervals. These intervals are then used more elegantly to create a decision graph with intervals as vertices and links between vertices belonging to different attributes as edges. The algorithm predicts the class-labels for testing data samples, by creating a decision rule which combines entropy based weights of each related vertex and edge from the trained model. This approach also helps to avoid overfitting considerably as the learned model maintains all possibilities based on entropy and allows flexible decision rules for predictions. The most important advantage of this algorithm is there are multiple ways to achieve this whole process depending upon the definition of the problem. Such flexibility opens new ways to handle various types of datasets using this attribute-intervals based method and then to re-look at many other machine learning techniques to unbundle more value from those techniques and corresponding datasets. Figure 1.1 illustrates the high level process of creating vertex, edges and assigning weights [6].

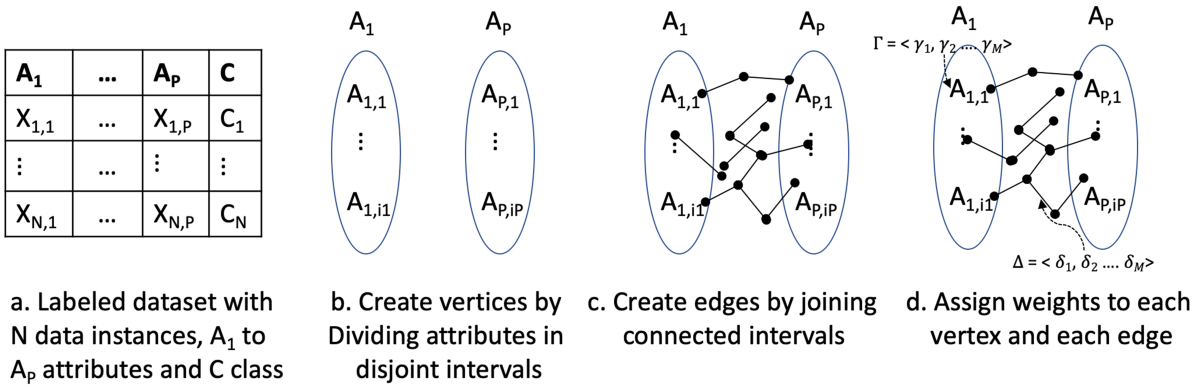

Figure 1.1: AbDG vertex, edge creation and assigning weights [6]

Legend: *a. let's consider labeled dataset with N instances, A_1 to A_P attributes and C class; b. create vertices by dividing each attribute to disjoint intervals $A_{1,i1}$ to $A_{P,iP}$; c.create edges by joining intervals corresponding to attribute-values of same data instance; d. based on the distribution attach weights to vertices and edges.*

1.2 Motivation

As the number of machine learning models are increasing daily, there is still a huge appreciation for interpretable and explainable tools like variants of decision trees. These have enormous advantages such as:

- generates transparent model showing decision rules explicitly;
- which in turn assists in model selection process;
- always provides decisions;
- incremental in nature;
- data split property shows clearer separation hidden in the data;
- allows better data analysis;
- efficient classification.

This list can keep growing, especially in the context of some applications for which decision trees provide huge advantages. However, at the same time, there are certain challenges that decision trees face such as:

- overfitting, which leads to less predictability for unseen samples;
- though, pruning helps to reduce the size of decision trees, very often little complex data can make decision trees quite complex to visualize;
- it is a weak classifier;
- it may require data pre-processing;

- due to greedy characteristics, it can lead to over-sensitivity to the training dataset, to irrelevant attributes and to noise.

Some of these disadvantages are trivial and can be handled by advanced methods. But, often it gets challenging to overcome some of those issues depending upon the applications and their available attributes and data.

Hence, it becomes important to explore decision trees to find alternative solutions which can enable its advantage and at the same time help to overcome its inherent disadvantages. For example, an algorithm which is transparent like decision tree, but, has its own built-in way to grow or reduce the size of decision tree as per the requirement; or an algorithm which easily produces interpretable graph structure like decision tree, but, creates a different meaning to its nodes and edges based on its attribute space which is converted into several intervals.

1.3 Objective

The primary objective of this study is to do empirical analysis of decision trees and Attribute-based Decision Graph (AbDG) algorithms for binary and multi-class supervised classification problems for balanced and imbalanced datasets obtained from the UCI repository. For decision tree, optimized CART based implementation provided by sk-learn library is employed. AbDG algorithm provides multiple variants based on two methods - a). number of edge based variants known as p-Partite and c-Partite; and b). interval methods based variants: MDLPC, EDADB, ESAD. In addition to these three basic interval methods, one more newer method known as K-Bins, implemented in sk-learn as KBinsDiscretizer is employed to showcase various possibilities that algorithms can take. This study involves in-depth analysis of multiple aspects between decision tree and AbDG algorithm including its eight variants, along with their advantages and disadvantages based on the results obtained by performance evaluation methods such as F1-score and confusion matrix. Also, based on these observations, it identifies possible improvements (if any) to the Attribute-based Decision Graph algorithm.

1.4 Document Structure

This document is divided into the following chapters. Chapter 2 provides background information on the research topic and how the idea of Attribute-based Decision Graph was evolved along with its implementation details. The chapter also discusses related work carried out in the industry during last few years. Chapter 3 specifies the research methodology used for this study and research hypothesis. Chapter 4 discusses the results obtained using various experiments conducted on multiple datasets. It also explains the observations based on results. Finally, Chapter 5 briefly concludes this study.

Chapter 2
Background and Related Work

2.1 Background

Recently there has been a new wave of interest in the field of machine learning and it is about using graphs as various mechanisms to solve machine learning problems. For example, graph neural networks [38], graph clustering [35], Deep Learning on Graphs [40], graph powered machine learning [24], etc. But, it has its roots in graph theory, which was introduced in the 17^{th} century by Leonhard Euler, who wrote a paper about solving Seven Bridges of Knigsberg problem and published it in 1736; this is considered the first paper on the history of graph theory [7]. However, it took 200 years before the first book *Theorie der endlichen und unendlichen Graphen* on graph theory was written by Dnes Knig in 1936. Since then the graph theory has been one of the favorite tool for mathematicians and other domain experts like chemists to represent their data and certainly it has proven its usability over such a long period. The reason for wider spread of graph theory is in its way of representing the most complex problems in a simplified and generic manner using graphs. Basically, graph theory involves the study of various graph structures established using various objects or their representations called nodes or vertices connected together over some relationship called links or often referred as edges.

Graph theory is used for a large number of machine learning problems. These problems can be broadly classified in two types: first is, where the data itself is organized in graph structures and second is, where the data is represented by a vector; but, it has some characteristics which allow it to utilize graph structure for its model or it can be represented in a graph structure. One of the most commonly seen example for the first type is the social networks where the data instances inherit the graph structure based on the relationships available in the actual network of data instance itself. Whereas, for the second type of problem any other data can be seen as a candidate as long as it can be represented in a vector form and whose instances are not related to each other, for example - individual income data, investment statements, phishing dataset for websites, etc. Some of the second type of problems do not have any output information (i.e., Unsupervised learning) and do not help any model to build direct understanding using available instances. In such scenarios, the knowledge is built using hidden aspects of data to form clusters and then use it to predict clusters for unseen data instances. A similar approach is taken for Semi-Supervised learning, where a few samples contain the output information, which further gets utilized to create partial knowledge about seen data instances and then, corresponding rules

or clusters are formed to allow predictions for the unseen data instances. The third category in this second type of problem is Supervised learning. For such type there are a few techniques in which graph structures are used, like decision-trees, neural networks, etc. The vertex of these graph structures either represent data attributes or a common characteristics between multiple data attributes.

However, rarely do any of these approaches delve into detail to analyze the attribute space and use it optimally, for a graph structure; for example, artificial Neural Networks uses graph but the graph nodes are not actual data instances except for input nodes where each attribute represents one graph node and all other nodes represent the propagation of calculated data or error instances. Similarly, k-Nearest Neighbor uses each full data instance as a node on the graph. A similar approach is taken by Support Vector Machine algorithm. However, decision trees have the closest relationship with the Attribute-based Decision Graph (AbDG) as they both are based on a similar foundation, i.e., decision graphs. Hence, this study is to compare AbDG with Decision Tree.

2.2 Decision trees

A typical way of defining a decision tree is - it is a directed acyclic connected graph; which starts with exactly one 'root node', from where all other 'subsequent nodes' are linked directly or indirectly using an 'edge' in an hierarchical manner and ends at 'leaf node' that has no further children. The components of a tree can be defined as below and shown in Figure 2.1 [31]:

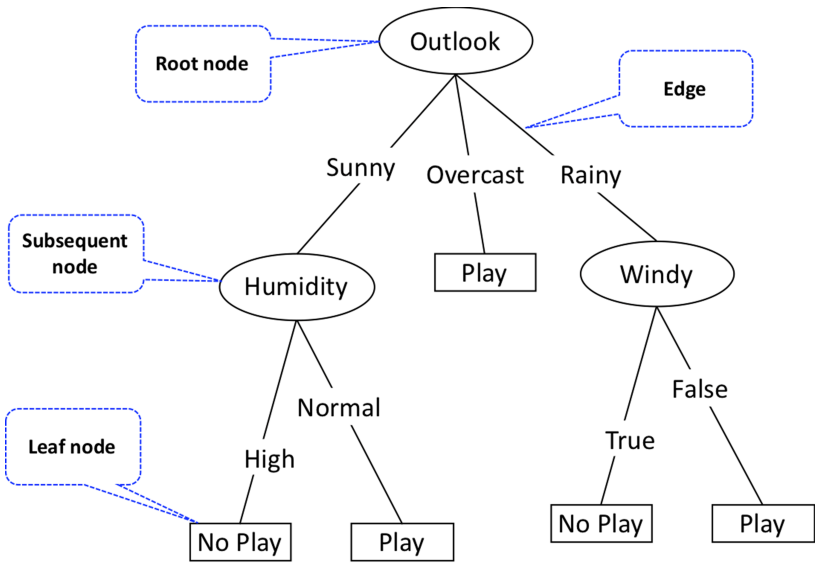

Figure 2.1: A simple decision tree [31]

- root node: where tree begins and which has zero or more children nodes, but no parent

node

- subsequent node: is any node of the tree which is not the root node and connected to at least one parent through the edge
- edge: is a link between two nodes having one parent node at the top and one child node at the bottom, generally seen as a method of creating relationship between the parent and child
- leaf node: is the last node in any given path of the tree, has no children and represents the resultant classification of the tree path, where each tree path represents the hierarchically linked set of decision rules based on the attribute values.

2.2.1 Inference

Based on available instances and their corresponding class labels, the decision tree creates rules in the form of tree paths, where groups of instances derive one path and every data instance follows only one path leading to only one class label. At any given node, the best available attribute becomes the root of the full tree or sub-tree, which can be derived using various methods, such as information gain [16] and gini index [9]. Generally, each node represents a binary branch, where the best available attribute's space is divided into two intervals where one side represents positive class label and the other side represents negative class label. After formulating the entire tree using all attributes recursively to divide all seen data instances, it can be used for validation.

2.2.2 Overfitting

In some cases, this can lead to an exact map of seen data instances to the class labels and not enough generalized rules are developed, which leads to lower accuracy in predicting class labels for unseen data instances. This is known as overfitting. Decision trees generally use two very common approaches to avoid the overfitting: a). use a stopping criteria to not allow the decision tree to grow for overfitting; and b). pruning by using validation set where algorithm allows to minimize the size of a tree by removing its parts which do not provide much of the information and do not really contribute much to identify the class of a testing sample. This reduces the classifier's size and makes it more simpler, which assists in reducing the overfitting and enhancing performance of the classifier.

2.2.3 Variants

C4.5, ID3 and CART are the most widely used Decision Tree algorithms. ID3 (Iterative Dichotomiser 3) [30] uses a top down and greedy search approach to test each attribute at each node of the tree to find the best possible split for that node. C4.5 [29] is considered as an

enhanced version of the ID3 algorithm, especially for the way it finds the splitting points. It uses gain ratio, which relies on the respective ratio of the information gain to allow the creation of more generalized models. In addition the algorithm also allows the handling of continuous attributes by transforming them to nominal attributes. Another enhancement, which allowed wide spread of C4.5 is that, it handles the missing values of the attributes by ignoring them and only uses the distribution based on the available values of that dataset attribute. CART (Classification And Regression Tree) [9] handles nominal and interval attributes along with nominal, interval and ordinal targets, i.e., it uses data in its raw state and does not require the conversion to other types. To support multiclass problems, it uses the towing splitting criteria. For handling the pruning, CART searches for an attribute which has minimal contribution or no significance, as such, to the classifier based on the cost computation. Also, like C4.5, it automatically handles the imbalanced data and missing values.

2.2.4 Decision Tree Algorithm

Algorithm 1: Decision Tree: GrowTree [34]

Function `GrowTree(`X, y, $attributenames$, $splitmethod$, $stopmethod$`):`
- create a new tree T with a single root node
- **if** *stopmethod(X)* $!= 0$ **then**
  - Mark T as a leaf with the most common value in X as a class label.
- **else**
  - $\forall\ a_i \in A$, find a that obtain the best result from $splitmethod(a_i, X, y)$.
  - `/* generally` $splitmethod$ `uses minimum message lengths provided by Gini index, Information Gain, etc as splitting criteria */`
  - label t with a.
  - **foreach** *outcome* v_i *of* a **do**
    - set $subtree_i$ = `GrowTree(`X, y, $attributenames$, $splitmethod$, $stopmethod$`)`
    - Connect the root node of t_T to $subtree(_i)$
    - with an edge that is labeled as v_i.
- **return** *PruneTree(*X, y, T*)*

Due to overfitting issues, decision tree learning has two phases: growing the tree and then pruning it. Generally a top-down induction method [31] is used by most of the decision trees. The following listing shows the steps of this method, based on the algorithm as laid out by Lior Rokach and Oded Maimon [34] in their book Data Mining with Decision Trees: Theory and Applications. In its Section 6.2, the book also shows several high level pruning approaches which are generally used by most of the decision tree variants.

Consider, X is a dataset sampled from a training dataset and y is the target for those training

dataset samples, $attributenames$ is the set of all input attributes, $splitmethod$ indicates the approach for evaluating the split and $stopmethod$ evaluates the criteria to be used to stop the tree growing. Algorithm 2.2.4 shows a generic algorithm for decision tree's growing process [34].

Generally, the tree growing and pruning tasks continue till one of the following criteria is met:

- if all target values of the training dataset, i.e., y contains same class-label for all training instances, i.e., $unique\_count(y) = 1$;
- if there is a predefined criterion for maximum tree depth and if that is reached;
- if the number of training samples in the leaf node is less than the predefined minimum number of training samples for parent nodes;
- if a node needs to be split then, the number of training samples in one or more child nodes is less than the predefined minimum number of training samples for child nodes;
- if the best splitting criterion retrieved by the method after evaluating all attributes is not greater than a predefined threshold value.

Now, consider T as the decision tree which requires to be pruned. Then, Below is a generic algorithm for its pruning process [34]:

Algorithm 2: Decision Tree: PruneTree [34]

Function `PruneTree(`X, y, T`):`

`/* Now, consider` T `as the tree which requires to be pruned. */`

while $t = \Theta$ **do**
select a node t in T such that pruning it maximally improve some evaluation criteria Θ based on a stopping criteria

if t != Θ **then**
$T = pruned(T, t)$

return T

Most of the decision tree variants uses generic algorithms as shown above to derive the decision tree from the training dataset samples. scikit-learn uses an optimized CART algorithm; but, it does not support categorical variables for now. Hence, it may require data pre-processing like encoding categorical attributes to numeric.

2.2.5 Splitting

Most of the decision tree variants mainly differ based on the splitting methods implemented in that algorithm. Various impurity measures are used for this as mentioned below:

- Entropy: it measures the impurity of a node. Let's say $p(x)$ is the fraction of examples for a given class label of the total number of classes c then, entropy is calculated as shown in the following equation:

$$Entropy = -\sum_{i=1}^{c} p(x_i) \log p(x_i). \tag{2.1}$$

- Gini Index: is also a measure of impurity, i.e., it is a criterion to minimize the probability of mis-classification and it is calculated as one minus sum of each fraction $p(x)$ squared.

$$GiniIndex = 1 - \sum_{i=1}^{c} (p(x_i))^2. \tag{2.2}$$

- Classification Error: is also sometimes called as mis-classification error and it is computed as one minus maximum value of the fraction $p(x)$:

$$Error = 1 - max(p(x)). \tag{2.3}$$

- Information Gain: it represents the difference between pre and post effect, if the split occurs using the node. It is calculated as entropy (or similar mis-classification function as shown above) of the parent node before the split happens and of the left and right child node after the split happens. Let's say x the parent node and x_l as left child node and x_r as right child node, then information gain for parent node will be calculated as:

$$InfoGain = Entropy(x) - Entropy(x_l) - Entropy(x_r). \tag{2.4}$$

- Gain Ratio: is a normalized form of the Information Gain, which reduces its bias on attributes with high distinct values and it is computed as follows, where Intrinsic Value is entropy of an attribute based on its all possible values.

$$GainRatio = \frac{InformationGain}{IntrinsicValue}. \tag{2.5}$$

It is very difficult to identify which measure is the best for splitting the tree at each node, as every measure has its own pros and cons based on its properties and also based on the attributes for which it is used.

2.3 Decision Graphs

Decision Trees are well known and one of the most used supervised machine learning techniques. However, with reference to complex supervised classification problems, decision trees have two major drawbacks: replication and fragmentation problems [27]. Basically, decision trees are proven to be inefficient representations for dis-junctions, which are shown with duplication of sub-trees, leading to the larger decision trees. One solution for a replication problem is to allow decision nodes to contain attributes that are functions of one or more attributes. This solution is quite complex and needs more exploration especially for noisy data and for multiclass problems. A fragmentation problem occurs when data contains more than two values, leading to many fragments of small data sizes, which results in larger tree sizes. Hence, to resolve these issues, Oliver and Wallace [27] introduced the machine-learning technique of decision graphs, which they adopted as a generalization of the decision trees. To solve the fragmentation problem, decision graph allows leaves with same class label to merge, leaving one leaf per class. To

solve the replication problem decision graph allows to merge identical sub-trees. In addition to these two advantages, like decision trees, the primary advantage of decision graphs is that – the resulting decision graph provides both a prediction method and also an explanation for the problem to increase acceptability of the model or to engage better understanding of the data. Fundamentally, decision trees and decision graphs are very similar in a few aspects like the way they categorize the objects (i.e., partition into disjoint sets of data) and the way they represent their decision functions and decision rules.

However, the way decision function categorizations occur in decision graphs, it can be different to that of decision trees. Another major aspect where both differ is that, decision graphs can also contain joins. Its two nodes may have a common child, which means that two subsets of a graph may have some common properties and using those properties or some criteria of those properties, two subsets can be merged or considered as one subset. Thus, it can be expected that decision graph models will always contain fewer leaves than that of decision trees. There are also other approaches to creating a graph, for example, a tree is grown fully based on the data and then, sub-trees are searched, so that related sub-trees are merged to form a decision graph. But, due to the limited success of this option Oliver and Wallace adopted a variant of Minimum Message Length Principle (MMLP) to build a decision graph. Minimum Message Length is a criterion for comparing inductively derived theories and the explanations that these theories provide, using which shortest theory can be selected; where, output of a program that performs inductive inference can be considered as a ”theory” of the input data. Also, the method avoids arbitrarily predetermine the gross structure of a decision graph before the graph is grown, instead the gross structure is determined by the data. To build and grow this graph, a root node is considered as a leaf node, which is then grown until a perfect graph is identified or if it cannot be grown any further. Using this node to iteratively perform graph growth process, i.e., for each leaf node find a candidate attribute to do a split and then find the message length for the split as well as for the join on each pair of leaves which would get effectively created using that split. Then based on the shortest message length between the two options, evaluate the next step, to either do a split or to do a join on that attribute. This has been generically explained by Joost de Nijs [12].

Later, decision graphs were reviewed quite extensively by many researchers and a number of studies were conducted to explore various variants, some of those well-known techniques can be found in a comprehensive survey of decision trees in an article *”simplifying decision trees: a survey”* by Leonard A Breslow and David W Aha [10].

2.4 Attribute-based Decision Graph

As mentioned above, the Attribute-based Decision Graph (AbDG) [6] introduced a totally a new way of using the Decision Graphs. To create vertices of the decision graph, it uses intervals of value-range if attribute is numerical or uses distinct values if attribute is categorical. Edges of the decision graph are created as links between the two vertices belonging to two different attributes. This results in various patterns for each class label, but at the granularity of the attribute values or range of values. Other decision graph related techniques do not consider the fact that each attribute's multiple values or range of values contain an information beyond just making binary partitions, especially if those values or range of values belong to the different class labels.

2.4.1 Intervals

The most vital part of this algorithm is to create intervals which can represent the set of appropriate vertices. Hence, the algorithm proposes flexible ways to create these intervals and then integrate such various mechanisms easily into the main algorithm. The algorithm further divides the strategy for creating intervals, based on the type of attribute. For example, if an attribute is categorical then, each distinct value of that attribute can represent a separate interval. However, this method is not possible in case the attribute is numerical. Hence, the suggested approach is – to use a discretization method to facilitate the intervals of the numeric attributes. There are a couple of recommendations specified by the algorithm to make sure the algorithm behaves efficiently irrespective of the technique selected; for example, the interval creation process should create a minimal version to achieve model simplicity, to enhance the classification accuracy and to have low computational costs. However, the algorithm does not limit to the way intervals are created. It can be seen very clearly that, the algorithm will benefit if the study gets extended for vast ranges of discretization techniques. For further view on discretization techniques a detailed survey *"Discretization: An Enabling Technique"* [18] conducted by Huan Liu, Farhad Hussain, Chew Lim Tan, Manoranjan Dash can be refereed.

2.4.2 Cut Points

Most of the methods of discretization rely on cut point candidates to identify intervals with the highest classification accuracy. [14] The cut point candidate is defined as – given the ordered list of values of an attribute, if the values consecutive to each other are not the same and do not belong to the same class then the middle point of those two values, yield highest classification accuracy compared to any other potential points belonging to the same range. The algorithm also recommends two methods based on this principle known as MDLPC (Minimum Description Length Principal Cut Criterion) [14] and EDADB (Entropy-based Discretization According to

Distribution of Boundary Points) [3].

2.4.3 MDLPC

It first identifies the cut point candidates using the method discussed above. And then, uses the entropy based method to filter those candidates to only select best cut points which will yield maximum accuracy. For example, an attribute A_a from the given training dataset X with the number of samples as N and the splitting point T identified using cut point candidate method, which splits the training dataset into two sets X_1 and X_2 of sizes N_1 and N_2 respectively then the gain for that splitting point is calculated as shown in the following equation [6].

$$Gain(A, T, X) = Entropy(X) - \frac{N_1}{N} \times Entropy(X_1) - \frac{N_2}{N} \times Entropy(X_2). \tag{2.6}$$

The splitting criteria defined for this method is as shown in the following equation [6].

$$Gain(A, T, X) > (\frac{\log(N-1)}{N} + \frac{\Delta(A, T, X)}{N}), \tag{2.7}$$

where,

$$\Delta(A, T, X) = \log(3^M - 2) - [M \times Entropy(X) - M_1 \times Entropy(X_1) - M_2 \times Entropy(X_2)], \tag{2.8}$$

where, M is the total number of classes in the given dataset, M_1 is the total number of classes in X_1 and M_2 is the total number of classes in X_2.

2.4.4 EDADB

Like MDLPC, EDADB also uses entropy, in addition it considers the distribution of cut point candidates over the entire value range of the attribute. It first identifies the number of splitting points as shown in below equation [6].

$$m = \max(2, M \times \log(L_a)), \tag{2.9}$$

where, L_a is the number of distinct values of the attribute A_a. The method focuses on spreading the cut points across the full range of the attribute, instead of only selecting the attributes with highest information gain. To achieve this, the method divides attribute value range in D equal parts by using $D = \max(1, \log(L_a))$. Let's assume the total number of cut point candidates are U_a, then each i^{th} part in D will contain cut point candidate u_i cut points. The count of all those u_i should sum to U. The number of cut points in i^{th} part can be derived as $q_i = [(u_i/U_a) \times m]$, where $[\cdot]$ rounds off the result to the closest integer. Once all cut points of each part are identified then they are sorted according to the information gain. For this information gain calculation, the effect of each cut point is considered over the entire space of the attribute instead of calculating local gain.

2.4.5 ESAD

The third technique recommended by the algorithm is called ESAD (Equi Sized Attribute Division), which does not need attribute sorting, rather all it needs is to find the minimum and maximum values of the attribute, resulting in a computationally effective approach. For this, the entire range of an attribute is divided into equally sized intervals based on the externally supplied parameter, for example let's say there is an attribute A_a belonging to a dataset D having p attributes. Then based on the training dataset samples minimum and maximum values of the attribute are identified as d_{min} and d_{max} respectively. Let's assume that, externally supplied value for number of expected intervals is n_a. Then, size of each expected interval is calculated as $s = |d_{max} - d_{min}|/n_a$. Let's assume boundaries of the k^{th} interval $I_{a,k}$ are indicated as $(d_k, d_{k+1}]$ then, d_k can be calculated as $d_{min} + k \times s$. Similarly, d_{k+1} can be calculated as $d_{min} + (k+1) \times s$.

2.4.6 Inference

The testing process can use these vertices and edges to match against the testing dataset samples and the corresponding class can be identified for those samples. However, in the real world there are very few chances that, exact matching patterns of graphs can be identified for those testing dataset samples. Hence, the algorithm introduces the 'weight based mechanism' for vertices, edges and also for the classifier function. These weights allow the algorithm to use probability based techniques to specify the patterns created during the training phase. This results in a more robust classifier which can use these weights to match the patterns for training dataset samples instead of finding exact matching patterns. Once, the intervals are identified by one of the methods mentioned above then, that set becomes the vertices of Attribute-based Decision Graph and corresponding edges are created to link those vertices. Then, weights pertaining to each class are assigned to each vertex and each edge. For this, two vectors are created of size M (count of distinct values of classes), one for vertex and other for edge. Where each component of this vector is a weight associated with the corresponding class j.

Let's assume k^{th} interval of a^{th} attribute A_a is $I_{a,k}$ and its corresponding vertex is $v_{a,k}$ and its weight vector is represented as $\Gamma_{a,k}$. The components of this vector are $\gamma_1, \gamma_2 \ldots \gamma_j \ldots \gamma_M$, where components representing weight for j^{th} class can also be represented as $\Gamma_{a,k}(j)$. Similarly, let's assume there is another vertex $v_{b,q}$ and the edge connecting to these two vectors can be represented as $e^{b,q}_{a,j}$ and its corresponding weight vector for j^{th} class is $\Delta^{b,q}_{a,j}(j)$, which can also be represented as $\delta(j)$.

Now, the weight component is calculated as [6]:

$$\gamma_j = \Gamma_{a,k}(j) = P(j|I_{a,k}) = \frac{P(I_{a,k}, j)}{P(I_{a,k})}, \tag{2.10}$$

i.e., it is the conditional probability of the given dataset sample belonging to the j^{th} class and whose attribute A_a's value belongs to the $I_{a,k}$ interval. And $P(j))$ is the marginal probability of

class j. This probability is calculated as [6]:

$$P(I_{a,k}, j) = P(j) \times P(I_{a,k}|j), \tag{2.11}$$

where,

$$P(j) = \frac{N_j}{N}, \tag{2.12}$$

where, N_j is total number of testing dataset samples belonging to class j and N is total number of testing dataset samples

$$P(I_{a,k}|j) = \frac{|(x_i|x_{i,a} \in I_{a,k} \wedge c_i = j)|}{|(x_i|c_i = j|)|}, \tag{2.13}$$

and $P(I_{a,k})$ is the normalizing term and calculated as:

$$P(I_{a,k}) = \sum_{j=1}^{M} P(I_{a,k}, j). \tag{2.14}$$

In summary, the weight for each element of the vector is calculated based on the conditional probability of getting class j given the data sample belonging to the interval A_{a_i}. This probability is computed by multiplying the probability of class j given the training dataset by the conditional probability of the class j instances belonging to the interval A_{a_i}. Then, all such probabilities are normalized by dividing those with the normalizing element, which is computed as the sum of all probabilities.

A similar approach is taken for edges as well. However, for edge weights the conditional probabilities are calculated using probabilities of both vertices which are linked by the edge. It can be shown as follow [6].

$$\delta(j) = \Delta_{a,j}^{b,q}(j) = P(j|I_{a,k}, I_{b,q}) = \frac{P(I_{a,k}, I_{b,q}, j)}{P(I_{a,k}, I_{b,q})}, \tag{2.15}$$

where,

$$P(I_{a,k}, I_{b,q}, j) = P(j) \times P(I_{a,k}, I_{b,q}|j), \tag{2.16}$$

$$P(I_{a,k}, I_{b,q}|j) = \frac{|(x_i|x_{i,a} \in I_{a,k} \wedge c_i = j \wedge x_{i,b} \in I_{b,q})|}{|(x_i|c_i = j|)|}, \tag{2.17}$$

i.e., the ratio of training dataset samples which not only belong to the j_{th} class but also has its values of attribute A_a are in the k_{th} interval and its values of attribute A_b are in the q_{th} interval. Then, like vertex, the normalizing term is calculated as shown below [6].

$$P(I_{a,k}, I_{b,q}) = \sum_{j=1}^{M} P(I_{a,k}, I_{b,q}, j). \tag{2.18}$$

Once a model is discovered, it is used for predicting the classes of testing dataset samples. To achieve this, corresponding intervals and their vertices and edges weights need to be re-

trieved for each attribute of each test sample. Weights for vertices are combined (represented as $PW(y)_j$) using product function and the class with highest normalized product value is considered as the class identified by the vertices. Similarly, corresponding edges' weights are combined (represented as $SW(y)_j$) to derive the class having the highest normalized sum value. Finally these two components are combined to derive the final class label using a classifier parameter. It allows to externally control the varying emphasis between the weights of vertices and edges.

$$PW(y)_j = \prod_{a=1}^{p} \Gamma_{a,k}(j), \tag{2.19}$$

where, $y_a \in I_{a,k}$,

$$SW(y)_j = \sum_{a=1}^{p-1} \Delta_{k,q}^{a,a+1}(j), \tag{2.20}$$

where, $y_a \in I_{a,k} \wedge y_{a+1} \in I_{a+1,q}$.

Above equation [6] is for p-Partite edges as it has maximum only one layer of edges associated with each attribute. But, for c-Partite it is not the same case. For c-Partite there are edges from each attribute interval to all other attribute intervals. Hence, the sum is calculated as shown in the following equation [6].

$$SW(y)_j = \sum_{a=1}^{p-1} \sum_{b=a+1}^{p} \Delta_{k,q}^{a,b}(j), \tag{2.21}$$

where, $y_a \in I_{a,k}$ and $y_b \in I_{b,q}$.

Then, both vectors of weights $PW(y)_j$ from (2.19) and $SW(y)_j$ from (2.20) or (2.21) needs to normalize as shown below.

$$P(y|j) = \frac{PW(y)_j}{\sum_{j=1}^{M} PW(y)_j}, \tag{2.22}$$

$$Q(y|j) = \frac{SW(y)_j}{\sum_{j=1}^{M} SW(y)_j}. \tag{2.23}$$

And finally, an empirical estimation for testing the dataset sample's class label (denoted as: $\varphi(y)$) needs to be calculated by combining those two probabilities $P(y|j)$ and $Q(y|j)$ using externally supplied parameter (denoted as η) of the classifier, as shown below [6].

$$\varphi(y) = argmax_{j=1..M}(\eta \times P(y|j) + (1-\eta) \times Q(y|j)). \tag{2.24}$$

2.4.7 Variants

If there are large number of vertices being created then, edge creation process can be taken a step further, by allowing some control mechanism over number of edges being created. This concept leads to two variants of the AbDG algorithm. The first variant contains all edges where each

vertex of an attribute is connected to each vertex of all other attributes, this is called c-Partite. Whereas the second variant is called p-Partite. p-Partite method limits the edge creation process by only allowing edges to be created between two sets of vertices belonging to two separate attributes which are consecutively ordered by some ordering mechanism. This ordering can be based on either a predefined order or a predefined ordering function. The algorithm also allows inheriting the edge control mechanism by using an information gain based filter for first type of variant, i.e., c-Partite, where filter allow dropping attributes which has the lowest information gain.

2.5 Empirical Analysis of Decision Trees

AbDG algorithm is relatively new in the machine learning field and has not been investigated much yet. Hence, this study is the first of its kind where AbDG is empirically compared with decision trees, other than its original proposition.

However, recently a few empirical analysis studies involving decision trees have been conducted, not many though. Some of which were done to compare decision trees with other **neural network based decision diagrams**. Mues et al., 2004 [23] performed a very interesting empirical study of two neural network rule (tree) extraction algorithms: Neurorule and Trepan, by comparing with the neural network with three other decision graph based algorithms: decision trees (C4.5), rules (C4.5rules) and diagrams (EODG - Entropy-based Oblivious Decision Graphs). They used three real-life credit-risk evaluation data sets: German credit, Bene1 and Bene2; which were discretized using Fayyad and Irani's MDLPC based discretization algorithm [14] with the default options. Based on the obtained performance results, they concluded that with Neurorule, Trepan and neural networks all these algorithms gave consistently, a very good classification performance when compared to C4.5, C4.5rules and EODG. Also, Neurorule and Trepan extracted quite concise models, when compared to others, except for EODG, whose decision models were most concise. However, the classification performance by EODG was considerably lower when compared to other algorithms.

Similarly, some of the comparison studies were conducted between various **supervised learning algorithms** such as Caruana et al., 2006 [11], who showcased empirical analysis of Decision Trees, SVMs, ANN, Logistic Regression, Naive Bayes, KNN and also some boosting techniques like Random Forests, Bagged Trees, Boosted Trees, Boosted Stumps. The study was done mainly on binary classification problems. They divided eight performance metrics into three groups: threshold metrics (accuracy (ACC), F-score (FSC) and lift (LFT)), ordering/rank metrics (area under the ROC curve (ROC), average precision (APR), and precision/recall break even point (BEP)) and probability metrics (squared error (RMS) and cross-entropy (MXE)). They proved that - calibrated boosted trees were the best learning algorithm with random forests very close on second position, followed by un-calibrated bagged trees, calibrated SVMs and uncalibrated

neural nets. The models that performed not so well were naive bayes, logistic regression, decision trees and boosted stumps. However, though decision tree performed poor its ensembles like boosted trees, random forests performed the best; which, suggests that little tweaks to manage decision trees can lead to best performing algorithms.

Similarly, Aruna et al., 2011 [4] also conducted an empirical analysis of multiple supervised classification techniques like Nave Bayes, Support vector machines (SVM Gaussian RBF kernel), Radial basis neural networks, Decision trees C4.5 (J48) and simple CART in aiming for to find the best classifier for detecting diseases, by using multiple UCI datasets for binary as well as multiclass classification problems. They used a ten fold cross validation method. In this study, a confusion matrix was employed as a major accuracy metric to analyze the classifier's performance based on accuracy, precision, recall and Mathews correlation coefficient (MCC). This study has shown that SVM-RBF kernel always exceeds in performances when compared to other classifiers not only for binary classification problems but also for multiclass classification problems.

Osofisan et al., 2014 [25] applied a similar study for decision trees and artificial neural network to mine an education database to predict student performance. A similar study was performed by Yiyan et al., 2017 [37], in which they used seven types of generally well-known and widely-used algorithms, namely, C4.5 decision tree, logistic regression, k-nearest neighbour, nave Bayes, support vector machine; which also includes ensemble techniques like AdaBoost and random forest over twelve top-clicked UCI datasets. This study concludes clearly about decision trees saying that, C4.5 decision tree algorithms performed well on binary classification and multi class classification problems when used for the biomedical datasets. The observation also mentions that, C4.5 is quite easy to understand and also interprets the underlying decision rules. The performance measure used in this study, is based on three aspects - classification accuracy, running speed and memory usage.

Also, some other studies were conducted to compare decision trees with **ensemble classifiers** like Kelarev et al., 2013 [17] who performed the empirical study between decision trees and ensemble classifiers for Monitoring of Diabetes Patients in Pervasive Healthcare. This study was focused on a few well known techniques like ADTree, C4.5 decision tree, NBTree, RandomTree, REPTree, SimpleCart along with their ensembling using techniques such as Adaboost, Bagging (bootstrap aggregating), Decorate, Dagging, Grading, HBGF, MultiBoost, Stacking and a combination of Boosting and Bagging. For performance assessment, they used a 10-trial 10-fold cross validation and the standard performance metrics like accuracy, precision, recall and ROC area. It is obvious that ensemble methods would perform much better than simple decision trees; but, the close look at comparison tables of decision trees' performance shows that RandomTree and SimpleCart performed the best, followed by C4.5 decision tree implementation.

2.6 Conclusion

Though there are a few empirical studies involving decision trees and decision graphs, most of the work in these two algorithms is about their optimization, by finding the shortest path, pruning, etc. and not much about their comparison with each other or any other algorithms. Hence, this study is quite important in this regard as it not only discusses Attribute-based Decision Graph in detail, but it also facilitates future work in the direction of empirical comparisons between decision tree and decision graph or other similar supervised classification algorithms.

Chapter 3
Research Methodology

3.1 Introduction

The previous chapter presented Decision Trees and Attribute-based Decision Graphs and their further details like algorithm, splitting criteria, etc. This chapter formally states the objectives of this research and details about the research methodology which are being followed to obtain the results and their observations. The next sections present the research hypotheses and motivates the choice of these hypotheses, along with the steps to be taken in order to accept or reject them, followed by a short summary of the chapter.

3.2 Research Hypothesis

As discussed in previous chapters, decision graphs have been used in machine learning for a while. However, recently introduced Attribute-based Decision Graphs have not been explored and their results have not been extended further yet. Hence, the purpose of this research is to examine the Attribute-based Decision Graph for various supervised classification problems categorized as binary-class and multiclass for balanced and imbalanced data problems. This study empirically compares in-depth performance of two techniques: decision tree (optimized CART from sklearn) and AbDG. Also, it briefly reveals other aspects about the explanability of the Attribute-based Decision Graph when compared to decision trees. And finally the analysis of results identify if there are any possible improvements to the original algorithm which would enhance the performance of AbDG for supervised classification learning.

This statement leads to the following hypotheses:

- *Hypothesis 1: Accuracy of Attribute-based Decision Graph's inference models is equally or better than decision trees.*
- *Hypothesis 2: Attribute-based Decision Graph enables concise inferred models which allow enhanced visibility of the decision rules.*
- *Hypothesis 3: There is a further scope of minor improvement for the originally specified Attribute-based Decision Graph algorithm.*
- *Hypothesis 4: Accuracy of ESAD variant based inference model is higher than other variants of the Attribute-based Decision Graph*

3.3 Hypothesis Evaluation

To measure *Hypothesis 1, 3 and 4*, *F-beta score* is used, which is supported by accuracy score, precision, recall and confusion matrix, for further analysis of results.

Confusion-matrix shows a matrix of actual versus predicted counts per class. Each column of this matrix represents the total number of testing samples in a predicted class and each row represents the total number of testing samples in an actual class (or vice versa). For example, let's assume $C_{i,j}$ denotes the number of instances which are represented as i instead of j, i.e., for binary-class problem, if i is the expected or actual class-label and j is the predicted class-label, then Table 3.1 can be used as a representation of the confusion-matrix. Thus, it clearly shows the number of true-negatives vs the number of false-positives and the number of false-negatives vs the number of true-positives.

Table 3.1: Confusion Matrix

Confusion Matrix for Binary-Class		
Confusion Matrix	**Predicted Class - Y**	**Predicted Class - N**
Actual Class - Y	100 (TP)	20 (FN)
Actual Class - N	15 (FP)	200 (TN)

Accuracy-score is calculated as a ratio between the sum of true-positives and true-negatives (i.e., matches between predicted and actual values) and the number of samples used for testing. For example, based on the above confusion matrix shown in Table 3.1, the accuracy of the model is calculated as shown below.

$$\begin{aligned} Accuracy &= \frac{correct\_predictions\_count}{total\_samples\_count} \\ &= \frac{TP+TN}{TP+TN+FP+FN} \end{aligned} \tag{3.1}$$

where, TP = True Positives, i.e., correctly predicted positive values, TN = True Negatives, i.e., correctly predicted negative values, FP = False Positives, i.e., incorrectly predicted negative values, FN = False Negatives, i.e., incorrectly predicted positive values.

Precision is calculated as a ratio between the number of true-positives and the number of total-predicted-positives (i.e., sum of true-positives and false-positives) i.e., TP/(TP+FP).

Recall is calculated as a ratio between the number of true-positives and the number of total-actual-positives (i.e., sum of true-positives and false-negatives) i.e., TP/(TP+FN).

F1 Score is calculated as a harmonic mean (also called as weighted average) of precision and recall. It is more useful when precision and recall both need to be used mainly to handle the imbalanced datasets where one class weighs higher than other and results in high accuracy even if the model is inaccurate. This study also includes the number of imbalanced datasets and

hence, F1 score is used as a primary accuracy measure.

$$F1score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.2}$$

Hypothesis 2, is accepted or rejected based on the most concise and representative decision tree generated by both algorithms, i.e., AbDG and Decision Tree. For this, the details of variants which give best results is used to compare the two algorithms. For example, let's say AbDG gives the highest performance for the Iris dataset, when the training ratio is 0.6 and vertices method used is ESAD and number intervals used is 4 with classification parameter 0.7. Similarly, let's say DT (decision tree) gives the highest performance for the Iris dataset at training ratio 0.5. Then, both these instances of the result array are selected and corresponding nodes and edges count is retrieved. For AbDG, training process generates a list of vertices and edges along with their weights as intermediate output of the fit process, to allow testing in the predict cycle. The nodes and edges counts are collected straight after the training of the final optimized run for each set of parameters. Similarly, sk-learn's decision tree classifier facilitates with two results a. Decision Tree, b. Node Count. A generic method to traverse this tree is used to identify leaf nodes count of each tree. Then, using total node count and leaf node count, edge count is determined, assuming that, for any non-leaf node there will be two edges and no edges for leaf nodes. This way, two sets of results are selected for each maximum performing output per dataset, to accept or reject hypothesis 2.

For *Hypothesis 3*, observations from the best performing variant of the Attribute-based Decision Graph algorithm is used to derive minor enhancements which can be made to the original algorithm and then its performance is measured against the previously selected best performing variant, to accept or reject the hypothesis. The method followed for this evaluation is the same as hypothesis 1, where the high performance run's results are selected for non-enhanced and enhanced versions of the algorithm.

For *Hypothesis 4*, similar to hypotheses 1 and 3, the best performing results are collected per interval-based variant method and compared with each other using F1-score.

3.4 Implementation

In order to empirically study the Attribute-based Decision Graph and Decision Tree, the Attribute-based Decision Graph must be implemented along with its three types of variants such as: types of edges based variants (**p-Partite and c-Partite**), interval method based variants (**MDLPC, EDADB, ESAD, KBINS**). Previous sections showcased the design of the algorithm and its variants. For implementation, those algorithms are leveraged to design and create fully functional variants of the Attribute-based Decision Graph algorithm.

3.4.1 Classifier

The implementation of variants of the Attribute-based Decision Graph algorithm is based on Python's pandas [20] and numpy [26] toolkits and sk-learn library [28].

For this experiment, most of the data is kept untouched and consumed as it is, except for the scenarios where categorical data is used for the decision tree. Sk-learn's decision tree classifier does not yet support categorical data, hence, it is encoded using pre-defined set of libraries like label encoder of sklearn [28].

Once the data is read from csv files, it needs to be split into a training dataset and a testing dataset, based on the training ratio parameter. Then the training dataset is used for creating models for each variant of the algorithm. This training involves a number of steps based on the type of variant. After training, the discovered model is used for predicting the classes of testing dataset samples, which were unexposed during the training phase.

The classifier implementation requires two types of parameters for which a grid search process is conducted.

- **classification parameter**: For AbDG, after fitting the model, vertices and edges are created along with their weights. Later, the predict process uses these weights to derive the class label. At this stage, classification parameter η is used as shown in the equation (2.24). Basically, it balances the weights between vertices and edges while deriving the class label. To grid search this parameter, the system uses a five-fold cross validation method on a discretized range from 0.1 to 0.9, with interval of 0.1.

- **vertices intervals**: Like the classification parameter, AbDG also grid searches vertices intervals in a discretized range from 2 to 9, with an interval of 1. However, unlike classification parameter, the results from this grid search are brought to the analysis, so that, it can be used for number of investigations, related to the intervals. This helps to compare complexity versus accuracy of the model when compared with other interval methods like MDLPC, EDADB and KBINS.

Various results derived by models of each variant of AbDG algorithm and also by sklearn's CART implementation, are then used to either accept or reject the research hypotheses based on the performance measured using F1 score. The external library - sk-learn [28] is used for all the metrics mentioned in the previous section.

3.4.2 System Design

The system is implemented using python and its modules like pandas [20] and numpy [26] and the external library from sk-learn [28]. The system is non-parametric and discovers its own parameters for each variant of algorithm, using a grid search method. It also uses various

training ratios to check its impact on generalization and overfitting of the models. The following are the versions of the applications used.

- python 3.7.1
- anaconda 2018.12
- numpy 1.16.2
- scipy 1.1.0
- sklearn 0.20.1
- matplotlib 3.0.2

3.4.3 Setup

The datasets used for this study are comparatively smaller and also the techniques employed do not demand high resources. Hence, a special setup is not required. However, for facilitating faster grid-search of two parameters, this requires a high number of runs and hence, a special shared system is employed as shown in the following configuration.

- AMD Ryzen Threadripper 2950X 16-Core Processor
- CPU MHz - 2249.307
- CPU(s) - 32
- L1d cache - 32K
- L1i cache - 64K
- L2 cache - 512K
- L3 cache - 8192K
- memory - 128GiB

3.5 Experimentation

This section presents the manner in which the methodology is carried out in practice. The experimentation work is carried out in three distinct phases - datasets collection, execution of experiments and analysis of results.

3.5.1 Datasets

This study uses different kinds of datasets from the UCI repository [1], some of which are considered as well-known supervised classification problems and most of the machine learning studies refer to these datasets to benchmark their conclusions. Selected datasets are then categorized in a few relevant groups as shown in Table 3.2.

As mentioned above, most of the data is consumed as it is except for categorical attributes in case of the decision tree class algorithm, which required encoding and one dataset ('wifi') has '-' in its numeric values, which has been converted to positive, just to avoid negative intervals

so that numeric columns are handled seamlessly by the code. Also, there are some datasets for which data pre-processing is done already by the owner of the data, for example, for Australian credit data missing values were already replaced by mode of the attribute if it is categorical and mean of the attribute if it is continuous.

The dataset selection is done consciously to avoid a large number of records and attributes, so that grid search processing becomes feasible, even with a normal computing powered machine.

Table 3.2: Datasets for used in this Research

Category	**Dataset**	**NI**	**NC**	**NN**	**NS**	**CD**
Binary Class Balanced	tictac [19]	958	2	0	9	332,626
	aus-credit [32]	690	2	6	8	383,307
	caesarian [2]	80	2	5	0	34,46
Binary Class Imbalanced	breast cancer [21]	569	2	30	0	212,357
	thoracic [39]	470	2	3	13	70,400
	wpbc [36]	198	2	33	0	151,47
Multi Class Balanced	iris [15]	480	3	4	0	160,160,160
	waveform [9]	5000	3	21	0	1657,1647,1696
	wifi [33]	2000	4	7	0	500,500,500,500
Multi Class Imbalanced	glass [13]	214	7	10	0	70,76,17,13,9,29
	car [8]	1728	4	0	6	1210,384,69,65
	phishing [22]	1353	3	9	0	702,103,548

Legend: *NI: Number of Instances, NC: Number of class labels, NN: Number of Numeric Attributes, NS: Number of Categorical Attributes, CD: Class Distribution*

3.5.2 Experiments

Based on various variants of the AbDG algorithm each dataset has undergone 648 runs which excludes nine runs for each classification parameter search that makes the total run count per dataset as 5832. In addition, the algorithm also searches for an information gain based filter so that the number of attribute is reduced without affecting accuracy of the c-Partite variant of AbDG. This run does not proceed further if it encounters loss of accuracy, as the main motive for this study is - comparing accuracy between AbDG and decision tree. This adds another set of 324 runs for each dataset. However, for real time classification problems, this characteristic can add a strong advantage to AbDG to facilitate a control over its execution time. Table 3.3 shows the number of variants and their options used for the experiments, along with the count of corresponding runs.

Table 3.3: Experiments for this Research

Classifier	**Run Type**	**Count**	**Options**
ABDG	Training Ratio	9	0.1 to 0.9
	Edge Type	9	P (p-Partite), C (c-Partite)
	Enhance-ment	2	0 (non-enhanced), 1 (enhanced)
	Interval Method	4	ESAD, KBINS, MDLPC, EDADB
	Intervals	8	2 to 8; applicable to ESAD and KBINS only
Decision Tree	Training Ratio	9	0.1 to 0.9
Total	$\mathbf{9\times2\times2\times(2+8\times2)+9}$	**657**	count per dataset

Legend: *MDLPC: Minimum Description Length Principal Cut Criterion and EDADB: Entropy-based Discretization According to Distribution of Boundary Points, KBINS: KBinsDiscretizer, ESAD: Equi Sized Attribute Division*

3.6 Conclusion

The goal of this study was to compare the performance of an Attribute-based Decision Graph with a Decision Tree classifier. And also, to compare it with its own variants. The research methodology used an empirical study approach to perform this comparison. This chapter particularly focused on how the comparison was executed during this study. Research Hypotheses were explained in Section 3.2. Section 3.3 provides details about the hypotheses evaluation methodology which is deployed for this empirical comparison. Section 3.4 focuses on the implementation details of this study, which includes classifier design, applications and systems used. Finally, section 3.5, briefly explains various experiments conducted during this study, along with the datasets and their properties. Acquired results from these experiments are produced and discussed in the following chapter.

Chapter 4
Experimental Results

4.1 Introduction

Research methodology briefed in previous chapter was employed to conduct the experiments and to obtain the observations of this study. The implementation is divided into three phases: the first phase comprised developing and implementing three algorithms - EDADB, MDLPC, ESAD for creating intervals from the training data samples; during the second phase AbDG algorithm is implemented for two edge-type based methods called c-Partite and p-Partite; and lastly, during the third phase fit and predict methods are implemented to conduct experiments over 12 UCI datasets.

This phase also requires grid searching of two inputs - classification parameter for optimum performance of the algorithm and for the input based interval methods such as ESAD, KBINS. KBINS method is an additional minor enhancement to the original proposition. All these phases lead to various experiments. This chapter discusses the results and observations of those experiments. Section 4.2 shows the accuracy based analysis of AbDG and DT along with analysis of their conciseness and maximum accuracy per training ratio. Section 4.3 shows a similar analysis for AbDG and its variants. It explains various other aspects involved during the experiments, for example, impact of training ratio on the performance of the algorithms, impact of the enhancements suggested based on a few observations, impact of number of attributes and number of class-values, etc. Finally, results summary is concluded in the section 4.4.

4.2 AbDG and Decision Tree

4.2.1 Accuracy

To show how AbDG algorithm performs classification, let's consider a model built on an Iris dataset [15] which contains 4 numeric attributes and 1 numeric target with 3 class labels - 0, 1, 2 where 0 is Setosa, 1 is Versicolour and 2 is Virginica. To allow a simplistic illustration let's use a minimally configured variant of the algorithm, i.e., let's use the p-Partite graph structure where each attribute's intervals are linked to only intervals which belongs to the next attribute in the descending order of the information gain criteria. Thus, the attribute order for this model can be obtained as Petal Width, Petal Length, Sepal Length and Sepal Width. Let's use ESAD based

interval method and keep 3 fixed intervals per attribute, which further simplifies the model representation. This model now comprises of 3 vertices per attribute and each connected to at most 3 other intervals belonging to the next attribute from the ordered list.

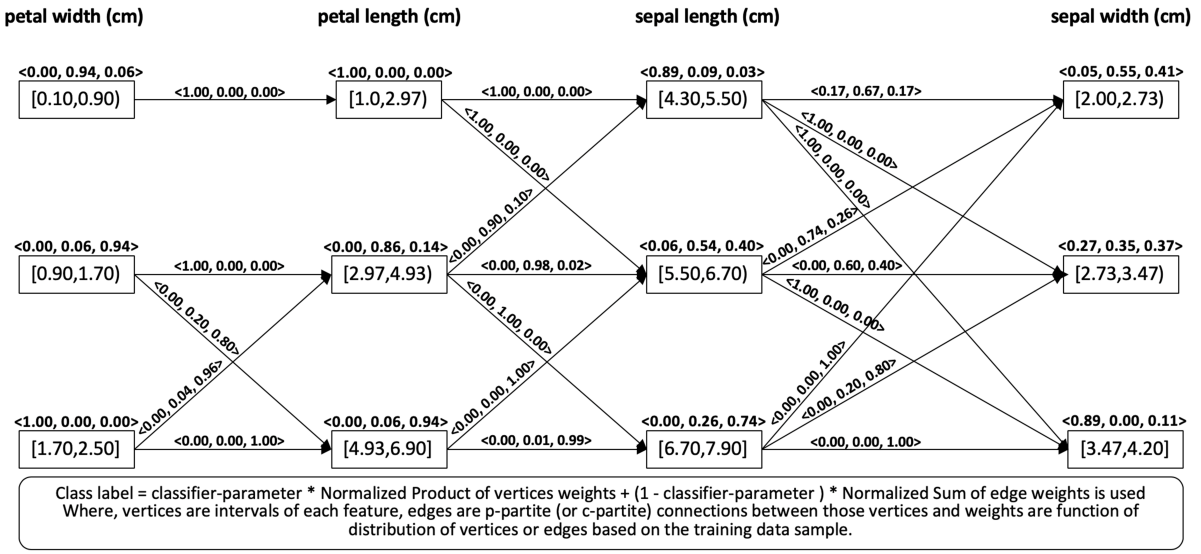

Figure 4.1: Attribute-based Decision Graph for Iris dataset

Then, the algorithm identifies weights using distributions of the classes and attributes of the training data samples, let's say using training ratio 70% of the total data samples. The algorithm uses equation (2.10) mentioned in section 2.2 to derive vertex weights and equation (2.15) to derive the edge weights, which requires underlying equations from (2.11) to (2.18). With this configuration, the model can be depicted as shown in the Figure 4.2.1. Each square box represents intervals of 1 attribute. Each such interval has 3 vertex-weights corresponding to 3 class labels. These weights are shown at the top of each square box. All these vertices are then connected by edges. These edges also carry weights corresponding to each class label and are shown at the top of each edge. Then as test data samples are received, each value of the test data sample gets converted to the corresponding numerical or categorical interval. Then, equation (2.24) is used to identify the corresponding class label for each test data sample using vertex and edge weights related to each attribute's value.

Similarly, sk-learn's decision tree classifier is used to identify the corresponding decision tree for the same training data samples, which can be depicted as shown in Figure 4.2.1. It gets created during the fit process of the decision tree. It starts at the top node and then based on the gini index, it keeps determining the two sets of rules for each node one level below, till it reaches the leaf node, which contains the corresponding class label. This tree then gets used during the predict process and based on the related attribute values, corresponding rules are applied to reach to one of the leaf nodes, which assigns the class label to that test data sample.

This way the decision tree and AbDG's various variants are defined for this study and corresponding experiments are run using those models. As shown in Table 4.1, maximum and minimum F1-score along with its standard deviation are obtained by summarizing those runs at dataset level. As explained in the Table 3.3, these scores have been derived using 648 runs for various AbDG variants, at multiple training ratios from 0.1 to 0.9.

Similarly, sk-learn's decision tree classifier is used at various training ratios to get its maximum and minimum F1-score per dataset, along with its standard deviation.

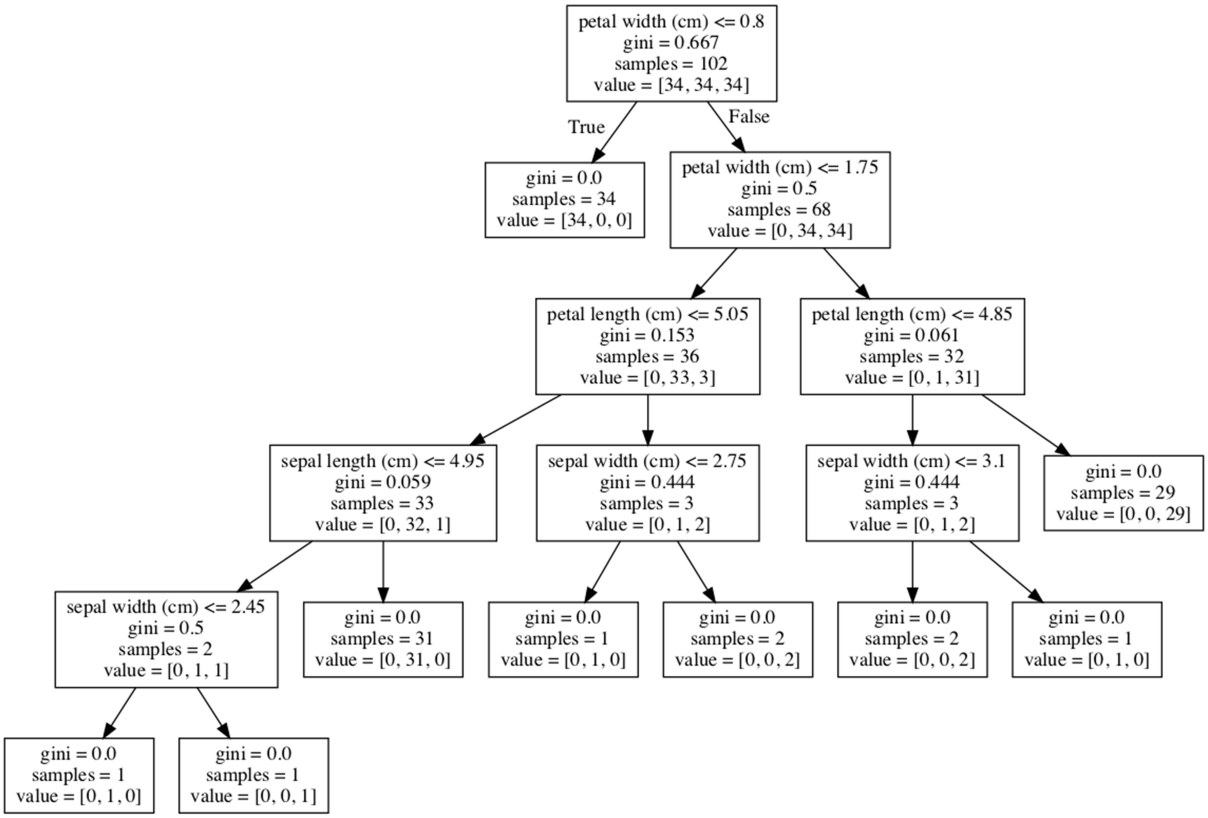

Figure 4.2: Decision Tree for Iris dataset

Thus, based on average rank which is calculated as the function of maximum F1-score, it can be clearly seen that AbDG performed much better than decision tree and hence, proves our Hypothesis 1 referred in section 3.2.

However, it can be noticed that, for most of the datasets decision tree has performed similarly to AbDG except for the datasets like caeserian, thoraric and waveform, where AbDG has outperformed decision tree quite considerably. At the same time, for smaller datasets with higher number of class labels the performance of AbDG may get impacted, for example decision tree has performed slightly better than AbDG for glass dataset. The reason for this could be that, there might be fewer samples per interval of each attribute available to determine the expected

class label.

It is also important to note that, the reason for higher standard deviations of AbDG is mainly because the statistics summarized in Table 4.1 includes runs for all variants of AbDG along with the runs for various interval sizes for variants like ESAD, KBINS, etc. Also, variants like MDLPC have contributed significantly to the higher standard deviation. Further analysis was conducted by referring to underlying results and metrics like confusion matrix and intermediate outputs like set of vertices and set of edges. Based on that analysis, it can be seen that, MDLPC could be overfitting the data using its intervals identified by its finer cut point based criteria, which might be leading to better accuracy during training phase, but, during testing or predicting unseen data samples, its performance reduces.

Table 4.1: Max, Mean, Min Std of F1-score for AbDG and DT

dataset	AbDG (F1-score)					DT (F1-score)				
	max	mean	min	std	rank	max	mean	min	std	rank
auscredit	0.990	0.840	0.718	0.062	1	0.841	0.808	0.762	0.026	0
breast_cancer	0.990	0.953	0.892	0.031	1	0.930	0.911	0.897	0.011	0
caesarian	1.000	0.900	0.436	0.167	1	0.570	0.520	0.360	0.064	0
car	0.983	0.980	0.976	0.005	1	0.983	0.938	0.816	0.052	0
glass	0.977	0.792	0.451	0.154	0	0.987	0.934	0.777	0.069	1
iris	1.000	0.950	0.857	0.056	0.5	1.000	0.952	0.835	0.046	0.5
phishing	0.978	0.944	0.858	0.045	1	0.887	0.862	0.815	0.023	0
thoraric	1.000	0.971	0.783	0.060	1	0.798	0.759	0.734	0.020	0
tictac	0.997	0.992	0.987	0.007	1	0.908	0.836	0.655	0.077	0
waveform	0.942	0.791	0.662	0.048	1	0.766	0.742	0.714	0.015	0
wifi	0.995	0.972	0.910	0.023	1	0.975	0.968	0.956	0.007	0
wpbc	1.000	0.932	0.660	0.107	1	0.858	0.709	0.620	0.065	0
average rank					0.875					0.125

Legend: *AbDG: Attribute-based Decision Graph, DT: Decision Tree, Max: Maximum, Min: Minimum, Std: Standard Deviation, Rank: maximum is better*

4.2.2 Conciseness

One of the primary motives behind this study is to understand if AbDG can be one of the algorithms, which may get preference over decision tree due to its advantage of accuracy plus due to its graph based origination, it is also interpretable like decision tree. Hence, to discover this aspect, both number of nodes and number of edges play a vital role. This study also focused on both aspects and as explained in research methodology, it uses a simple method to identify both numbers from both algorithms - AbDG and decision tree, for their respective runs with

maximum accuracy. The results shown in Table 4.2 and Table 4.3 are used for evaluation of Hypothesis 2 mentioned in section 3.2.

The average rank test for count of nodes from Table 4.2 has turned out non-conclusive as both algorithms received the same average rank. To resolve this scenario, the next property can be seen, which helps to understand the behaviour of 'number of nodes' for both algorithms. When node count's ratio with 'number of attributes' multiplied by 'number of classes' is compared between both algorithms then, it is clearly seen that, AbDG has an advantage over decision tree, because of its stable and predictive nature about its node counts. Though, it has wider range from 2.00 to 4.99, it is still much predictive compared to decision tree, which has range from 0.48 to 15.7.

Table 4.2: Minimum Nodes for Maximum F1-Score

			AbDG			DT		
dataset	NA	NC	NN	rank	ratio	NN	rank	ratio
auscredit	14	2	82	1	2.93	133	0	4.75
breast_cancer	30	2	161	0	2.68	29	1	0.48
caesarian	5	2	25	0	2.5	13	1	1.3
car	6	4	69	1	2.88	179	0	7.46
glass	10	7	186	0	2.66	11	1	0.16
iris	4	3	26	0	2.17	17	1	1.42
phishing	9	3	54	1	2.00	307	0	11.37
thoraric	16	2	78	1	2.44	155	0	4.84
tictac	9	2	54	1	3.00	157	0	8.72
waveform	21	3	178	1	2.83	989	0	15.7
wifi	7	4	130	0	4.64	91	1	3.25
wpbc	34	2	339	0	4.99	45	1	0.66
Average Rank				0.5			0.5	

Legend: *AbDG: Attribute-based Decision Graph, DT: Decision Tree, Rank: maximum is better, NA: Number of Attributes, NC: Number of Classes, NN: Number of Nodes, ratio: (number of nodes) / (number of attribute $\times$ number of classes)*

But, before concluding about the robustness of AbDG, let's look at the minimum edges count from Table 4.3. The number of edges required for optimum performance is lower for decision tree, when compared to AbDG.

It is also important to note that, for datasets like wifi, with only 7 numeric attributes and 4 class labels the count of edges for AbDG is very high i.e. 1302, whereas decision tree only has 90 edges. However, with dataset like waveform, the count of edges is quite high for decision tree, i.e., 989, than AbDG, which has 178 edges. Hence, this can be interpreted as even though edge

count for decision tree is less for most of the datasets, these tests are not sufficient enough to derive the conclusion using given number of datasets. Hence, it is recommended that, further exploration with more datasets and with alternative evaluation criteria be selected to evaluate the conciseness of both algorithms.

Thus, to conclude for Hypothesis 2, AbDG is not more concise than decision tree. Though, it has more predictability about the expected node and edge counts.

Table 4.3: Minimum Edges for Maximum F1-Score

			AbDG		DT	
dataset	NA	NC	NE	rank	NE	rank
auscredit	14	2	137	0	132	1
breast_cancer	30	2	320	0	28	1
caesarian	5	2	64	0	12	1
car	6	4	163	1	178	0
glass	10	7	318	0	10	1
iris	4	3	34	0	16	1
phishing	9	3	421	0	306	1
thoraric	16	2	140	1	154	0
tictac	9	2	648	0	156	1
waveform	21	3	454	1	988	0
wifi	7	4	1302	0	90	1
wpbc	34	2	938	0	44	1
Average Rank				0.25		0.75

Legend: *AbDG: Attribute-based Decision Graph, DT: Decision Tree, Rank: maximum is better, NA: Number of Attributes, NC: Number of Classes, NE: Number of Edges*

4.2.3 Training Ratio

For machine learning algorithms, the training phase is one of the most crucial aspects. To facilitate optimum training, the most vital role is played by the model training period per training data cycle and size of the data fed for each of those cycle. High training times lead to considerable reduction of number of parameters which can be explored during the training phase. Also, incorrect or insufficient data samples for a given algorithm may lead to incorrect inferences. Hence, it is important to select well distributed data per class label for each training cycle for most of the machine learning algorithms. A similar approach has been taken for this study which has revealed that, for most of the datasets AbDG required considerably lower amounts/ratios of the training data. However, this study also notices that, given the system design and setup,

the training and testing time frames per dataset and per parameter, were considerably larger for AbDG than compared to almost non-parametric algorithm decision tree. However, from the training ratio Table 4.4, it can be seen that, most of the optimum performances achieved by AbDG, were at a lower training ratio. This allows AbDG for quick learning, even with the small amount of the training data to produce better results, and it does not expect the availability of a full dataset for the complete training phase. This can also be visualized in a graph (Figure 4.3.4.) showing training ratio vs accuracy for various variants of AbDG.

Table 4.4: Minimum Training Ratio For Maximum F1-Score

dataset	AbDG (training_ratio)				DT (training_ratio)			
	max	min	std	rank	max	min	std	rank
auscredit	0.3	0.3	0.000	1	0.8	0.8	0.0	0
breast_cancer	0.3	0.3	0.000	1	0.8	0.8	0.0	0
caesarian	0.9	0.2	0.154	0	0.2	0.2	0.0	1
car	0.5	0.5	0.000	1	0.8	0.8	0.0	0
glass	0.8	0.8	0.000	0	0.3	0.3	0.0	1
iris	0.9	0.3	0.145	1	0.9	0.9	0.0	0
phishing	0.5	0.5	0.000	1	0.7	0.7	0.0	0
thoraric	0.9	0.2	0.227	1	0.9	0.9	0.0	0
tictac	0.6	0.6	0.000	0	0.5	0.5	0.0	1
waveform	0.4	0.4	0.000	1	0.9	0.9	0.0	0
wifi	0.5	0.5	0.000	1	0.9	0.9	0.0	0
wpbc	0.9	0.6	0.105	1	0.8	0.8	0.0	0
Average Rank				0.75				0.25

Legend: *AbDG: Attribute-based Decision Graph, DT: Decision Tree, Max: Maximum, Min: Minimum, Std: Standard Deviation, Rank: maximum is better*

4.3 AbDG variants

4.3.1 By vertices methods

As explained in the previous chapter, AbDG has been developed as a flexible algorithm, which allows multiple methods for its sections. One of them is creating vertices for numerical data, where it proposes to use three proven methods ESAD, MDLPC and EDADB. This study focuses on all of them and also adds another method known as KBinsDiscretizer provided by sk-learn toolkit. This allows verification of the flexibility aspect of AbDG algorithm. ESAD and KBINS are parametric methods unlike MDLPC and EDADB, which uses cut-point method to derive its algorithm. Looking at smaller size of data samples, the study constrained to use range 2 to 9 intervals for ESAD and KBINS. For categorical attributes, AbDG algorithm has specified only one method to select the intervals, where each value represents an interval (vertex) on its own.

Table 4.5: Std, Max, Min of F1-score for AbDG variants by vertices methods

dataset	ESAD			KBINS			MDLPC			EDADB		
	max	std	rank	max	std	rank	max	std	rank	max	std	rank
auscredit	0.896	0.025	2	0.896	0.025	2	0.718	0.000	3	0.990	0.003	1
breast_cancer	0.990	0.019	1	0.990	0.019	1	0.892	0.000	3	0.916	0.001	2
caesarian	1.000	0.041	1	1.000	0.041	1	0.436	0.000	2	1.000	0.043	1
car	0.983	0.005	1	0.983	0.005	1	0.983	0.005	1	0.983	0.005	1
glass	0.977	0.072	1	0.977	0.072	1	0.710	0.068	2	0.453	0.001	3
iris	1.000	0.072	1	1.000	0.072	1	0.967	0.003	2	1.000	0.020	1
phishing	0.978	0.013	1	0.978	0.013	1	0.888	0.021	3	0.973	0.000	2
thoraric	1.000	0.012	1	1.000	0.012	1	0.783	0.000	3	0.988	0.002	2
tictac	0.997	0.007	1	0.997	0.007	1	0.997	0.007	1	0.997	0.007	1
waveform	0.819	0.013	2	0.819	0.013	2	0.942	0.198	1	0.817	0.042	3
wifi	0.995	0.014	1	0.995	0.014	1	0.940	0.021	3	0.957	0.005	2
wpbc	1.000	0.033	1	1.000	0.033	1	0.660	0.000	2	0.660	0.000	2
Average Rank	1.17			1.17			2.17			1.75		

Legend: *KBINS: KBinsDiscretizer, ESAD: Equi Sized Attribute Division, Max: Maximum, Min: Minimum, Std: Standard Deviation, Rank: minimum is better*

This lead to four variants of AbDG algorithm by vertices method. Table 4.5 shows the maximum accuracy, its corresponding standard deviation and rank by comparing them with each other. It can be seen that, ESAD and KBINS performed much better than the other two methods and MDLPC was the lowest. As explained above, reason for MDLPC's reduced performance could be, its stricter method of finding most efficient cut points. Those cut-points get closely tied back to the distribution of training data samples leading to a tightly coupled inference model, which overfit on training data and fails to predict unseen data samples that accurately.

On the other hand, parametric interval methods like ESAD and KBINS create more coarse intervals, which allow loosely coupled distribution of data samples and help the model to learn

better and predict test data more effectively.

Also, it can be noticed that except for a couple of instances, standard deviation for most of these variants applied to selected datasets, is quite low, which indicates their robust nature for attaining the accuracy.

4.3.2 By edge type

Like vertices, the AbDG algorithm has also provided two types of flexible options for creating edges. One of them is known as p-Partite, where intervals belonging to an attribute can connect to other intervals only if they belong to its neighboring attribute. For creating this ordered list of attributes, the algorithm allows flexible approaches, i.e., required ordered list of attributes used by variant can be created by any method which helps to organize them in some way of information relevance. The original proposition mentions about the information gain, but it can be done by any other method as well. To allow simplistic exploration, this study focuses on information gain to get this ordered list of attributes.

Table 4.6: Maximum F1-Score Per Edge Type of AbDG

dataset	P (F1-score)			C (F1-Score)		
	max	std	rank	max	std	rank
auscredit	0.986	0.096	0	0.990	0.094	1
breast_cancer	0.990	0.054	0.5	0.990	0.058	0.5
caesarian	1.000	0.135	0.5	1.000	0.127	0.5
car	0.976	0.037	0	0.983	0.040	1
glass	0.931	0.180	0	0.977	0.194	1
iris	1.000	0.115	0.5	1.000	0.111	0.5
phishing	0.978	0.030	0	0.973	0.022	1
thoraric	1.000	0.054	0.5	1.000	0.053	0.5
tictac	0.997	0.029	0	0.987	0.034	1
waveform	0.814	0.078	0	0.942	0.066	1
wifi	0.995	0.042	0	0.993	0.040	1
wpbc	1.000	0.116	0.5	1.000	0.109	0.5
Average Rank			0.21			0.79

Legend: *C: c-Partite, P: p-Partite, Max: Maximum, Min: Minimum, Std: Standard Deviation, Rank: maximum is better*

The other variant in this category is known as c-Partite, where each interval gets connected to all other intervals except for those belonging to the same attribute. For this variant the ordered list of attributes is not required. However, it can be used to allow filtering of the attributes which do not contribute much to the model, making the model simpler and cost-efficient.

However, for this study each run checks, if there is any loss of accuracy, because one of the attribute is skipped using criteria like information gain. If the answer is yes, then, it does not

remove that attribute. Most of the datasets used for this study have small data samples and a small number of attributes. Hence, none of the run removed any of the attribute, as it would cost the loss of accuracy.

Looking at Table 4.6, it can be easily seen that, c-Partite variant has performed better or equal to p-Partite variant. But, it is also important to note that, the size of edges created by c-Partite is quite high compared to p-Partite variant. Hence, it is always important to weigh both options for accuracy versus system performance, especially knowing that loss of accuracy is not very high and if system performance is improvised largely because of p-Partite variant then, it allows for more exploration of the data.

4.3.3 Conciseness

Table 4.7: Minimum Nodes for Maximum F1-Score Per Vertices Method

			ESAD		KBINS		EDADB		MDLPC	
dataset	NA	NC	NN	rank	NN	rank	NN	rank	NN	rank
auscredit	14	2	43	2	43	2	82	3	36	1
breast_cancer	30	2	161	2	161	2	303	3	92	1
caesarian	5	2	26	3	26	3	25	2	10	1
car	6	4	69	1	69	1	69	1	69	1
glass	10	7	186	3	186	3	98	2	86	1
iris	4	3	26	2	26	2	44	3	20	1
phishing	9	3	54	2	54	2	48	1	48	1
thoraric	16	2	78	2	78	2	106	3	60	1
tictac	9	2	54	1	54	1	54	1	54	1
waveform	21	3	491	2	491	2	731	3	178	1
wifi	7	4	130	2	130	2	242	3	78	1
wpbc	34	2	339	3	339	3	206	2	85	1
Average Rank				2.08		2.08		2.25		1

Legend: *MDLPC: Minimum Description Length Principal Cut Criterion and EDADB: Entropy-based Discretization According to Distribution of Boundary Points, KBINS: KBinsDiscretizer, ESAD: Equi Sized Attribute Division, NA: Number of Attributes, NC: Number of Classes; NN: Number of Nodes, Rank: minimum is better*

While analyzing the conciseness for AbDG and decision tree, it is noticed that, certain variants always create a large number of nodes and edges. Hence, this study also includes analysis of conciseness for each variant of AbDG. This results in the statistics as shown in the Table 4.7 and 4.8. Looking at both of these statistics it can be seen that, MDLPC always keeps node and edge counts very low, when compared to other variants.

It mainly focuses on the high informative intervals, hence it filters out some of the intervals, which sometimes can reduce accuracy, hence, for most of the datasets MDLPC has lower accu-

racy when compared to other variants. EDADB performed the poorest with average rank as 2.25, followed by ESAD and KBINS with average rank as 2.08 for both and MDLPC outperformed all those 3 variants by quite a large margin of node counts with average rank as 1.

Like node counts, edge counts play quite an important role, in checking the conciseness, as it defines overall complexity of the graph. In this test as well, MDLPC outperforms the other three variants. This is quite a big advantage for MDLPC as it already avoids parameter searching by relying on cut points; plus it creates a smaller number of edges, leaving it as first choice of variant if system performance is the key criterion for the application domain.

Table 4.8: Minimum Edges for Maximum F1-Score Per Vertices Method

			ESAD		KBINS		EDADB		MDLPC	
dataset	NA	NC	NN	rank	NN	rank	NN	rank	NN	rank
auscredit	14	2	411	3	411	3	137	2	44	1
breast_cancer	30	2	320	2	320	2	612	3	124	1
caesarian	5	2	64	2	64	2	91	3	8	1
car	6	4	163	1	163	1	163	1	163	1
glass	10	7	318	2	318	2	125	1	508	3
iris	4	3	34	1	34	1	118	3	41	2
phishing	9	3	421	3	421	3	77	1	80	2
thoraric	16	2	140	2	140	2	235	3	100	1
tictac	9	2	648	1	648	1	648	1	648	1
waveform	21	3	2191	2	2191	2	2232	3	454	1
wifi	7	4	1302	3	1302	3	691	2	144	1
wpbc	34	2	938	3	938	3	336	2	83	1
Average Rank				2 .08		2.08		2.08		1.33

Legend: *MDLPC: Minimum Description Length Principal Cut Criterion and EDADB: Entropy-based Discretization According to Distribution of Boundary Points, KBINS: KBinsDiscretizer, ESAD: Equi Sized Attribute Division, NA: Number of Attributes, NC: Number of Classes; NN: Number of Nodes, Rank: minimum is better*

4.3.4 Training Ratio

To analyze the performance of all eight AbDG variants, it was pertinent to look at their accuracy against the training ratio. This helps in understanding performance as well as overfitting threshold. All F1-scores for all runs of that variant are grouped together to create the graph of mean accuracy as shown in Figure 4.3.4. The graph shows the accuracy of ESAD and KBINS for c-Partite as well as for p-Partite, matches exactly. This leads to the conclusion that, KBINS variant can be used in place of ESAD, as KBINS is a readily available solution, through external library sk-learn and also its system performance is a little better than, ESAD. However, to test their exact system performance in detail, it is recommended to conduct a separate study and observe the system performance, by using appropriate methods and measures. Followed

by ESAD and KBINS, the graph shows lines for EDADB's 2 variants (c-Partite and p-Partite) and then, for MDLPC. Though, c-Partite MDLPC has less accuracy while comparing to other variants, its performance shows a positive slope along with the training ratio, before showing the down slope after 0.8. This trend is shown by almost all c-Partite variants with little variation. The most important aspect to note with this analysis is, for almost all the variants 0.7 is a threshold training ratio and after that point the performance is seen declining. This is mainly due to overfitting of the data as the balance between training and testing data samples is shifted more towards training data samples.

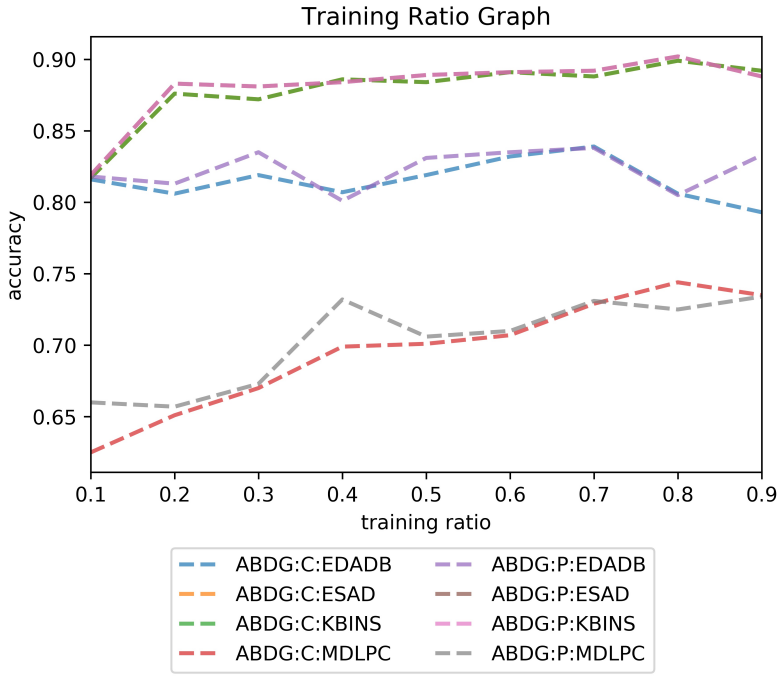

Figure 4.3: Mean F1-Score vs Training Ratio for all AbDG variants

4.3.5 Minor Enhancements

During the implementation phase of this study it was noticed that the original proposition, focuses on intervals created based on the training data samples. However, there are a few occurrences where unseen data samples do not fall under the same interval range as it was seen during the training phase, especially at low training ratios. This leads to excluding that value of the data sample, while calculating the class label prediction, i.e., the data point for that interval gets treated as an outlier. Though, there are a few scenarios observed with the selected datasets; there are such possibilities. Hence, this study theoretically proposes two minor enhancements to the original proposition, such as: a. using closed intervals for the each boundaries of the attribute space and b. if corresponding vertex, edge and their weights are not identified during

the predict process then, it can be filled in using minimum weight from all weights of that attribute. This will have some level of contribution from each value of the data instance, leading to more accurate results, otherwise the class label will be more positive in the absence of a value as these values are decimal values and no value means it will be either treated as 0 or as null. If it gets treated as 0 then, it nullifies even the high influential interval and if it is treated as null then, no-value leads to a biased calculation of that class label. However, with the selected datasets and evaluation methodologies, it could be seen from Table 4.9 that, at overall maximum accuracy (F1-score) level, there is no positive impact due to these enhancements. But, it is important to note that, during further analysis of experiments, it is observed that, there are 250 runs (125 configurations for each enhancement option, i.e., 0 as No enhancement and 1 as Enhancement) out of total 7776 runs (3888 configurations), for which accuracy (F1-score) changes due to enhancement options. 101 out of those 125 configurations have positive impact ranging from 0.003 to 0.087, whereas there are 24 configurations for which there is a negative impact ranging from 0.009 to 0.015. For example, as shown in Table 4.10, for the first set of configuration enhancement the accuracy increased for 2nd and 3rd class label; but for the second set of configuration the accuracy decreased for 1st class label. Hence, Hypothesis 4 mentioned in section 3.2, fails to get accepted or rejected by looking at the statistics from maximum accuracy level. But, there are a few scenarios for which these enhancements assist in increasing the accuracy.

Table 4.9: Maximum F1-Score Per Enhancement of AbDG

dataset	No Enhancement (F1-score)				Enhancement (F1-Score)			
	max	mean	min	std	max	mean	min	std
auscredit	0.990	0.776	0.395	0.095	0.990	0.776	0.395	0.095
breast_cancer	0.990	0.937	0.489	0.056	0.990	0.937	0.489	0.056
caesarian	1.000	0.841	0.397	0.132	1.000	0.841	0.397	0.132
car	0.983	0.932	0.851	0.039	0.983	0.932	0.851	0.039
glass	0.977	0.652	0.156	0.188	0.977	0.652	0.156	0.188
iris	1.000	0.921	0.425	0.112	1.000	0.912	0.425	0.114
phishing	0.978	0.930	0.774	0.026	0.978	0.930	0.774	0.026
thoraric	1.000	0.927	0.776	0.054	1.000	0.928	0.776	0.054
tictac	0.997	0.958	0.890	0.032	0.997	0.958	0.890	0.032
waveform	0.942	0.751	0.231	0.072	0.942	0.751	0.231	0.072
wifi	0.995	0.952	0.729	0.041	0.995	0.952	0.729	0.041
wpbc	1.000	0.829	0.648	0.113	1.000	0.829	0.648	0.113

Legend: *Max: Maximum, Min: Minimum, Std: Standard Deviation*

Table 4.10: Sample Configuraition Matrices for Enhancement Options

Configuration		Actual	Predicted Class		
	Enhanced	Class	C1	C2	C3
dataset: iris, training ratio: 0.2,		C1	25	0	0
vertices method: EDADB, Edge Type: P,	0	C2	17	8	0
Classification Parameter: 0.9		C3	0	1	24
dataset: iris, training ratio: 0.2		C1	25	0	0
vertices method: EDADB, Edge Type: P,	1	C2	15	10	0
Classification Parameter: 0.9		C3	0	0	25
dataset: caesarian, training ratio: 0.2		C1	25	3	
vertices method: KBINS, Edge Type: P,	0	C2	0	37	
Classification Parameter: 0.9,					
Vertices Intervals: 6					
dataset: caesarian, training ratio: 0.2		C1	24	4	
vertices method: KBINS, Edge Type: P,	1	C2	0	37	
Classification Parameter: 0.9,					
Vertices Intervals: 6,					

Legend: *Cn: Class Label 'n', Enh: Enhancement Option (0: No Enhancement, 1: Enhanced, EDADB: Entropy-based Discretization According to Distribution of Boundary Points, KBINS: KBinsDiscretizer)*

4.4 Conclusion

Below box plot shown in Figure 4.4 can be referred to summarize the above results and their observations from an accuracy point of view. Each box represents a major distribution of the accuracy of decision tree, AbDG and its primary variants. The blue line shows median value, green triangle shows mean and red horizontal small line shows outlier based on the major distribution of the given data.

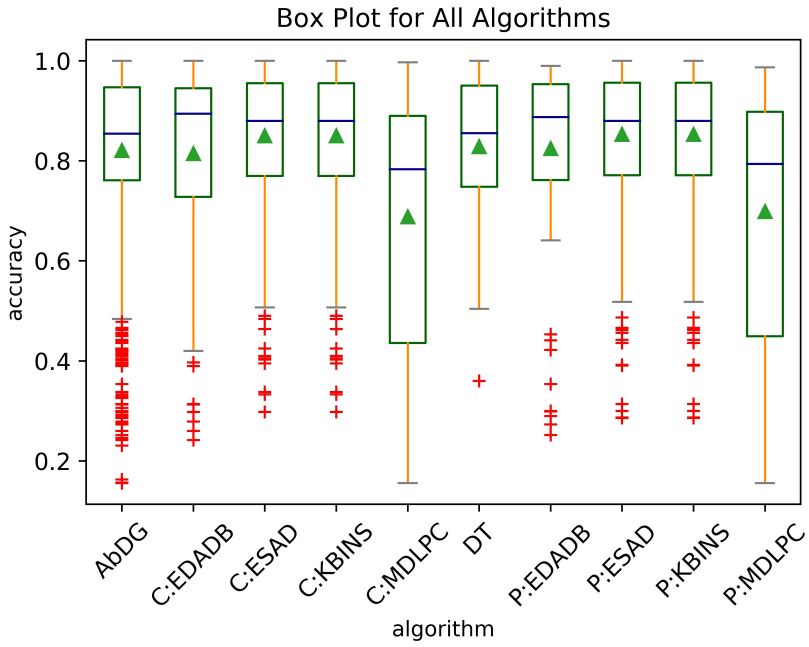

Figure 4.4: Box Plot Accuracy of all Algorithms

Legend: *AbDG: Attribute-based Decision Graph, C: c-Partite, P: p-Partite, MDLPC: Minimum Description Length Principal Cut Criterion, EDADB: Entropy-based Discretization According to Distribution of Boundary Points, KBINS: KBinsDiscretizer, ESAD: Equi Sized Attribute Division, Green triangles show mean and red horizontal small lines show outliers in the given data.*

The box for AbDG represents accuracy data for all of its variants which collectively has a wider range of outliers, where MDLPC has major contribution. Also, to note, its mean shown by green triangle is high. Decision tree runs are limited to 1 variant hence, its accuracy data distribution is cleanly represented in the box. The graph shows raised box for AbDG, representing higher range of accuracy for AbDG algorithm. Applying a similar principle to AbDG variants, it can be seen that, KBINS and ESAD's c-Partite variant mean is higher than other variants, which suggests that they both perform better than other variants. This proves Hypothesis 1 and 4.

For conciseness, it is seen that, AbDG's top performing variants like ESAD, KBINS created a large

number of nodes and edges. Though, in most cases their counts are in a close proportion to the number of attributes and number of class labels. Also, it can be seen that, MDLPC's performance is not efficient when compared to other variants. But, it created fewer nodes as well as edges. Thus, though, Hypothesis 2 fails in this case, MDLPC based variants of AbDG can be used more effectively to achieve the better conciseness and higher performance.

Minor enhancements suggested are as follow.

- assign last or first interval to an outlier test data value which is out of range because of the intervals which are created using training data samples;
- assign minimal weights for the data values where corresponding weights are not available especially for categorical attributes.

It is shown in the results that, there is no significant impact to the maximum accuracy after applying those enhancements. However, a few underlying runs' accuracy is influenced by those enhancements. Thus, Hypothesis 3 fails, with an important observation that, with further exploration with larger datasets and other evaluation criteria, there is a possibility that, these two enhancements may prove the value.

Also, the addition of the KBINS method as an option to vertices creation process, proves that there can be additional ways to achieve similar or better accuracy, using newer state of art options. This has already been enabled by the algorithm, by flexibly allowing various options.

Chapter 5
Conclusion

Recently, there have been developments in machine learning fields. A number of new algorithms and their variants are being proposed quite regularly. However, there is still a need of more interpretable, explainable algorithms. So, that the acceptance and rejection of the models become more explainable to the end users who have a bigger and wider impact of selecting a particular model and its decision rules. This highlights one of the major motivations for re-looking at decision trees and their generalized extension known as decision graphs. However, existing methods which use decision trees and decision graphs have their own limitations. This lead to a uniquely built method based on decision graphs, which has advantages inherent from decision tree i.e. generating decision rules for ease of learning, model selection and model interpretation; along with other advantages inherent from decision graphs such as robustness of pattern matching techniques which allows detailed analysis of the data at the level of attributes, along with considering the impact from neighbouring attributes of the dataset. Despite being advantageous, this algorithm has not been yet explored much; hence, it becomes inevitable to do such empirical study to extend the results achieved by the original proposition. This study not only aims to extend the results and the capabilities of the algorithm; but, also to create an in-depth analysis of the algorithm and its variants, so that it allows more exploration of the algorithm or similar techniques in future.

In this research, the algorithm is implemented for training a model which includes creating vertices from the given attribute space and connecting them using the edges based on two edge creation approaches - p-partite and c-partite. The training process concluded by attaching corresponding distribution based weights to those vertices and edges. For testing this model, implementation included assigning vertices, edges and their weights to each testing dataset element based on its value. Finally, classifier combines the weights for assigned vertices and edges, to derive the predicted class for each data instance. The prediction is then evaluated using standard classification evaluation metrics known as F-beta score, which includes the effect of precision and recall. It also helps to evaluate biased results more accurately using a weighted approach. For further investigations purposes, two other metrics were referred to, known as accuracy-score and confusion matrix. This research also included implementation and evaluation of AbDG's variants based on interval methods and minor enhancements as proposed in the previous chapter. All these implementations and evaluations are then compared to sk-learn's optimized CART based decision tree classifier; using various classification problems such

as binary-class and multi-class with balanced and imbalanced datasets.

The detailed empirical analysis revealed that, AbDG algorithm outperformed decision tree for almost all the datasets. It created nodes and edges, as a function of the number of attributes and the number of classes, leading to more predictable behaviour. But, AbDG always created a large number of nodes and edges when compared to those of decision tree. This becomes more complex as the size of the dataset grows and the number of its attributes increases. Despite of this drawback, AbDG can be highly recommended if accuracy is more important for the given supervised classification problem. Also, its is recommended that, though performance of MDLPC variant is not as good as other AbDG variants, it can be further explored to use more effectively as it creates fewer nodes and edges. AbDG is also highly recommended whenever interpretable model creation algorithms are required. Another major advantage of AbDG is that, the model can be easily read using intermediate outputs (vertices and edges, along with their weights) derived right after the training process. In such cases the output does not only show the interpretable rules, but since these rules are in numeric format, it can be easily fed or interfaced with other systems for more robust usage.

5.1 Future work

Results and their observations discussed in the above section leads to numerous ideas about the implementation and applications of the AbDG algorithms. There could be some particular set of scenarios where AbDG fits better than any other algorithm. Hence, it is recommended that such a high performing algorithm must receive more attention from the research community and more such experiments should be performed with larger datasets, a wider variety of alternatives for options like interval-methods, information relevance ordering, etc.

One of the most important hurdles faced during this study was, the total execution time for such a high volume of experiments, which forced the selection of datasets with limited number of samples, attributes and class labels. To allow wider acceptance of this algorithm it is vital to create a high performing open source library for its implementation which can handle larger volumes in a fraction of time.

In addition to this, MDLPC or similar approaches can be further explored with this algorithm, as it created fewer nodes and edges and it is also one of the non-parametric algorithms. Both these factors will definitely save computation time. The major focus of this study should be, to improve the accuracy of the algorithm for MDLPC based variant.

It is also highly beneficial to explore AbDG algorithm as one of the alternatives to decision tree and decision tree based algorithms, especially where interpretability of the model is important. Such experiments will not only increase the acceptance of AbDG algorithm in that domain, but, it will also help in providing guidance for interpretable algorithms' research, by showing that, interpretable models can be created using pure distribution based methods and not only rule

based methods. Also, another benefit of such model is the set of numeric rules, which is created as an output of the fit process. These rules (also called vertices and edges weights) can be easily consumed by other systems which need to be interfaced with AbDG based system.

Nowadays, ensembling is quite popular in the current machine learning era. It can also be explored for AbDG by ensembling it with its own variants or with other algorithms. It may also prove a great alternative to current ensemble algorithms like random forest, adaboost, gradient tree boosting, etc.

Thus, there are multiple ways to further explore AbDG, but the most important recommendation is, it needs to be explored for some of the real world use cases where it interacts with real time data or if possible to the time-series data and assists in creating a model which can make real value add to the machine learning field as well as to some application domains.

References

[1] (2017). UCI machine learning repository.

[2] Amin, M. and Ali, A. (2018). Performance evaluation of supervised machine learning classifiers for predicting healthcare operational decisions.

[3] An, A. and Cercone, N. (1999). Discretization of continuous attributes for learning classification rules. In *Proceedings of the Third Pacific-Asia Conference on Methodologies for Knowledge Discovery and Data Mining*, PAKDD '99, pages 509–514, London, UK, UK. Springer-Verlag.

[4] Aruna, S., Rajagopalan, D. S., and Nandakishore, L. V. (2011). An empirical comparison of supervised learning algorithms in disease detection. *International Journal of Information Technology Convergence and Services (IJITCS)*, 1(4).

[5] Bertini, J. R. (2013). Attribute-based decision graphs for multiclass data classification. *2013 IEEE Congress on Evolutionary Computation*.

[6] Bertini, J. R., do Carmo Nicoletti, M., and Zhao, L. (2017). Attribute-based decision graphs: A framework for multiclass data classification. *Neural networks : the official journal of the International Neural Network Society*, 85:69–84.

[7] Biggs, N., Lloyd, E. K., and Wilson, R. J. (1986). *Graph Theory, 1736-1936*. Clarendon Press, New York, NY, USA.

[8] Bohanec, M. and Rajkovic, V. (1988). Knowledge acquisition and explanation for multi-attribute decision making. *8th Intl. Workshop on Expert Systems and their Applications, Avignon, France*, pages 59–78.

[9] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Statistics/Probability Series. Wadsworth Publishing Company, Belmont, California, U.S.A.

[10] Breslow, L. A. and Aha, D. W. (1997). Simplifying decision trees: A survey. *Knowl. Eng. Rev.*, 12(1):1–40.

[11] Caruana, R. and Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 161–168, New York, NY, USA. ACM.

[12] de Nijs, J. (2011). Decision dags - a new approach. In *cs.brown.edu*.

[13] Evett, I. W. and Spiehler, E. J. (1987). Rule induction in forensic science. In *Rule Induction in Forensic Science*, pages 107–118. Online Publications.

[14] Fayyad, U. M. and Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *IJCAI*.

[15] Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals*

Eugenics, 7:179–188.

[16] Hunt E, M. J. and P, S. (1966). *Experiments in Induction*. Academic Press, New York, NY, USA.

[17] Kelarev, A. V., Stranieri, A., Yearwood, J., and Jelinek, H. F. (2012). Empirical study of decision trees and ensemble classifiers for monitoring of diabetes patients in pervasive healthcare. *2012 15th International Conference on Network-Based Information Systems*, pages 441–446.

[18] Liu, H., Hussain, F., Tan, C. L., and Dash, M. (2002). Discretization: An enabling technique. *Data Min. Knowl. Discov.*, 6:393–423.

[19] Matheus, C. J. and Rendell, L. A. (1989). Constructive induction on decision trees. In *IJCAI*, page 645.

[20] McKinney, W. (2011). pandas : powerful python data analysis toolkit.

[21] Mickalski, R. S., Mozetic, I., J., H., and Lavrack, H. (1986). The multi purpose incremental learning system aq15 and its testing application to three medical domains. In *Proceedings of the 5th National Conference on Artificial Intelligence*. Morgan Kaufmann publisher.

[22] Mohammad, R. M., Thabtah, F. A., and McCluskey, L. (2012). An assessment of features related to phishing websites using an automated technique. In *7th International Conference for Internet Technology and Secured Transactions, ICITST 2012, London, United Kingdom, December 10-12, 2012*, pages 492–497.

[23] Mues, C., Baesens, B., Files, C. M., and Vanthienen, J. (2004). Decision diagrams in machine learning: An empirical study on real-life credit-risk data. In *Diagrams*.

[24] Negro, A. (2018). Machine learning and graph: An introduction. In *Graph-Powered Machine Learning*.

[25] O, O. A., O, A. O., and T, O. S. (2014). Empirical Study of Decision Tree and Artificial Neural Network Algorithm for Mining Educational Database. *African Journal of Computing and ICT*, 7(2):191–193.

[26] Oliphant, T. (2006–). NumPy: A guide to NumPy. USA: Trelgol Publishing. [Online; accessed ¡today¿].

[27] Oliver, J. (1992). *Decision graphs: an extension of decision trees*. Citeseer.

[28] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[29] Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

[30] Quinlan, J. R. (1979). Discovering rules from large collections of examples: a case study. In Michie, D., editor, *Expert Systems in the Microelectronic Age*, pages 168–201. edinburgh, edinburgh-ad.

[31] Quinlan, J. R. (1986). Induction of decision trees. *Mach. Learn.*, 1(1):81–106.

[32] Quinlan, J. R. (1987). Simplifying decision trees. *Int. J. Man-Mach. Stud.*, 27(3):221–234.

[33] Rohra, J., Thakur, P., J.N., S., Perumal, B., and Bhatt, R. (2016). User localization in an indoor environment using fuzzy hybrid of particle swarm optimization and gravitational search algorithm with neural networks.

[34] Rokach, L. and Maimon, O. (2014). *Data Mining With Decision Trees: Theory and Applications*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2nd edition.

[35] Schaeffer, S. (2007). Graph clustering. *Computer Science Review*, 1:27–64.

[36] Street, N., Mangasarian, O. L., and Wolberg, W. H. (1995). An inductive learning approach to prognostic prediction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 522–530. Morgan Kaufmann.

[37] Zhang, Y., Xin, Y., Li, Q., Ma, J., Li, S., Lv, X., and Lv, W. (2017). Empirical study of seven data mining algorithms on different characteristics of datasets for biomedical classification applications. *BioMedical Engineering OnLine*, 16.

[38] Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., and Sun, M. (2018). Graph neural networks: A review of methods and applications. In *eprint arXiv:1812.08434*.

[39] Zikeba, M., Tomczak, J. M., Lubicz, M., and 'Swikatek, J. (2013). Boosted svm for extracting rules from imbalanced data in application to prediction of the post-operative life expectancy in the lung cancer patients. *Applied Soft Computing*.

[40] Ziwei Zhang, Peng Cui, W. Z. (2018). Deep learning on graphs: A survey. In *eprint arXiv:1812.04202*.