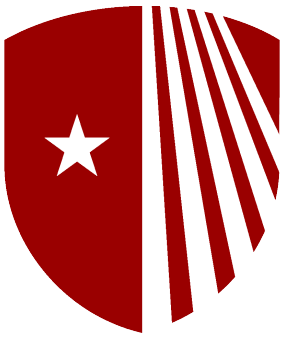
Decision Tree Induction Algorithm

ESE 589

Zahin Huq & Hualin Zheng

December 2019



Stony Brook University

Abstract

Decision Tree Induction is a classification method that generates learned decision trees from partitioned training data. It is a common method of classification because of its conceptual simplicity and speed, as well as its ability to handle large amounts of data. However, it can be limited by the specific data used. In this paper, we implement the Decision Tree Induction algorithm in order to analyze its performance and effectiveness. In addition, we experiment with three different attribute selection metrics to observe their effect on the Decision Tree Induction algorithm. Using benchmark examples, we can test the accuracy and precision of the classifier and experiment with factors such as percentage of data used for training and dataset types.

1 Introduction

Decision Tree Induction is a top-down recursion based algorithm that utilizes the "divide and conquer" method. The training set is recursively broken down into smaller units while the algorithm creates the tree. Classification arises when tuples belong to the same class or category, and the decision tree is developed in a corresponding manner. When traversing the created decision tree with test data, the goal is to accurately predict the category for each tuple. The algorithm requires a method of selecting the attribute order to develop the tree, in addition to utilizing different splitting criterion for different attributes. In this paper, we implement the Decision Tree Induction algorithm with three different attribute selection metrics.

The information gain, gain ratio, and gini index are the three attribute selection metrics to be tested. By comparing them to one another with several benchmark datasets, we can observe which metrics are better suited for different kinds of data. However, the algorithm still follows the same general process. The inputs are the partioned training data, the list of attributes, and the attribute selection criterion (information gain, gain ratio, or gini index). As stated before, the algorithm then develops a decision tree model based on the training data.

1.1 Algorithm

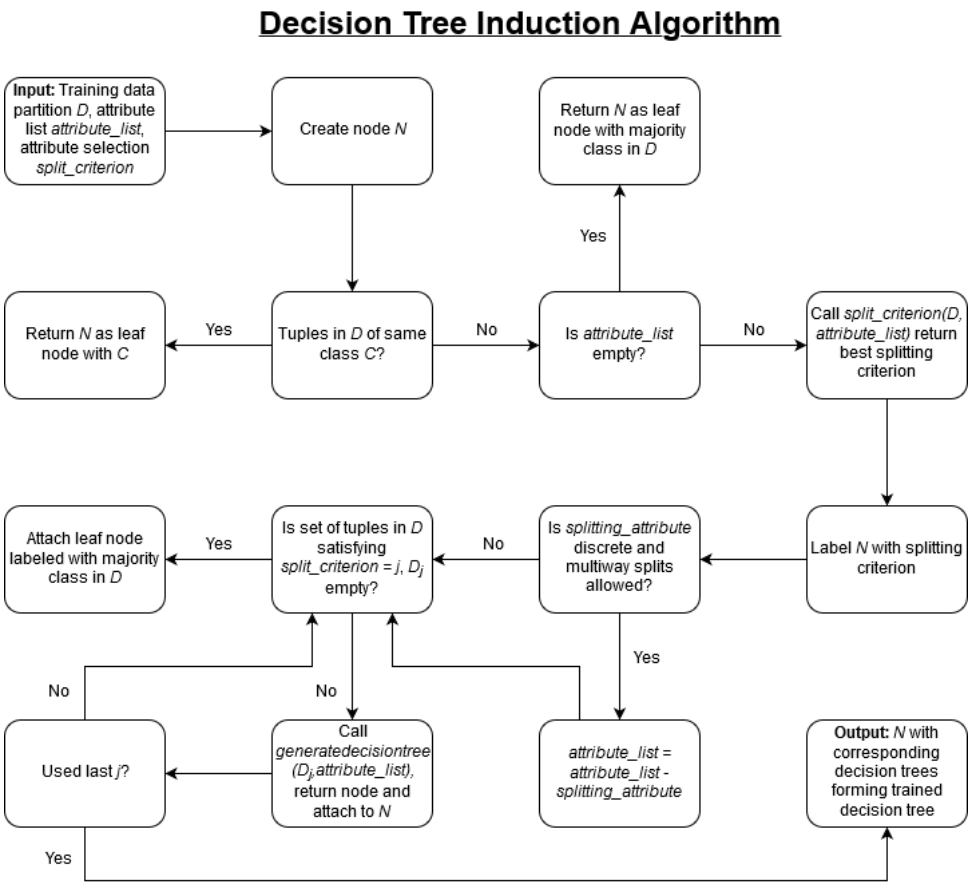


Figure 1: Flow chart demonstrating the process to generate the decision tree using the Decision Tree Induction algorithm.

The algorithm is a recursive process based on splitting the current attribute. The algorithm follows these basic steps:

1. Select training data and attribute selection metric to partition the data.
2. Partition tuples based on splitting criterion and generate subtrees for each partition.
3. Assign generated subtree to current node.
4. Repeat process until end of attributes and equivalent splitting criterion.

1.2 Attribute Selection Metric

The goal of the attribute selection criterion is to partition the training dataset where each split consists of tuples belonging to the same class. This is not always achievable, but the optimal splitting criterion creates partitions that closely resemble this condition.

Information gain is one such metric used. It calculates the average amount of information needed to classify a certain tuple within the training set before and after partitioning, and calculates the gain for each attribute. Information gain has an inherent bias based on tests with many tuples assigned to an attribute or class. Gain ratio builds on information gain to overcome the bias. It calculates the possible info generated when splitting based on partitions in the dataset, and then calculates the ratio using the information gain. The gini index considers binary splits on the partitions, where the gini index for the attribute is calculated using a ratio summation of the gini metric for both partitions. The gini index for the entire training set is then used to calculate the difference between it and the gini index for a certain attribute. Depending on the data, these metrics can be used as a blueprint for splitting the partition.

2 Software Design

The algorithm was implemented in Java. The Decision Tree Induction algorithm has 6 steps to get the final decision result:

1. Input a dataset for processing.
2. Select algorithm.
3. Calculate entropy, information gain, gain ratio of attributes.
4. Build the decision tree.
5. Make the prediction from test dataset.

The prediction is outputted in a separate text file, adding the predicted classes to the test data.

2.1 Organization

The data structure organization of our implementation is below:

Structures and Functions	Description	Inputs	Outputs
TreeNode	The basic structure to record tree node	N/A	N/A
Tree	The higher structure to building Tree	nodes	tree
createTree	Create Decision Tree	data, attributes	Decision tree
partitionTress	Create partition	data, attributes	partitions
ID3	Generate information gain based on ID3 algorithm	attributes, lablels	entropy, info gain
C4.5	Generate gain ratio based on C4.5 algorithm	attributes, lablels	entropy, gain ratio
CART	Generate gini index based on CART algorithm	attributes, target attributes, lablels	gini index
predict_infogain	make prediction based on infomatin gain and entropy	root, tuple	results file
predict_gini	make prediction based on gini index	root, partition, tuple	results file
fileReader	Read input datasets, user inputs, and write output data and resuls	dataset, input choices	train data, test data, results data, labels

Figure 2: Structures and Functions.

3 Implementation & Validation

In order to test whether our code outputs the desired results correctly (i.e. the decision tree, metric values, and CPU time), we first validate it using simple examples. The figures below are diagrams of the decision tree that is created using our java implementation.

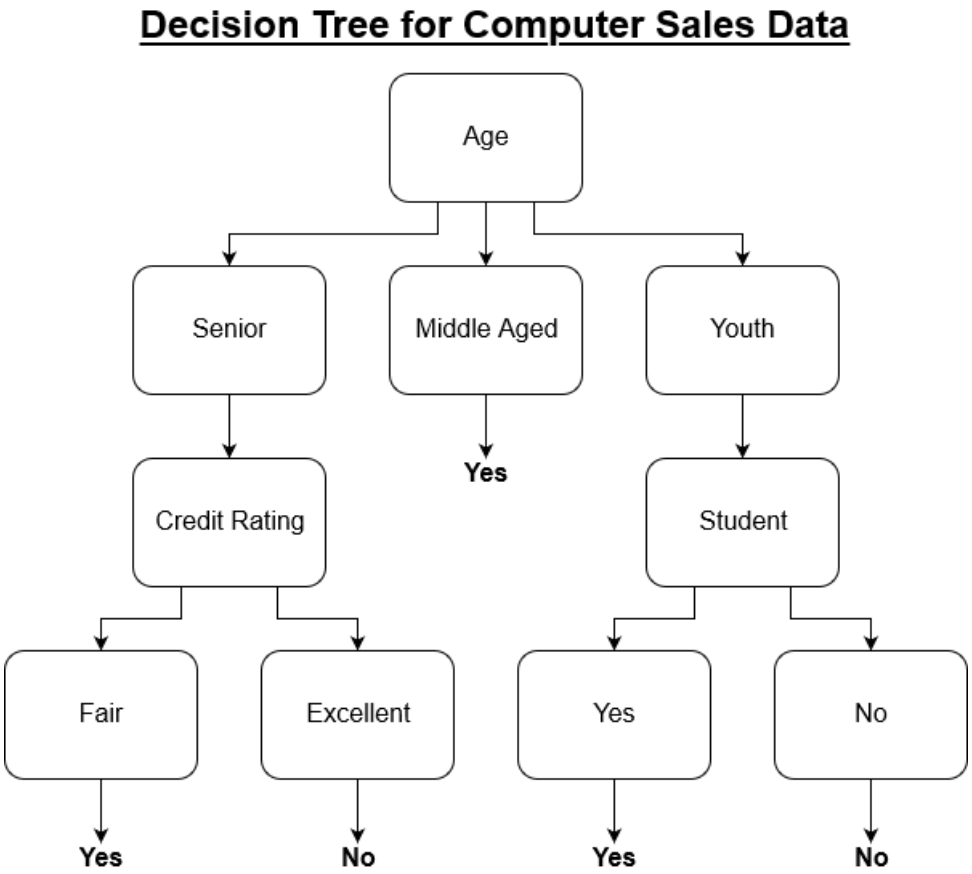


Figure 3: Diagram of the outputted decision tree for the computer sales sample data developed from the Decision Tree Induction algorithm.

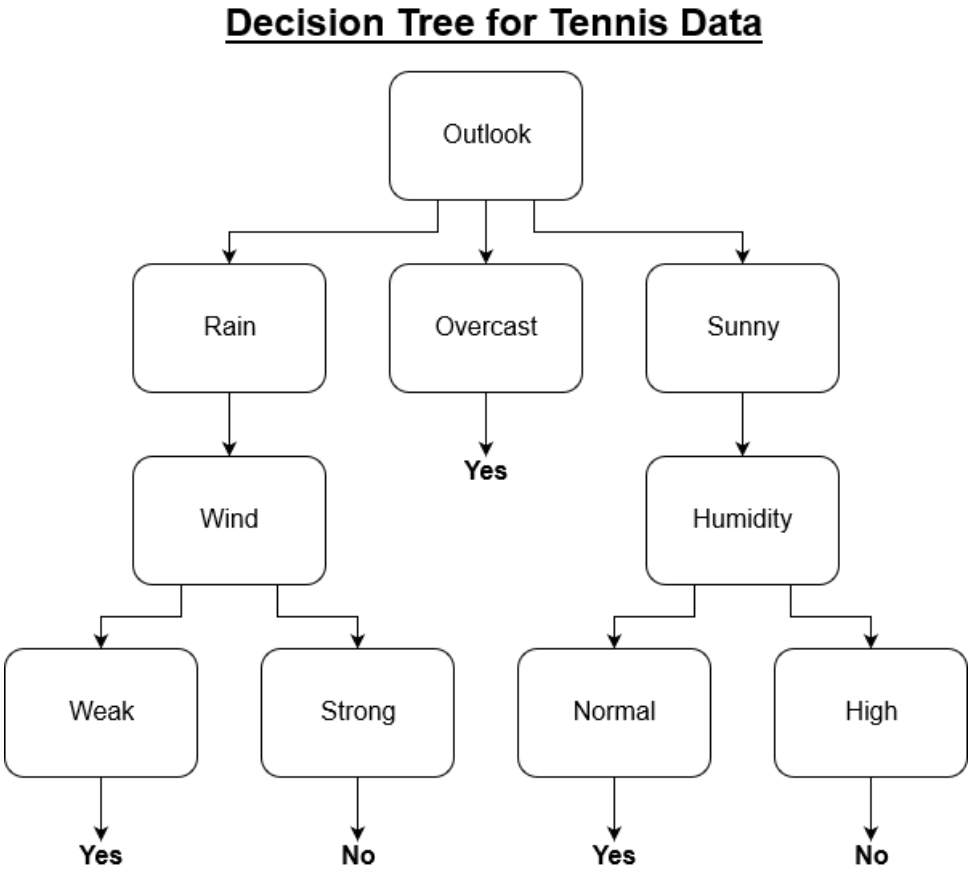


Figure 4: Diagram of the outputted decision tree for the tennis sample data developed from the Decision Tree Induction algorithm.

As expected, the CPU time outputted for the textbook example dataset is 0 ms because of its simplicity and size. Using the textbook example, which we use $\frac{15}{18}$ tuples for the training and $\frac{3}{18}$ for the test, the outputted decision tree accurately develops a tree that predicts the test data classifications. Although income is an attribute, it does not appear in the decision tree indicating that it is insignificant given the training data when predicting the likelihood of someone purchasing a computer. This shows that not all attributes will be included in the decision tree, and it depends on the ratio of the training partition to the whole dataset as well.

For the tennis dataset, the CPU time is instead 51 ms. This time, $\frac{15}{22}$ tuples were used for the training and $\frac{7}{22}$ for the testing. Our implementation outputs the metric values depending on the desired attribute selection metric. The figure below shows the information gains at different entropies corresponding to the different attributes.

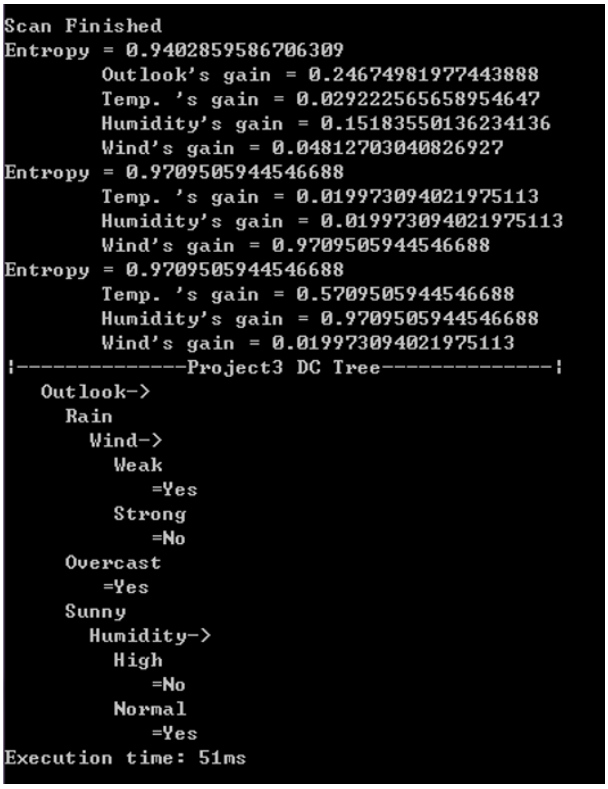


Figure 5: Output using ID3 for the tennis prediction decision tree.

4 Experiments & Discussion

To test the performance of the algorithm, we can measure the CPU time and accuracy. The control variables are the attribute selection metrics and the ratio of training data to test data, which is 60%, 70%, and 80%. We hypothesize that the accuracy of the classifier would increase with increasing training to test data ratios. This is because as the ratio increases, less tuples are available to traverse the generated decision tree and therefore has a lesser chance of tuples not satisfying the classifier. Because various benchmarks are used, we can observe which attribute selection criteria offer the most optimal CPU time to build a classifier.
We found that the gini index is optimal for small datasets because of the amount of CPU it requires. Below are figures demonstrating the difference in CPU time for two example datasets.

CPU Time for Caesarian Section Dataset

Training Dataset Percentage:	60%	70%	80%
ID3	40 ms	40 ms	50 ms
C4.5	38 ms	40 ms	28 ms
CART	1829 ms	7496 ms	6520 ms

Figure 6: CPU time for each attribute selection metric and training partition ratio for the Caesarean Section dataset. Attributes = 5 and Instances = 80.

CPU Time for Transfusion Dataset

Training Dataset Percentage:	60%	70%	80%
ID3	248 ms	168 ms	215 ms
C4.5	132 ms	116 ms	115 ms

Figure 7: CPU time for each attribute selection metric and training partition ratio for the Transfusion dataset. Attributes = 5 and Instances = 748.

First we notice that the gini index is not suitable for larger datasets due to the amount of memory usage. For the computer used in these specific experiments, the CART method either did not execute properly due to memory allocation errors or it had a relatively high CPU time as seen in the figure for the Caesarean Section dataset. In comparison, the C4.5 and ID3 methods produce similar CPU time for the Caesarean Section dataset but C4.5 is faster for the Transfusion dataset. Examining the accuracy of the output when compared to the test data would provide insight on the balancing the CPU time with precision.

Figure 8: Accuracy percentage for the Caesarean Section dataset.

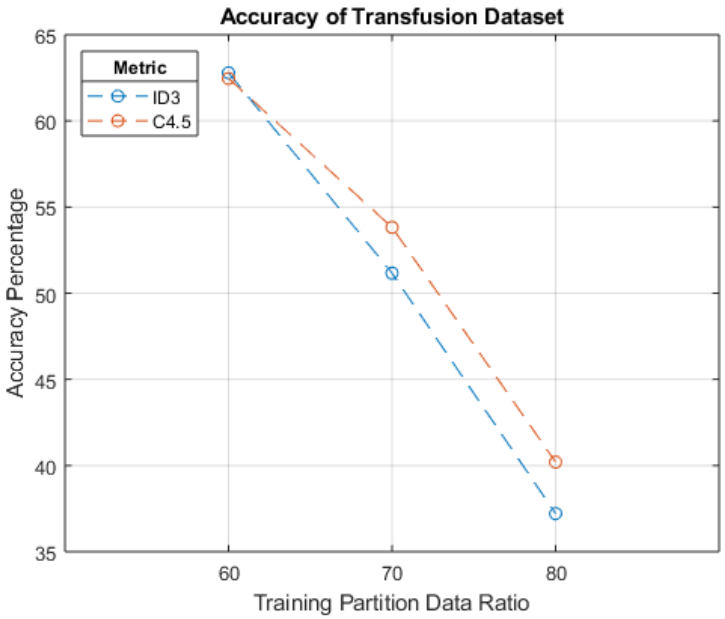


Figure 9: Accuracy percentage for the Transfusion dataset.

We observe two different trends. For the Caesarean Section dataset the accuracy increases with increasing partition percentage, while for the Transfusion dataset we observe decreasing acurracy with increasing partition percentage. In addition, the Caesarean Section dataset demonstrates how although the gini index requires more CPU to process, it has a significant larger accuracy. The Transfusion dataset shows similar accuracy, however it was noted that the gain ratio method requires less CPU to process showing that it is more effective.

We see that the nature of the attribute selection method and its benefits are data dependent. While this sample size is small, we need to test other benchmarks to develop an accurate conclusion, however it is apparent that data specific factors tend to dictate accuracy and CPU time.

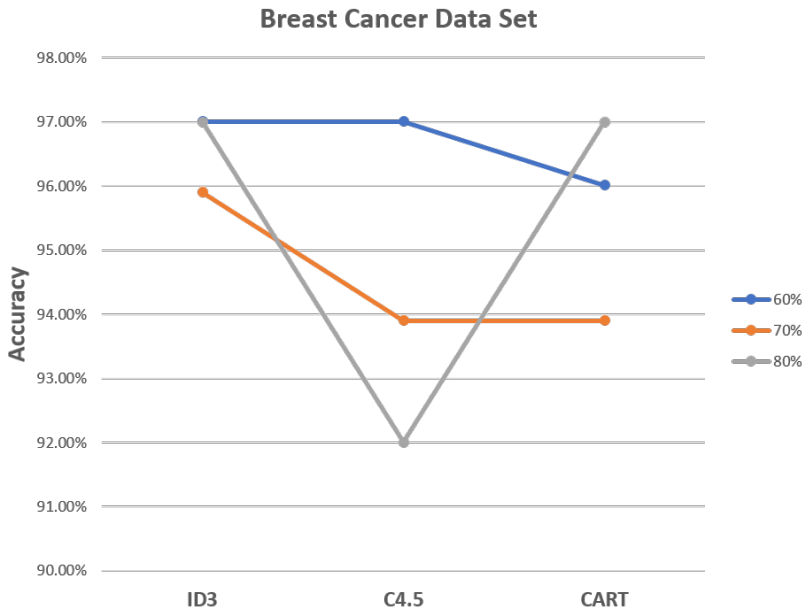


Figure 10: Accuracy percentage for the Breast Cancer dataset. Attributes = 10 and Instances = 515.

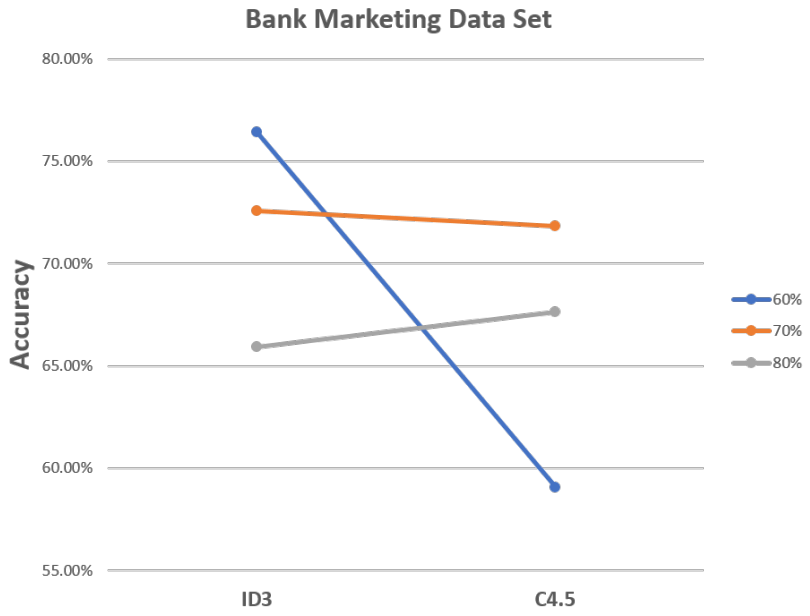


Figure 11: Accuracy percentage for the bank marketing dataset. Attributes = 17 and Instances = 45212.

The Breast Cancer dataset shows inconsistent results when compared to the Caesarean Section dataset. The CART method does not provide significantly greater accuracy, instead it decreases

from ID3 to C4.5 to CART. There is a greater difference between the training partition ratios in the Breast Cancer dataset. Greater accuracy occurs at 60%, and using the ID3 method. This is also the case for the Bank Marketing dataset, which is the largest of the ones tested, and shows that ID3 may be the best metric for large datasets and using 60% training percentage. However, as seen in the previous experiments, ID3 requires greater CPU time than C4.5 so it is possible the programmer may want to use a higher training percentage with C4.5 despite lesser accuracy.

5 Conclusion

The Decision Tree Induction algorithm is a conceptually simple method of developing a classifier. It generates a decision tree based on a training dataset and an attribute selection metric, where this metric allows fluidity in the algorithm design depending on the problem specifications. As a result, the programmer is able to customize the algorithm accordingly to develop a speedy yet accurate classifier. We determine that using the gini index requires a significant amount of CPU time, making it unsuitable for larger datasets. Using the gain ratio requires less CPU time than using the information gain, however for smaller datasets the accuracy is relatively similar whereas for larger datasets, the information gain provides more accurate results. As assumed, data uniqueness affects which attribute gain metric is the most optimal and to proceed with the algorithm accordingly.

It is possible that the disparity in the accuracy is a result of the nature of the data itself. If attributes have simple options, for example "low", "medium", and "high", then there are less possibilities and larger amounts of combinations within the tuples. This makes it easier than with discrete values over a wide range to develop nodes and a logic based decision tree. However, this can be remedied by preprocessing data techniques such as binning and correlative analysis. Another possible limitation of the Decision Tree Induction algorithm is if the data is sorted in classification order. The training dataset would then contain all or a majority of a certain class, and not wholly represent the dataset. Not only does data uniqueness affect which attribute selection criterion is optimal, but it affects the accuracy and computation time independent of the attribute selection criterion.

For these reasons we propose that the Decision Tree Induction algorithm is best suited for smaller datasets with less standard deviation within attributes. For large datasets with discrete values over a wide range or a large amount of varying tuples, other classifiers may provide more accurate results. However, this algorithm does have advantages in terms of speed.