Abstract

Data Mining is widening its scope by using new algorithms applicability in several domains which assist us in gaining more knowledge and further in decision making. Decision Tree is one such important technique which builds a tree structure by incrementally breaking down the datasets in smaller subsets. Decision Trees can be implemented by using popular algorithms such as ID3, C4.5 and CART. The present study considers ID3 algorithm to build a decision tree. Entropy and Information Gain are used by ID3 to construct a decision tree.

Introduction

Data mining is relatively a new concept emerged in 90's as a new approach to data analysis and knowledge discovery. Data mining has originated from statistics and machine learning as an interdisciplinary field. The degree of information is growing in bulk and people's ability of using information technology to collect and produce data is significantly enhancing. In such a huge volume of data, discovering useful knowledge and improving the effectiveness of information utilization are the challenges to be addressed. It was under this background, Data Mining evolved. Data mining have several classes of tasks. Classification is the task of generalizing known structure to apply to new data. For example, an e-mail program might attempt to classify an e-mail as "legitimate" or as "spam".

Classification is one of the most common tasks in data mining to solve a wide range of real problems, such as credit scoring, bankruptcy prediction, medical diagnosis, pattern recognition, and multimedia classification. It is recognized as a powerful way for companies to develop effective knowledge-based decision models to gain competitive advantages. The current classification methods include decision trees (DTs), neural networks, logistic regression, and nearest neighbor. DTs are widely used for classification because of their ability to manage noisy data and return well organized results that are easily interpreted, computationally efficient, and robust. Several algorithms, such as ID3 and C4.5, have been devised for the construction of DTs and are applied in several areas, including risk management, customer relationship management, text categorization, medical diagnosis, and credit rating of loan applicants.

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy). Leaf node (e.g., Play) represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

Related Work

In traditional DT algorithms, a label (target variable) of a tuple is either categorical or Boolean that is, the algorithms operate under the assumption that the labels are nominal and flat. However, several practical situations involve more complex classification scenarios, in which the label to be predicted can occur in a variety of types, for example, continuous variable, hierarchically related variable, or both. To bridge this gap in the research, Hu et al. proposed a method called the continuous label classifier (CLC) algorithm, which added a novel dynamic partition mechanism to partition continuous labels during the construction of a DT [2]. For hierarchy related problems with class labels, Chen et al. proposed a hierarchical class label classifier (HLC) algorithm that can construct a DT from data with hierarchical class labels. They also demonstrated the unsuitability of traditional methods for hierarchical class labels and showed the differences resulting from their use. The HLC and CLC algorithms address several crucial problems associated with the induction of DT from data with either continuous or hierarchical labels. However, both algorithms have the following substantial limitations:

- The HLC algorithm can be used only for class labels and is unable to manage continuous labels.
- For the CLC algorithm, if the partitioned ranges are excessively fragmentary because of extensive variation in continuous-label distribution, they are not useful in prediction accuracy or an understanding of the induced models. This condition makes us difficult to use these rules to form business strategies.

In addition, these algorithms cannot directly manage a situation in which the labels are continuous in the data and can be naturally organized as a hierarchical structure. This type of problem is encountered in several domains, including the insurance industry, functional genomics, supply chains, risk management, and object recognition. In functional genomics, biologists create a set of possible gene functions that are subsequently organized hierarchically. A hierarchical structure is necessary because each gene may have multiple functions. It is crucial to obtain an interpretable model to understand the interactions between various genes. In each of these cases, the labels are continuous in the data and can be naturally organized as a hierarchical structure of class labels, which defines an abstraction over class labels.

Structured continuous-label classification is a variety of classification in which the label is continuous in the data, but the goal is to classify data into classes that are a set of predefined ranges and can be organized in a hierarchy. In the hierarchy, the ranges at the lower levels are more specific and inherently more difficult to predict, whereas the ranges at the upper levels are less specific and inherently easier to predict. Therefore, both prediction specificity and prediction accuracy must be considered when building a decision tree (DT) from any kind of data[3]. ID3 is a novel classification algorithm for learning DT classifiers from data with structured continuous labels. This approach considers the distribution of labels throughout the hierarchical structure during the construction of trees without requiring discretization in the preprocessing stage.

Survey on decision tree algorithms

ID3 (Iterative Dichotomiser 3): This is a decision tree algorithm introduced in 1986 by Quinlan Ross. It is based on Hunts algorithm. The tree is constructed in two phases. The two phases are tree building and pruning ID3 uses information gain measure to choose the splitting attribute. To build decision tree, information gain is calculated for each and every attribute and select the attribute with the highest information gain to designate as a root node. Label the attribute as a root node and the possible values of the attribute are represented as arcs. Then all possible outcome instances are tested to check whether they are falling under the same class or not. If all the instances are falling under the same class, the node is represented with single class name, otherwise choose the splitting attribute to classify the instances.

C4.5: This algorithm is an extension to ID3 developed by Quinlan Ross. It is also based on Hunt's algorithm. C4.5 handles both categorical and continuous attributes to build a decision tree. In order to handle continuous attributes, C4.5 splits the attribute values into two partitions based on the selected threshold such that all the values above the threshold as one child and the remaining as another child. It also handles missing attribute values. C4.5 uses Gain Ratio as an attribute selection measure to build a decision tree. It removes the biasness of information gain when there are many outcome values of an attribute. At first, calculate the gain ratio of each attribute. The root node will be the attribute whose gain ratio is maximum. C4.5 uses pessimistic pruning to remove unnecessary branches in the decision tree to improve the accuracy of classification

CART: CART stands for Classification And Regression Trees introduced by Breiman. It is also based on Hunt's algorithm. CART handles both categorical and continuous attributes to build a decision tree. It handles missing values. CART uses Gini Index as an attribute selection measure to build a decision tree .Unlike ID3 and C4.5 algorithms, CART produces binary splits. Hence, it produces binary trees. Gini Index measure does not use probabilistic assumptions like ID3, C4.5. CART uses cost complexity pruning to remove the unreliable branches from the decision tree to improve the accuracy.

Random Forest: Random Forest decision tree was developed by Leo Breiman. A Random Forest is a collection of simple tree predictors, such that each tree produces a response when a set of predictor values are given as input. Similar to CART algorithm. Random Forest also works both for classification and regression problems. When solving classification problems, the response or the output appears in the form of a class membership, which associates or classifies, a set of independent predictor values with the matching category present in the dependent variable. When solving regression problems the output or response of the tree is an approximation of the dependent variables given the predictors.

CHAID*:* CHAID is a type of decision tree technique, based upon adjusted significance testing (Bonferroni testing). CHAID can be used for prediction as well as classification, and for detection of interaction between variables. In practice, CHAID is often used in the context of direct marketing to select groups of consumers and predict how their responses to some variables affect other variables, although other early applications were in the field of medical and psychiatric research.

Survey on Classification Algorithms

Naive Bayes Algorithm: The naive Bayesian classifier is a type of probabilistic classifier. This method uses Bayes' theorem and also assumes that each and every feature in a class are highly independent, that is the appearance of a feature in a particular category is not connected with to the presence of any other feature. The naive Bayesian classification done based on the prior probability and likelihood of a tuple to a class.

The training data samples are partitioned based on class labels. Each data partition id denoted using class labels *Ci* where *i=1, 2,...,m* and each class will have set of tuples represented as *Xj* where *j=1,2,...,n. 2*. After training process, if an unknown tuple *X* is given for classification then the classifier will find the posterior probability of Ci for give tuple *X* and assign *X* to class Ci if and only if posterior probability of *Ci* given *X* is greater than posterior probability of *Cj* given X where *1<=j>=m* and *j* not equal to *i*. The posterior probability of *Ci* for give tuple *X* can be calculated as, Posterior probability *(Ci given X)=(Likelihood of tuple X with class Ci * Class prior probability Ci)/ Prior probability of tuple X.* The Naive Bayesian classifier works with both continuous and discrete attributes and works well for real time problems. This method is very fast and highly scalable. The drawback of this technique is when a data set which has strong dependency among the attribute is considered then this method gives poor performance.

K-nearest neighbor: KNN is a classification method which very simple and works practically. The training sample consists of set of tuples and class labels associated with that. This algorithm works for arbitrary number of classes. KNN uses distance function to map the samples with classes. Classification process of KNN will find the distance between the given test instance *X* with that of existing samples $y_1, y_2, ..., y_k$. The nearest neighbors are found and based on the voting of neighbors, the majority neighborhood class is assigned to the test samples. The distance between the samples can be found by Euclidean method or Manhattan method or Murkowski method, if the values are continuous. If the value used is categorical the Hamming distance is used. The probability of assigning a sample *X* to that of a class *C* is based on the number of neighbors considered, denoted as *k*.

$$\text{Probability of X to a class C} = \frac{\sum_{i=1}^{k} dis\tan ce(c, c(yi))}{k}$$

This will improve the probability estimates of KNN which in turn will improve the performance of classification. KNN can perform well in many situations and it particularly suits well for multi-model classes as well as applications in which an object can have many labels. The drawback of KNN is it involves lot of computation and when the size of training set taken is large then the process will become slow.

SVM: Support vector machine is a classification model based on supervised training method for binary classification. Here the training samples belong to any one of two classes. Based on the training data samples, Support Vector Machine builds a prediction model that will classify the new sample properly to any one of the two classes. Here input to the system is the training samples $x_1, x_2 ... x_n$ with the class labelled as y, and the samples are mapped in the plane. Then a proper hyper plane is identified that optimize the classification result. The SVM uses mapping function called kernel function φ.There are four basic kernel function given as, *{Linear function: K(xi ,xj)= φ (xi Transpose). φ (xj)},{Polynomial function: K(xi ,xj)=(γ . φ (xi Transpose). φ (xj)+r)d , γ >0} ,{Radial basis function (RBF): K(xi ,xj) = exp(-γ ||xi -xj ||2), γ >0} ,{Sigmoid function: K(xi ,xj) = tanh(γ . φ (xi Transpose). φ (xj)+r)}* where, γ ,r and d are kernel parameters. This prediction model is simple and very accurate. This is used to model real time problems which are more complex and have many attributes. The major drawback of this model is it suits only for binary classification.

Neural Network: An artificial neural network (ANN), often just called a "neural network" (NN), is a mathematical model or computational model based on biological neural networks, in other words, is an emulation of biological neural system. It consists of an interconnected group of artificial neurons and processes information using a connectionist approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase.

Decision Tree

Decision trees are an efficient nonparametric method that can be applied either to classification or to regression tasks. They are hierarchical data structures for supervised learning whereby the input space is split into local regions in order to predict the dependent variable. A decision tree can be seen as a graph $G = (V, E)$ consisting of a finite, non- empty set of nodes (vertices) V and a set of edges E. Such a graph has to satisfy the following properties:

- The edges must be ordered pairs (v, w) of vertices, i.e., the graph must be directed;
- There can be no cycles within the graph, i.e., the graph must be acyclic;
- There is exactly one node, called the root, which no edges enter;
- Every node, except for the root, has exactly one entering edge;
- There is a unique path a sequence of edges of the form $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$ from the root to each node;
- When there is a path from node v to w, v /= w, v is a proper *ancestor* of w and w is a proper *descendant* of v. A node with no proper descendant is called a *leaf* (or a *terminal*). All others are called *internal* nodes (except for the root).

Root and internal nodes hold a test over a given data set attribute (or a set of attributes), and the edges correspond to the possible outcomes of the test. Leaf nodes can either hold class labels (classification), continuous values (regression), (non-) linear models (regression), or even models produced by other machine learning algorithms. For predicting the dependent variable value of a certain instance, one has to navigate through the decision tree. Starting from the root, one has to follow the edges according to the results of the tests over the attributes. When reaching a leaf node, the information it contains is responsible for the prediction outcome. For instance, a traditional decision tree for classification holds class labels in its leaves.

Each path from the root to a leaf is actually a conjunction of attribute tests, and the tree itself allows the choice of different paths, that is, a disjunction of these conjunctions. Other important definitions regarding decision trees are the concepts of *depth* and *breadth*. The average number of layers (levels) from the root node to the terminal nodes is referred to as the *average depth* of the tree. The average number of internal nodes in each level of the tree is referred to as the *average breadth* of the tree. Both depth and breadth are indicators of tree complexity, that is, the higher their values are, the more complex the corresponding decision tree is. In Fig. 2.1, an example of a general decision tree for classification is presented. Circles denote the root and internal nodes whilst squares denote the leaf nodes. In this particular example, the decision tree is designed for classification and thus the leaf nodes hold class labels.

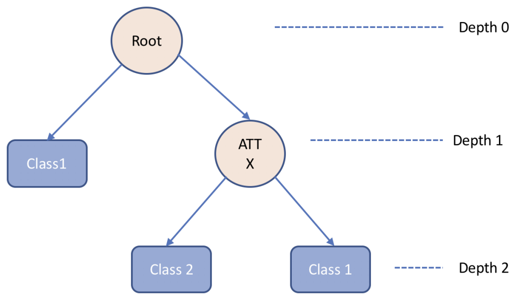

Figure 1: Basic Decision Tree

ID3 ALGORITHM:

The main ideas behind the ID3 algorithm are:

- Each non-leaf node of a decision tree corresponds to an input attribute, and each arc to a possible value of that attribute. A leaf node corresponds to the expected value of the output attribute when the input attributes are described by the path from the root node to that leaf node.
- In a "good" decision tree, each non-leaf node should correspond to the input attribute which is the *most informative* about the output attribute amongst all the input attributes not yet considered in the path from the root node to that node. This is because we would like to predict the output attribute using the smallest possible number of questions on average.

ID3 pseudo code

Algorithm 1: ID3 algorithm

```
Input: Training set $D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$
       Attribute set $A = \{a_1, a_2, \cdots, a_d\}$
Output: a decision tree
1 Generate node
2  If samples in $D$ belong to the same class. Then
3    The node is labeled as class C leaf node; Return
4  End if
5  If $A = \phi$ or the values in A are same in $D$, Then
6    The node is labeled as leaf node; Return
7  End if
8  Maximum Information Entropy is chosen as a heuristic strategy to select the optimal
splitting attribute $a_i$ from A;
9  For every value $a_i^v$ in $a_i$ do
10  Generate a branch for node; $D_v$ is the sample subset that has value $a_i^v$ from $a_i$ in $D$
11    If $D_v$ is empty. Then
12      The branch node is labeled as a leaf node; Return
13    Else
14      Take Tree Generate $(D_v,\ A \backslash \{a_i\})$ as the node that can continue be divided
15    End if
16  End for
```

Entropy is used to determine how informative a particular input attribute is about the output attribute for a subset of the training data. Entropy is a measure of uncertainty in communication systems introduced by Shannon. It is fundamental in modern information theory.

Entropy

In information theory, entropy is a measure of the uncertainty about a source of messages. The more uncertain a receiver is about a source of messages, the more information that receiver will need in order to know what message has been sent. For example, if a message source always sends exactly the same message, the receiver does not need any information to know what message has been sent, it's always the same. The entropy of such a source is zero: there is no uncertainty at all. On the other hand, if a source can send possible messages and each message occurs independently of the preceding message with equal probability, then the uncertainty of the receiver is maximized. The receiver will need to ask $log_2 n$ yes/no questions to determine which message has been sent, *i.e.* the receiver needs $log_2 n$ *bits* of information. The average number of bits required to identify each message is a measure of the receiver's uncertainty about the source and is known as the entropy of the source.

Consider a source which can produce messages $\{m_1, m_2, \ldots\ldots m_n\}$. All messages are produced independently of each other, and the probability of producing message m_i is p_i. For such a source S with a message probability distribution $P = \{p_1, p_2, \ldots\ldots p_n\}$, the entropy $H(P)$ is

$$H(P) = -\sum_{i=1}^{n} p_i \log(p_i).$$

In the equation above, $log(p_i)$ means $log_2(p_i)$ - from now on we will assume that all logarithms are to the base two. If a set T of records from a database (*i.e.* the training set for building the decision tree) is partitioned into k classes$\{C_1, C_2, \ldots\ldots C_k\}$on the basis of the output attribute, then the average amount of information (measured in bits) needed to identify the class of a record is $H(P_T)$, where P_T, is the probability distribution of the classes, estimated from the data as

$$P_T = \left(\frac{|C_1|}{|T|}, \frac{|C_2|}{|T|}, \ldots, \frac{|C_k|}{|T|} \right).$$

The notation $|C_i|$ means the number of elements in set C_i. For the weather data, where *play* is the output attribute, we have for the entire dataset T. Note that here we identify the entropy of the set $H(T)$ with the entropy of the probability distribution of the members of the set, $H(P_T)$.

Information gain

Now consider what happens if we partition the set on the basis of an input attribute *X* into subsets T_1, T_2 ,… T_n . The information needed to identify the class of an element of *T* is the weighted average of the information needed to identify the class of an element of each subset:

$$H(X,T)=\sum_{i=1}^{n}\frac{|T_i|}{|T|}H(T_i).$$

In the context of building a decision tree, we are interested in how much information about the output attribute can be gained by knowing the value of an input attribute *X*. This is just the difference between the information needed to classify an element of *T* before knowing the value of *X*, *H(T),* and the information needed after partitioning the dataset *T* on the basis of knowing the value of *X, H(X,T)*. We define the *information gain* due to attribute X for set *T* as

$$\mathrm{Gain}(X,T)=H(T)-H(X,T).$$

In order to decide which attribute to split upon, the ID3 algorithm computes the information gain for each attribute and selects the one with the highest gain.

The problem of attributes with many values

The simple ID3 algorithm above can have difficulties when an input attribute has many possible values, because Gain *(X,T)* tends to favor attributes which have a large number of values. It is easy to understand why if we consider an extreme case. Imagine that our dataset *T* contains an attribute that has a different value for every element of *T.* This could arise in practice if a unique record ID was retained when extracting *T* from a database—for example a patient ID number in a hospital database. Such an attribute would give the maximum possible information gain, since all the training data can be correctly classified by examining its value. It would result in a decision tree in which all nodes below the root node were leaf nodes. This tree, however, would be completely useless for classifying new data: there would be no arc corresponding to the value of the ID attribute. Moreover, there is no reason to suspect that such an attribute would have any causal relationship with the output attribute. The problem also arises when an attribute can take on many values, even if they are not unique to each element. The solution based on considering the amount of information required to determine the value of an attribute *X* for a set T. This is given by $H(P_{X,T})$, where $P_{X,T}$ is the probability distribution of the values of *X* :

$$P_{X,T}=\left(\frac{|T_1|}{|T|},\frac{|T_2|}{|T|},\ldots,\frac{|T_n|}{|T|}\right).$$

The quantity $H(P_{X,T})$ is known as the *split information* for attribute X and set T. We will call it $SplitInfo(X,T) = H(P_{X,T})$. It is suggested that rather than choosing the attribute with the biggest $Gain(X,T)$, we select the one with the biggest $GainRatio(X,T)$, where

$$\mathrm{GainRatio}(X,T) = \frac{\mathrm{Gain}(X,T)}{\mathrm{SplitInfo}(X,T)}.$$

Note that $GainRatio(X,T)$ might not always be defined. Quinlan (1986) specifies a *gain ratio criterion*, which says that we should select the attribute X with the highest $GainRatio(X,T)$ from amongst those attributes with average-or-better $Gain(X,T)$.

Explanation of ID3 with an example

For instance, the following table informs about decision making factors to play tennis at outside for previous 14 days.

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Table 1: Decision making factors to play tennis

We need to calculate the entropy first. Decision column consists of 14 instances and includes two labels: yes and no. There are 9 decisions labeled yes, and 5 decisions labeled no.

$\text{Entropy(Decision)} = -\, p(\text{Yes}) \,.\, \log_2 p(\text{Yes}) - p(\text{No}) \,.\, \log_2 p(\text{No})$
$\text{Entropy(Decision)} = -\,(9/14) \,.\, \log_2(9/14) - (5/14) \,.\, \log_2(5/14) = 0.940$

Now, we need to find the most dominant factor for decisioning.

Wind factor on decision

$$Gain(Decision,\ Wind) = Entropy(Decision) - \sum [\, p(Decision|Wind) \,.Entropy(Decision|Wind)]$$

Wind attribute has two labels: weak and strong. We would reflect it to the formula.

$$Gain(Decision, Wind) = Entropy(Decision) - [\, p(Decision|Wind{=}Weak) \,.\, Entropy(Decision|Wind{=}Weak)] \\ - [\, p(Decision|Wind{=}Strong).\ Entropy(Decision|Wind{=}Strong)]$$

Now, we need to calculate *(Decision|Wind=Weak)* and *(Decision|Wind=Strong)* respectively.

Weak wind factor on decision

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
13	Overcast	Hot	Normal	Weak	Yes

Table 2: Weak wind factor on decision

There are 8 instances for weak wind. Decision of 2 items are no and 6 items are yes as illustrated below.

$\text{Entropy(Decision|Wind=Weak)} = -\, p(\text{No}) \,.\, \log_2 p(\text{No}) - p(\text{Yes}) \,.\, \log_2 p(\text{Yes})$
$\text{Entropy(Decision|Wind=Weak)} = -\,(2/8) \,.\, \log_2(2/8) - (6/8) \,.\, \log_2(6/8) = 0.811$

Strong wind factor on decision

Day	Outlook	Temp.	Humidity	Wind	Decision
2	Sunny	Hot	High	Strong	No
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
14	Rain	Mild	High	Strong	No

Table 3: Strong wind factor on decision

Here, there are 6 instances for strong wind. Decision is divided into two equal parts.

$$\text{Entropy(Decision|Wind=Strong)} = -\,p(\text{No}) \,.\, \log_2 p(\text{No}) - p(\text{Yes}) \,.\, \log_2 p(\text{Yes})$$
$$\text{Entropy(Decision|Wind=Strong)} = -\,(3/6) \,.\, \log_2(3/6) - (3/6) \,.\, \log_2(3/6) = 1$$

Now, we can turn back to Gain(Decision, Wind) equation.

$$Gain(Decision,\ Wind) = Entropy(Decision) - [\, p(Decision|Wind{=}Weak) \,.\, Entropy(Decision|Wind{=}Weak)] - [\, p(Decision|Wind{=}Strong) \,.\, Entropy(Decision|Wind{=}Strong) \,]$$
$$= 0.940 - [\, (8/14) \,.\, 0.811 \,] - [\, (6/14).\ 1] = 0.048$$

Calculations for wind column is over. Now, we need to apply same calculations for other columns to find the most dominant factor on decision.

Other factors on decision

We have applied similar calculation on the other columns.

1- Gain(Decision, Outlook) = 0.246
2- Gain(Decision, Temperature) = 0.029
3- Gain(Decision, Humidity) = 0.151

As seen, outlook factor on decision produces the highest score. That's why, outlook decision will appear in the root node of the tree.

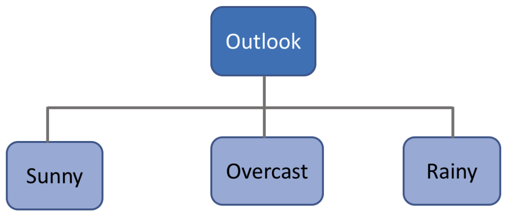

Figure 2: outlook decision

Now, we need to test dataset for custom subsets of outlook attribute.

Overcast outlook on decision

Basically, decision will always be yes if outlook were overcast.

Day	Outlook	Temp.	Humidity	Wind	Decision
3	Overcast	Hot	High	Weak	Yes
7	Overcast	Cool	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes

Table 4: Overcast outlook on decision

Sunny outlook on decision

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Table 5: Sunny outlook on decision

Here, there are 5 instances for sunny outlook. Decision would be probably 3/5 percent no, 2/5 percent yes.

1- (Outlook=Sunny|Temperature)gain = 0.570
2- (Outlook=Sunny|Humidity)gain = 0.970
3- (Outlook=Sunny|Wind)gain = 0.019

Now, humidity is the decision because it produces the highest score if outlook were sunny. At this point, decision will always be no if humidity were high.

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No

Table 6: Sunny outlook on decision

On the other hand, decision will always be yes if humidity were normal

Day	Outlook	Temp.	Humidity	Wind	Decision
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Table 7: Sunny outlook on decision

Finally, it means that we need to check the humidity and decide if outlook were sunny.

Rain outlook on decision

Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
10	Rain	Mild	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Table 8: Rain outlook on decision

1- Gain(Outlook=Rain | Temperature)
2- Gain(Outlook=Rain | Humidity)
3- Gain(Outlook=Rain | Wind)

Here, wind produces the highest score if outlook were rain. That's why, we need to check wind attribute in 2nd level if outlook were rain.
So, it is revealed that decision will always be yes if wind were weak and outlook were rain.

Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes

Table 9: Rain outlook on decision

What's more, decision will be always no if wind were strong and outlook were rain.

Day	Outlook	Temp.	Humidity	Wind	Decision
6	Rain	Cool	Normal	Strong	No
14	Rain	Mild	High	Strong	No

Table 10: Rain outlook on decision

So, decision tree construction is over. We can use the following rules for decisioning.

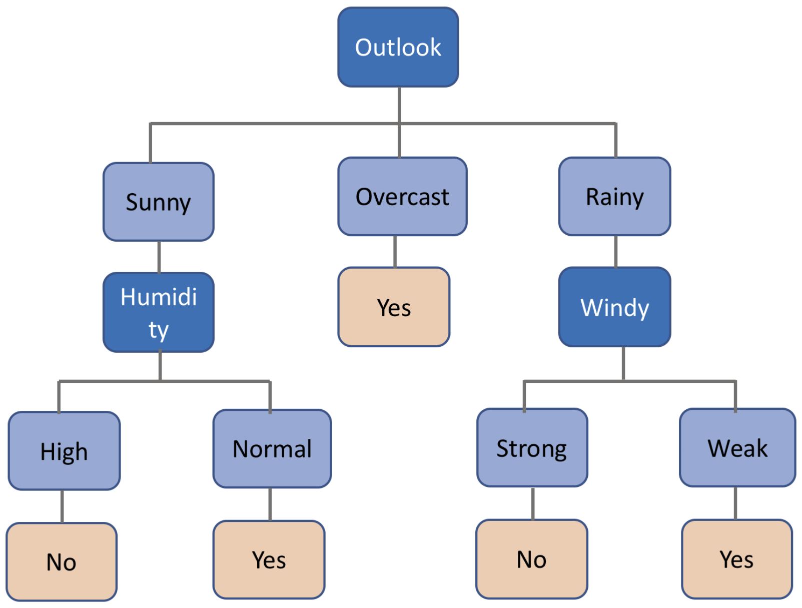

Figure 3: Final Decision Tree

Implementation & Evaluation

The algorithm implemented is the ID3 algorithm for decision tree classification. The algorithm has been implemented in Java Programming language and was run on a Mac Book Pro with an Intel i7 processor and 8GB of RAM. The datasets used are previously split into training data which consists of a set of tuples along with their associated class labels, and test data on which the consists of some tuples for which the algorithm predicts the class label based on the trained data. The implementation is divided into two phases –

1. Training, and
2. Classification

1. Training - Our program trains the dataset recursively. The 'train()' method in the program initializes the decision tree variable with null and negative 1 (-1), because these values are set later. Further, it calls the 'trainRecursive()' method, thus passing in the decision tree and training data. The parameters for 'trainRecursive()' method are the current node, the current set of training data and the list of already used columns. The choice of keeping track of the used columns was made to avoid the division of the subset based on the same attribute more than once.

   The base case for 'trainRecursive()' is the case where all items in the class have the same classifier. If this is the case, it creates a leaf node and the recursion stops. If this is not the case, it finds the next attribute to divide the remaining data by calling 'findNextQuestion()'. The method, 'findNextQuestion' calls 'split()' for every attribute. The HashMap returned from split(), along with the value returned from 'calculateHS()' (used to find the H(S) value for the remaining subset) is used to calculate the gain for each column of the data. The attribute with the highest gain is returned, and this is the question that divides the remaining data into subsets. If at least one subset is empty, the majority class is found instead of dividing the data. If not, ‘trainRecursive()’ is called on each subset of the data, as long as no subsets are empty. If any are, the majority class is found.

2. Classification – Classification also uses recursion. For each example in the test data, the ‘classify()’ method calls the method ‘classifyHelper()’. The ‘classifyHelper()’ method is a recursive function that traverses the tree according to the data in the example. The base case for this function is if the node has no children, which means it is a leaf node. It returns the location of the classifier in strings, which the method ‘classify()’ prints to the output file. If the node is not a leaf node, ‘classifyHelper()’ finds the child node that is correct for the current example, and then calls ‘classifyHelper()’ on this node.

Although we’ve generally implemented the Decision tree - ID3 algorithm, we’ve made certain unique design choices. For example, we have chosen to have the method ‘split()’ return a HashMap mapping Strings to HashMap’s, because, we figured this would be the fastest and most efficient way to access this data further down the program. HashMap’s are very fast for lookup and retrieving. Further, we’ve also chosen to have a parameter in ‘trainRecursive()’ that is an Array-List of already used columns so that the program would not choose a question it has already used. However, different branches are allowed to ask the same question, so we had to make deep copies of the Array-List, so that, an addition in one branch would not add it to the Array-List in another branch and also mark it as already used. We have also chosen to find the majority when a subset is empty because it is impossible to further divide an empty subset.

CART algorithm was written in python with Gini Index being the attribute selection measure. This is a simple algorithm with the first step being loading the dataset, for which the helper function ‘load_csv()’ is used to load the file and ‘str_column_float()’ is used to convert string numbers to float. Next, the function ‘decisionTree()’ was developed to manage the applications of CART algorithm, by first creating a tree from the training dataset and then using the tree to make predictions on the test data.

We have chosen to validate our algorithm using 5 folds, so around 270 records will be in use every fold. The method ‘validate_algorithm()’ is used to evaluate the algorithm with cross validation and the method accuracy() is used to calculate the accuracy of prediction.

We have tested the program by using multiple different training and test files that were provided online in UCI Machine Learning Repository, and also on some data set, we created for this project. There were no unknown bugs found.

Tennis Data - We validated our Algorithm in 2 steps. At first, we manually deduced a decision tree on a small dataset, called Tennis dataset, which we have explained in the Algorithm section of this report. Then, we gave the same dataset as Training file (Table 11) to our implemented code and the Test file give to the algorithm is as shown in the table below.

Outlook	Temp.	Humidity	Wind
Sunny	Mild	Normal	Weak
Sunny	Cool	High	Strong
Sunny	Cool	High	Strong
Overcast	Mild	Normal	Weak
Overcast	Hot	Normal	Weak
Rain	Cool	Normal	Strong

Table 11: Tennis test data

Both algorithms predicted the class labels for the above test data and is as shown in the table below.

Outlook	Temp.	Humidity	Wind	Decision
Sunny	Mild	Normal	Weak	Yes
Sunny	Cool	High	Strong	No
Sunny	Cool	High	Strong	No
Overcast	Mild	Normal	Weak	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No

Table 12: Prediction

As we can see from the table above, the predicted results are seen to be accurate and the memory used is very less because of the size of the dataset. The algorithm execution time is seen to be equal to 40ns for ID3 and around 800ns for CART.

Breast Cancer Dataset – This breast cancer dataset was obtained from the University of Wisconsin Hospitals, Madison meant for classification purposes with class label being either 4 for malignant or 2 for benign, a total 699 instances and 10 attributes excluding the class. Some of the attributes are Clump Thickness, Cell size, shape, Bland Chromatin etc. The missing values in the dataset was filled in by the mean values of the respective columns.

As we executed the algorithm for this dataset, previously split into train and test data, we observed that the best split was observed for the attribute, Uniformity of Cell size in both the cases. The predicted class labels for the test set for this dataset is shown in the table below.

Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class Label
9	1	2	6	4	10	7	7	2	2
10	4	7	2	2	8	6	1	1	4
7	6	10	5	3	10	9	10	2	4
8	10	10	8	5	10	7	8	1	4
5	8	8	10	5	10	8	10	3	4
3	1	1	1	2	1	2	1	1	2
5	3	2	8	5	10	8	1	2	2
10	4	4	10	2	10	5	3	3	4
5	7	7	1	5	8	3	4	1	4
10	5	8	10	3	10	5	1	3	4
5	10	10	6	10	10	10	6	5	4
10	4	4	10	6	10	5	5	1	4
7	9	4	10	10	3	5	3	3	4
3	3	5	2	3	10	7	1	1	2
3	3	5	2	3	10	7	1	1	2
1	1	1	1	2	1	2	1	1	2
10	8	8	2	3	4	8	7	8	4
1	1	1	1	3	1	1	1	1	2
1	1	3	1	2	1	1	1	1	2
10	5	10	3	5	8	7	8	3	4

Table 13: Prediction of Breast Cancer Data

The class label column consists of the predicted values. The ID3 algorithm's accuracy, execution time and the memory usage for this dataset is seen as –

Accuracy	**98.4%**
Execution Time	**0.006**
Memory usage	**2.69 Mb**

The class label column consists of the predicted values. The CART algorithm's accuracy, execution time and the memory usage for this dataset is seen as –

Accuracy	**96.4%**
Execution Time	**2.45ms**
Memory usage	**11.9 Mb**

IRIS Data – This dataset consists of 150 instances with totally 5 attributes including the class variable. The attributes are –

1. Sepal length in cm.
2. Sepal width in cm.
3. Petal length in cm.
4. Petal width in cm.
5. Class – Iris Setosa(0), Iris Versicolour(1), Iris Virginica (2)

This dataset was run on the ID3 & CART algorithm.

The accuracy for Gini Split, the execution time of the program & amount of memory utilized is as shown in the table below.

Accuracy	**90.3%**
Execution Time	**9.67ms**
Memory usage	**11.13 Mb**

The accuracy for ID3, the execution time of the program & amount of memory utilized is as shown in the table below.

Accuracy	**93.26%**
Execution Time	**0.008ms**
Memory usage	**2.7Mb**

Bank Note Authentication – This dataset consists of 1372 instances with 5 attributes and meant for classification. The attributes are –

1. Variance of Wavelet transform image
2. Skewness of Wavelet transformed image
3. Curtosis of Wavelet transformed image
4. Entropy of image
5. Class

This dataset was run on the ID3 & CART decision tree algorithm.

The accuracy for Gini Split, the execution time of the program & amount of memory utilized is as shown in the table below.

Accuracy	97.18%
Execution Time	12.03ms
Memory usage	12.68 Mb

The accuracy for ID3, the execution time of the program & amount of memory utilized is as shown in the table below.

Accuracy	98.26%
Execution Time	0.0035ms
Memory usage	2.57Mb

Social Network Ad Dataset – This dataset was tested to visualize the data points and to show that decision tree algorithm results in Overfitting. This dataset contains 400 tuples with dimensions namely UserID, Gender, Age, & estimated salary, with the class variable called as 'Purchased' taking the values 0 & 1. Using this dataset, we show the confusion matrix, which determines the number of correct v/s wrong predictions and also look at the overfitting problem caused by using decision trees. The Matplotlib library present in python was used to plot the data points. This dataset was only run over the CART algorithm to get the plot. The confusion matrix is as shown in the table below.

CM	0	1
0	62	6
1	3	29

The dataset is split into 300 tuples of training set & 100 tuples as test set. The confusion matrix table generated by predicting over test set, shows that there was a total of 9 wrong predictions and 91 correct predictions, making the accuracy to be close to 90%.

To visualize the data points of the training set & test set, we plot the data points with the green area representing '1' and red area representing '0'. The figures are shown below.

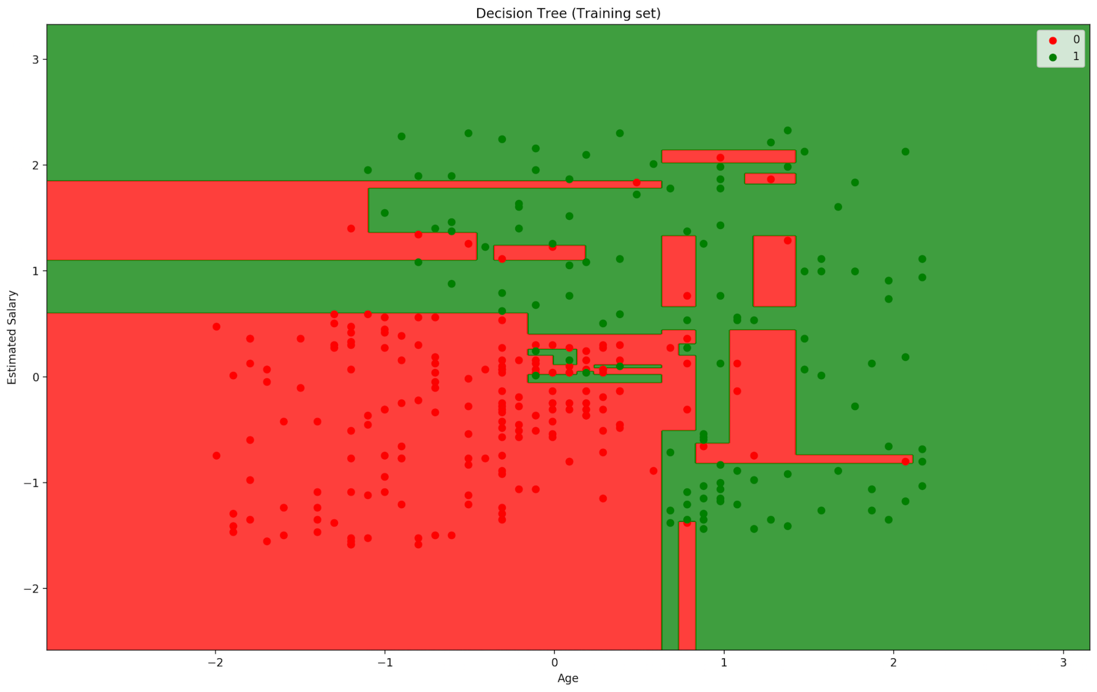

Figure 4: Data points of Training

The plot above is the data points in the training set. We can set that this plot is unique, nothing similar to either a straight line or a curve which differentiates the data. Here, we see that the prediction boundary is composed of vertical and horizontal lines. From the plot above, one can visualize the algorithm is trying to make splits based on conditions on the independent variable age & estimated salary. The algorithm is trying to catch every single user in the right category which can be termed as Overfitting. This concept can be shown clearly by plotting the test set points as shown in the figure 5.

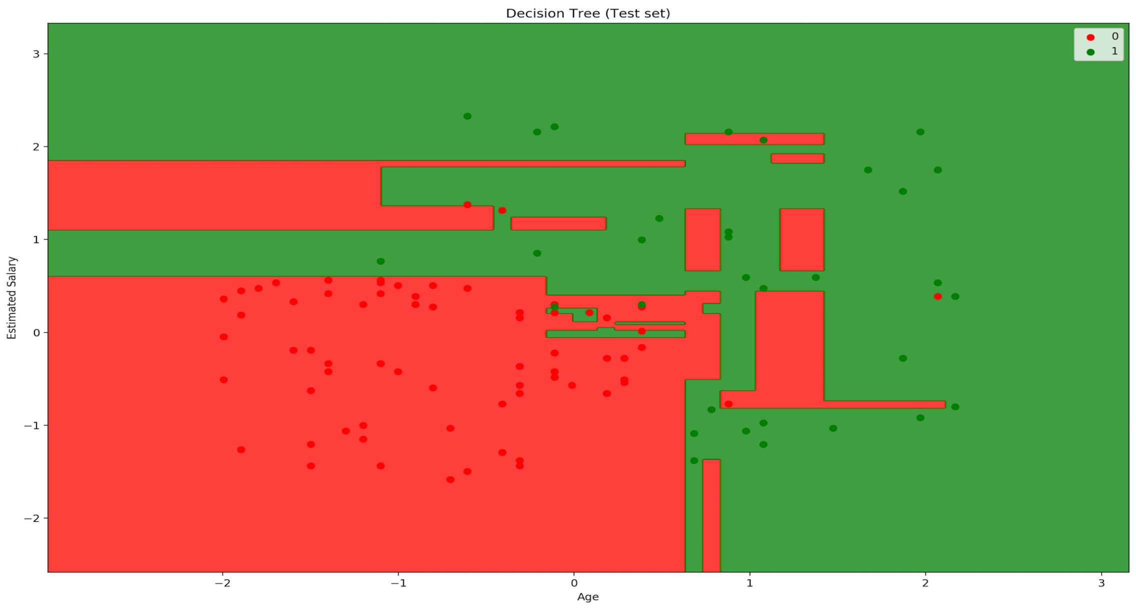

Figure 5: Data points of Testing

Consider the red rectangle in the above figure at the position (1.2, 0.8) & (0.6,0.8). This red rectangle in the green area without any red points represents the concept of overfitting which has resulted from training the data too well.

Thus, in order to improve the algorithm, we need to employ methods like tree pruning which reduces the size of decision trees by removing sections of the tree that provide little power to classify instances. It also reduces complexity of the final classifier and hence improves predictive accuracy by the reduction of overfitting.

The execution time for just the algorithm was seen to be 8.1ms and the memory used is 10.89Mbytes. The execution time of this code is seen to be **3.03s** for this dataset because of the plotting function implemented and memory usage was found to be around **230 Mbytes.**

Comparison of ID3 with SVM & Deep Learning(TensorFlow)

SVM – The support vector machine algorithm present in python sci-kit learn library was used. The test v/s train split was the same as that used for decision tree. The accuracy, execution time and memory usage were tested for breast cancer dataset, IRIS dataset, Social network ad dataset & Bank note authentication dataset and is as shown in the table below.

Dataset	Accuracy	Execution time	Memory usage
IRIS data	88%	4.456ms	13.5Mb
Breast Cancer data	94.66%	5.6ms	12.67Mb
Social network ad	89.56%	5.1ms	12.23Mb
Banknote Authentication	92%	15.5ms	24.67Mb

Table 14: Comparison of Accuracy, time & memory

Deep Learning (TensorFlow) – Google's TensorFlow eager execution model for deep learning was used over the same datasets as tested in the decision tree. The table below shows the execution time, Accuracy, Memory Usage with respect to TensorFlow model.

Dataset	Accuracy	Execution time	Memory usage
IRIS data	99.1%	7.44ms	20.77Mb
Breast Cancer data	99%	11.67ms	22.96Mb
Social network ad	89.2%	12.78ms	22.63Mb
Banknote Authentication	97%	29.012ms	32.6Mb

Table 15: Comparison of Accuracy, time & memory

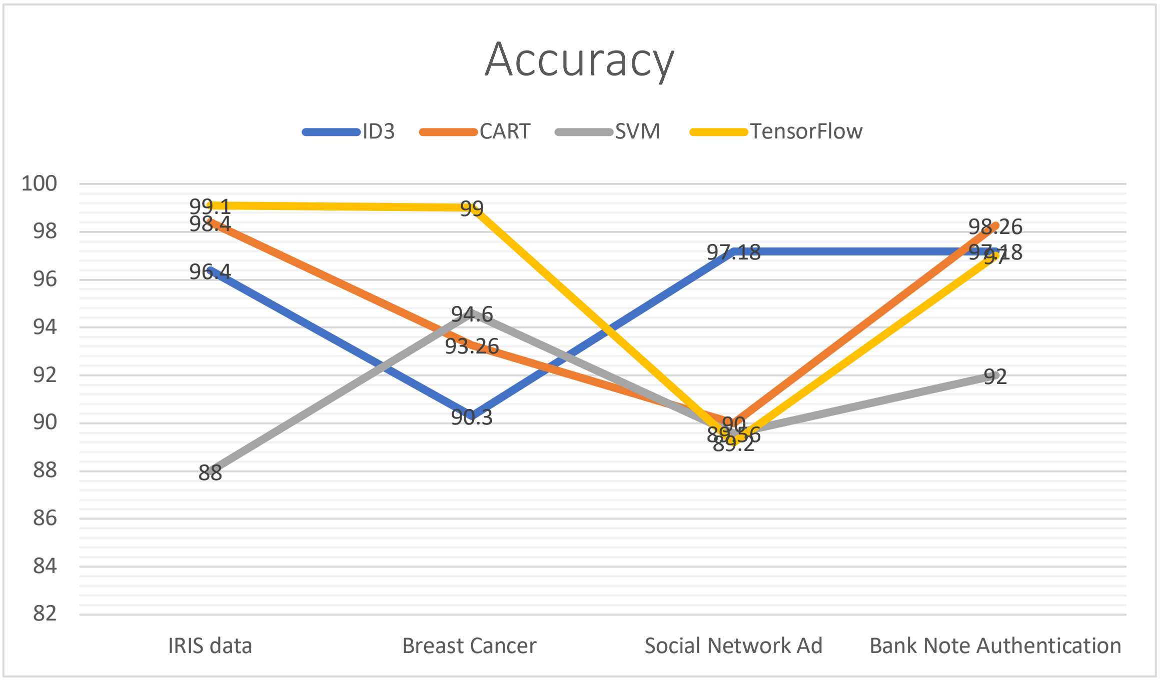

Figure 6: Accuracy Plot

Accuracy - The above graph represents the comparison of accuracy values for ID3, CART, SVM and Deep Learning(TensorFlow). As we can see that, for the datasets used, we have achieved accuracies almost in the same range for all the 4 types of algorithms.

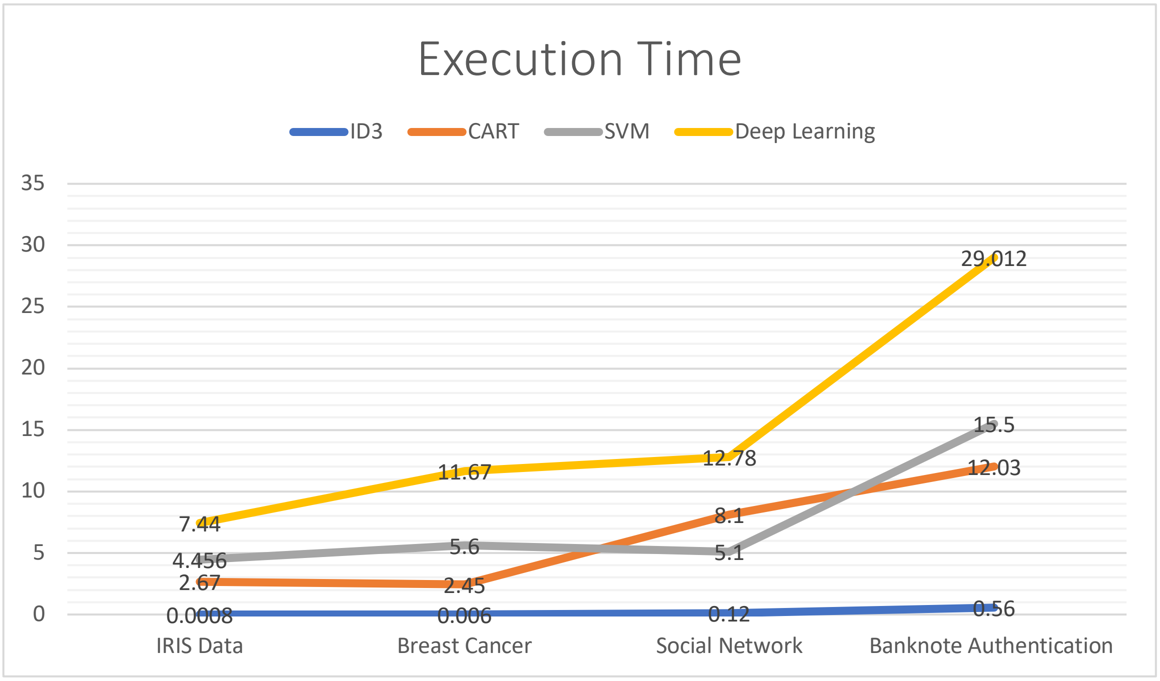

Figure 7: Execution Time

Execution Time – The above graph represents the comparison of execution time values for ID3, CART, SVM & Deep Learning(TensroFlow). We observe that, the execution time for algorithm increase as the size of the dataset increases.

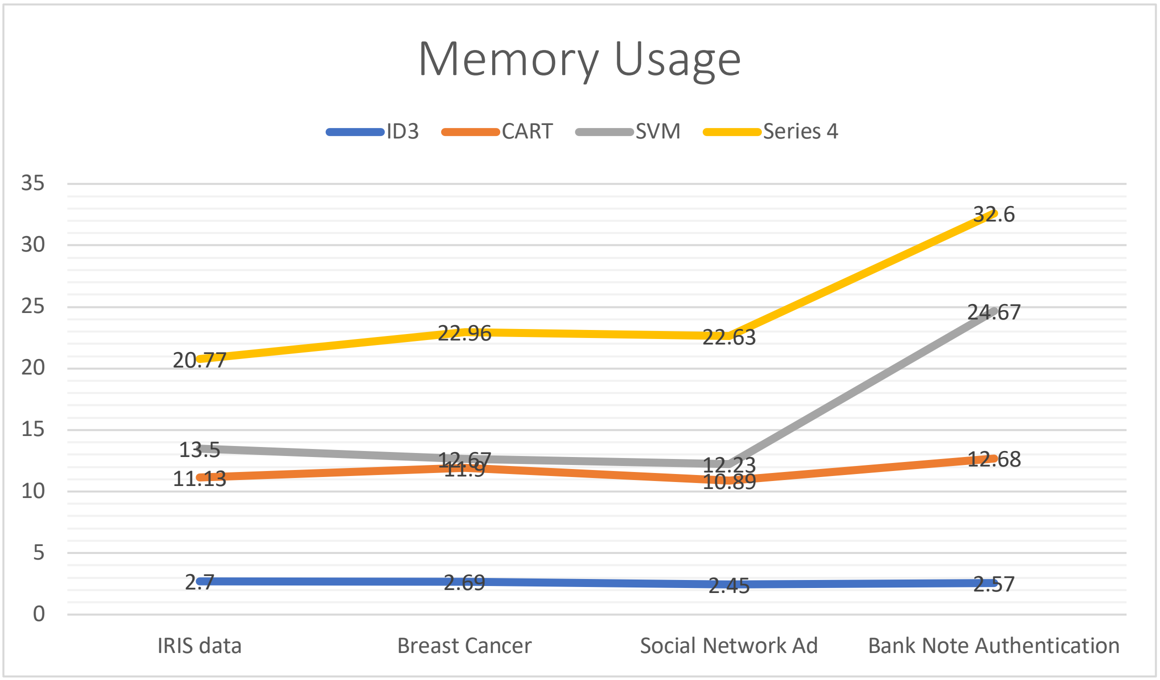

Figure 8: Memory Usage

Memory Usage – The above graph represents the comparison of the amount of memory used for ID3, CART, SVM & Deep Learning. We get an observation from the graph that ID3 < CART < SVM < TF, also as size of dataset increases, the memory used increases.

Conclusion

In this project, we have successfully implemented the Decision tree algorithm based on Information gain and entropy, and Gini index attribute selection measures and have produced an accuracy of about 94% for ID3 implementation and about 90% for CART Classification algorithm. We've also shown how the decision trees are prone to overfitting by plotting the data points and mentioned an improvement on our algorithm, tree pruning, that reduces overfitting. Further, we've compared our algorithm with Support Vector Machines (SVM) and TensorFlow deep learning model provided by google with respect to accuracy, execution time & memory used. We conclude that decision trees are simple, less cost, allow addition of new data & they help determine best, worst & expected values for a given scenario.

Bibliography

[1] Han, J., & Kamber, M. (2006). "Data Mining: Concepts and Techniques" (3rd ed.). Morgan Kaufmann Publishers.

[2] Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (2006, March). "From Data Mining to Knowledge Discovery in Databases". The Knowledge Engineering Review,Vol 21, No.1, pp. 1-24.

[3] Agarwal, R., Imieliński, T., & Swami, A. (1993, June). "Mining association rules between sets of items in large databases". ACM SIGMOD Record, Vol 22,No.2, pp. 207-216.

[4] Pujari, A. K. (2001). "Data Mining Techniques". Universites Press India Private Limited.

[5] Quinlan, J. R. (1987). "Simplifying decision trees". International Journal of Man-Machine Studies. **27** (3): 221. doi:10.1016/S0020-7373(87)80053-6

[6] R. Quinlan, "Learning efficient classification procedures", *Machine Learning: an artificial intelligence approach*, Michalski, Carbonell & Mitchell (eds.), Morgan Kaufmann, 1983, p. 463-482. doi:10.1007/978-3-662-12405-5_15

[7] Kamiński, B.; Jakubczyk, M.; Szufel, P. (2017). "A framework for sensitivity analysis of decision trees". *Central European Journal of Operations Research*. doi:10.1007/s10100-017-0479-6