Comparative Study of Decision Tree Algorithms on Weather Forecast

Forum Atul Gala
fgala@ufl.edu

Aarthi Kashikar
aarthikashikar@ufl.edu

Radhika Agrawal
agrawalr@ufl.edu

Abstract

Weather forecast is a challenging task, as it is dependent on various features that need to be considered for classification. There are numerous algorithms that solve this classification problem. Decision trees turn out to be an apt solution, as they can be easily visualized with each non-leaf node being a decisive question. In this project, we will do a comparative study of the implementation of three popular decision tree algorithms(ID3, C4.5, CART) based on their accuracy and time complexity, for facilitating the weather forecast.

1 Introduction

For a supervised learning problem, predictions are dependent on the features of the data. Decision Trees are popular machine learning techniques, that are efficient and simplified methods for supervised learning. Decision Tree algorithm makes calculated choices of the features. There are two ways in which this algorithm can be performed, it is dependent on the hierarchy or based on the rules(if,else). This algorithm represents each choice based on features as a branch of a flowchart, and every decision is represented as a node of the flowchart. All these decisions together helps in predictions with high accuracy rate. In brief, decision tree models are tools for comprehensive analysis of the outcome of a decision. It helps to understand whether these conclusions made are uncertain or definite, and is there a possibility to make new decisions from the conclusion.

Based on a research it was concluded that Decision tree also outperforms some of the algorithms like K-NN and Bayesian in error rate and accuracy.Decision tree has less error rate and is easier compared to other classification algorithms. It is also easier to visually understand for humans.

This principle of the model is used to simplify the tedious task, to classify data and makes it simple to understand, interpret, and can be visualized easily. Other reasons for popularity of this algorithm is the less data preparation required, compared to other techniques, which require normalizing the data, removing blanks in the data, and creating some extra variables. Decision trees are efficient in handling both the categorical and numerical value data, by taking decisions at every level. The decisions made in the Decision Tree model are dependent upon the principle of calculating entropy which is the dissimilarity between the data, and then segregating the data based on it.

The study of Decision Trees helps to understand various methods to calculate these dissimilarities, such as entropy, Information Gain Ratio, Gini score. Information Gain Ratio is the inverse of entropy and Gini score tells how good a split is by indicating how mixed the classes are in the two groups created by the split. Based on these different ways for dissimilarity calculation, there are different decision tree algorithms such as ID3(Iterative Dichotomiser 3), C4.5(extension of ID3) and CART(Classification And Regression Tree).

To understand these different decision tree algorithms better, we decided to do a comparative study between these models, using a Weather data-set that we will use to predict the occurrence of rain. This data-set contains 10 years of data of Australian weather in terns of temperature, wind speed, rainfall, humidity, pressure, etc. and prediction of whether it would rain the next day. We implemented these ID3, C4.5 and CART algorithms on the above mentioned data-set in python and compared their accuracy and time complexity.

2 Problem Statement

Decision Tree is very popular in the fields of data mining, machine learning and statistics for its ability to handle different data and simplicity to understand its human-like thinking approach. Our aim is to study decision trees and implement three different decision tree classification algorithms ID3, C4.5 and CART in python using Australian weather data-set. This data-set consists of data like temperature, wind-speed, humidity, pressure,etc for different cities in Australia for the span of 10 years and is used to predict whether it will rain tomorrow or not. These three supervised learning algorithms will be implemented to predict occurrence of rain and compare their performance.

3 Algorithm

Decision tree learning is one of the popular and commonly used predictive modeling approaches used in machine learning. A decision tree is used for prediction using observations about a subject known as features to conclude about the subject's target value. So, the purpose of decision trees is to generate a model that predicts the target variable based on several input features.

A decision tree is a tree in which each internal node is labeled with an input feature. These feature values are represented in the branches of the decision tree and its leaves represent the target values. Each branch from input feature node is labeled with some possible value of the target or it points towards a subordinate decision node for an another input feature. The tree is build in a top-down approach by selecting a feature that best splits the data-set. The data-set is said to be classified using a decision tree when each leaf of the tree is associated with a class or a probability distribution. Decision trees are more commonly used prediction model as they often resemble the human thinking and are very simple to understand.

The decision tree is built by splitting the data-set using a set of rules based on input features. The splitting is recursively carried on and is stopped only when the subset at a node has the same value as that of the target variable or when splitting further does not change the predictions.

The decision tree can either be a regression tree or a classification tree. The decision tree is a Classification tree when the predicted value is the class of the data. In this case, the leaf values are discrete set of values, called as labels. The decision tree is a regression tree when the predicted value can be considered as a real number. In that case, the leaf values are continuous values (real numbers). In our study, we are focussing on the decision tree for classification.

A decision tree classifier is a binary tree where predictions are made by traversing the tree from root to the leaf through each node. We go left if a feature is less than a threshold or else we go right. We will be predicting if it is going to rain tomorrow or not. So the leaf values will be in terms of 0 and 1, Where class '0' indicates no rain and class '1' indicates rain.

There are different algorithms used for constructing decision trees like ID3 (Iterative Dichotomiser 3), C4. 5 (successor of ID3), CART (Classification And Regression Tree), CHAID (CHi-squared Automatic Interaction Detector) and MARS: extends decision trees to handle numerical data better. In our study we will be focusing on ID3, C4.5 and CART algorithms.

Algorithms used for constructing decision trees usually use the top-down approach, by selecting a feature at each step that best splits the data-set. This best split is decided by different metrics in different algorithms. These metrics are calculated to measure the homogeneity of the target variable. CART algorithm uses Gini Index as metric while ID3 algorithm uses Entropy function and Information Gain as metric.

In this study, the data-set used was obtained from Kaggle [11]. The weather data-set contains 142,193 daily weather observations from 49 weather stations across Australia over the period November 2007 to June 2017. The following are the features in the data-set: Date, Location, MinTemp(in degrees celsius), MaxTemp(in degrees celsius), Rainfall (in mm), WindGustSpeed (in km/h) , WindSpeed9am (in km/hr), WindSpeed3pm (in km/hr), Humidity9am(percent), Humidity3pm(percent), Pressure9am

(hpa), Pressure3pm (hpa), Temp9am (in degrees Celsius), Temp3pm (in degrees Celsius), RainToday (Boolean: 1/0), RISKMM(mm), RainTomorrow (target variable).

All the entries with 'NA' value were dropped for better prediction, resulting in the data-set being reduced to 58,079 observations over 26 unique Locations as opposed to 49 orginally. The analysis and results that follow is for these 26 locations using the truncated data-set. After cleaning the weather data-set, it was then randomly split in 7:3 ratio for training and testing purposes. The larger data-set was used for training the models while the smaller part was used for testing.

3.1 ID3

Iterative Dichotomiser 3 or ID3 algorithm is invented by J.Ross Quinlan in 1975. The algorithm follows the Occam's razor principle where smallest possible decsion tree is attempted to be created. It generates a multiway decision tree from a given data set by using greedy search top to down where each attribute is tested at every node of the tree. The resulting tree is used to classify future samples.

The algorithm depends mainly on the entropy and information gain of every attribute in the data set. The data is split at every node to select the best spilt. The best split is decided based on the best attribute which means the attribute that produces the minimum entropy or maximum information gain. Decision tree node is built containing that attribute. ID3 requires intense pre-processing in order to overcome some of the shortcomings of the algorithm.

The complexity of the tree depends on the order in which the attributes are chosen. Information theory is used to determine the attribute that gives the most information about the data-set.

3.1.1 Entropy

The amount of disorder or uncertainty is measured using Entropy. data-set that provides least information is due to the disorder or uncertainty. Entropy defines the certainity to guess the instance. The entropy in case of date set with different class for each instance would be very high since the level of uncertainty is high. Therefore, it is a formula to calculate the homogeneity of a sample.

Lets say we are given a probability distribution P=($p_1, p_2 ... p_n$) the formula for the entropy is as follows:

$$E(P) = -\sum_{i=1}^{n} p_i \times log_2 p_i$$

3.1.2 Information Gain

Information gain can be said as the effective change in entropy after deciding on a particular attribute. The relative change in entropy for each attribute is measured by information gain. It is based on the decrease in entropy after a data-set is split on an attribute.

The formula for the entropy is as follows:

$$Gain = E(p) - \sum_{i=1}^{n} p_i \times E(p_i)$$

3.1.3 Algorithm

The algorithm follows the below steps:

1. The entropy of the total data-set is calculated.
2. The data-set is then split on different attributes. The entropy for each branch is calculated.
3. Then it is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split. The result is the Information Gain, or decrease in entropy.
4. The attribute that yields the largest IG is chosen for the decision node.
5. Repeat till we get the tree.

For the implementation of ID3 algorithm on the dataset I used python as the language. After the pre-processing of the data the data is converted to dataframe. The data is split into training and testing data in 8:2 ratio. First the entropy of the attributes is calculated except the 'RainTomorrow' attribute that is the attribute that needs to be predicted.Then information gain is calculated using the entropies of all the attributes. This showed that 'RISK_MM' is the attribute with maximum information gain of 0.704. Hence, this is taken as the attribute to split the data. Then the training data is sent to ID3 algorithm to train the model and build the tree using the entropies and information gain at each node. This created a multiway tree with a depth of 1. This produced an accuracy of 99.84

3.1.4 Advantages

1. Understandable prediction rules are created from the training data.
2. Builds the fastest tree.
3. Builds a short tree.
4. Only need to test enough attributes until all data is classified.
5. Finding leaf nodes enables test data to be pruned, reducing number of tests. Whole data-set is searched to create tree.

3.1.5 Disadvantages

ID3 may not predict correctly when there is too much noise or when data is over-fitted or over-classified when a small sample is tested. Only one attribute at a time is tested for making a decision. Classifying continuous data may be computationally expensive as many trees must be generated to see where to break continuum.

3.2 C4.5

C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan. The C4.5 generated decision tree is used for classification, thus, C4.5 can be referred as the statistical classifier. This algorithm is quite similar to the ID3 algorithm, and improves ID3 behaviors, such as using continuous data, unknown values, and attributes with different weights. For the C4.5 we are going to find the dissimilarity between the data, based on the information gain ratio, which is an extension of the entropy gain, which was used in the ID3 algorithm. We can calculate the key component of C4.5, information gain based on the spread of the data, and the Entropy Gain. The complexity of the C4.5 is dependent on the size of the training data, and the number of the features present in the data.

3.2.1 Information Gain Ratio

Information gain is the value which is the principle for branching the decision tree. The information gain is the ratio of the Entropy gain and the Split Information by calculating the spread of the dataset. The Entropy gain can be calculated by finding the difference between the entropy calculated and the summation of the probability of the Entropy. The split information can be calculated by summing up the probability of occurrence of the data and logarithm of the data.

3.2.2 Algorithm

The algorithm is as follows:

1. Start, we need to check what are the best cases above all the decisions made.

2. Calculating the normalised information gain ratio for each feature, when splitting on the decision on that particular feature.

3. If suppose a particular node, is the one which has the highest information gain ratio.

4. Dependent on this particular node, the decision node will be formed.

5. Recur this process on the remaining list, with that particular node as the root node for branching, and we'll be adding the other nodes as the children of the parent node.

6. Setting the labels for all the decision nodes.

The algorithm is based on the Information Gain Ratio, which has the concept that if we know more about a topic, then the chances to gain new information about the topic is very less. It means that if we know a event is probable to happen, then there is no information, and it provides very less information, when it actually happens. Information gain ratio is calculated by taking ratio of the observations to the total number of observations. When branching the tree, if we calculate the entropy of the next decision and if it is lesser than the previous decision and if this value is comparatively lesser than the all the decisions, then this is feasible to be the root.

3.2.3 Advantages

1. The algorithm allows to overcome over-fitting.
2. It works with continuous and discrete data.
3. This algorithm also works, when there are unknown or missing values.
4. It handles features which have different costs.
5. The algorithm supports the single pass pruning process.

3.2.4 Disadvantages

1. The speed for other algorithms is much faster than this algorithm, C4.5.
2. The C4.5 algorithm is not memory efficient, and memory usage can be improved.
3. The decision trees for C4.5 are comparatively larger than for other algorithms.
4. Weighting of different cases cannot be altered for the C4.5 algorithm.
5.The accuracy of the predictions can be improved by boosting.

3.3 CART

CART Algorithm stands for classification and regression trees. It was introduced by Breiman in 1984. It builds both classifications and regression trees. It is a recursive training algorithm.

The CART is constructed by binary splitting of attributes. The recursion stops when the maximum depth is reached or when the split does not result in children purer than their parent. CART is a decision tree classifier with high classification and prediction accuracy. The important part of the CART Algorithm is to find the optimal feature that minimizes the gini index value. In the process of finding that optimal feature, we try all the possible splits and compute the resulting Gini impurities.

3.3.1 Gini Index

The metrics used to construct CART is called gini index. Gini index is also called as gini impurity. Gini impurity is the concept used to determine the homogeneity of a node. Gini index helps in determining the best split point for the data-set. A gini score tells us how good is the split by determining how mixed are the classes in the two groups created by the split. If the gini score is 0 that means the split is perfect. The worst split is when the gini score is 1

At every node,further splitting point is identified by calculating gini index. The Gini impurity is calculated as

$$G = 1 - \sum_{k=1}^{n} {p_k}^2$$

where, n is the number of training samples, k is the number of classes and p[k] is the fraction of samples belonging to class k.

However, this is not feasible and is very slow as we would need to look at all samples and partition them into left and right. Mathematically, there will be n splits with each split having $O(n)$ operations , making the overall cost $O(n^2)$.

So, we look for a faster approach. The more feasible function is called the best-split function. Here we iterate through the sorted feature values as possible thresholds, keep track of the number of samples per class on the left and on the right, and increment/decrement them by 1 after each threshold. This computes Gini index in constant time.

If m is the size of the node and m[k] are the number of samples of class k in the node, then

$$G = 1 - \sum_{k=1}^{n} {p_k}^2 = 1 - \sum_{k=1}^{n} \left(\frac{m_k}{m}\right)^2$$

and since after seeing the i_th threshold there are i elements on the left and m–i on the right,

$$G_i^{left} = 1 - \sum_{k=1}^{n} \left(\frac{m_k^{left}}{i}\right)^2$$

and

$$G_i^{right} = 1 - \sum_{k=1}^{n} \left(\frac{m_k^{right}}{m-i}\right)^2$$

The resulting Gini:

$$G_i = \frac{i}{m} G_i^{left} + \frac{m-i}{m} G_i^{right}$$

3.3.2 Algorithm

1. start generating the tree with a single root.

2.repeat 2.1-2.4 until all the nodes are homogeneous

2.1 select a non-homogeneous node X such that it's gini index is not 1.

2.2 attach left and right children nodes to the node X.

2.3 for all covariates find the threshold that minimizes Gini index value.

2.4 find the feature that minimizes Gini index value and set this feature to node X.

3. set the labels to all nodes.

3.3.3 Advantages

1. CART can handle both numerical and categorical data.
2. CART algorithm can identify the most significant features and disregard the insignificant features.
3. CART decision trees are not affected by the nonlinear relationships among the features.
4. CART can handle outliers.

3.3.4 Disadvantages

1. CART algorithm splits only by one feature.
2. CART decision tree is unstable in nature. Even a small change in the data-set can cause variance in tree complexity and splitting features.

4 Experimental Comparison

Though ID3, C4.5 and CART are solving the same decison tree learning, there are some significant differences among them which reflect in the procedure, performance and accuracy. ID3 and C4.5 use entropy and info gain to measure the attributes whereas CART calculates Gini index. ID3 and C4.5 does a top-down decision tree construction with the attributes in a greedy approach, CART construct a binary decision tree. ID3 and C4.5 performs pre-pruning using a single pass algorithm and CART does post pruning based on cost-complexity measure. ID3 can handle only categorical values whereas C4.5 and CART can handle both categorical and numerical values. ID3 does not handle missing data from the data-set, there needs heavy preprocessing to be performed whereas C4.5 and CART can handle missing data.

C4.5 is an improvement over ID3 where C4.5 can handle both continuous and discrete attributes. This is possible because C4.5 creates a threshold and the split is made based on the attribute values above and below the threshold value. C4.5 can handle missing data whereas ID3 can not. ID3 needs preprocessing where rows with missing data have to be dropped but in case of C4.5 the attribute values can be marked with a question mark and those values are not used. ID3 does not have a post pruning mechanism whereas C4.5 takes care of pruning after tree creation. In C4.5 branches that do not help are removed and replaced with leaf nodes after the tree is created.

CART is a parallel discovery of ID3 around the same time. CART has lower probability of misclassification when compared to the other two algorithms. CART looks for surrogate tests that approximate the outcomes when the tested attribute has an unknown value, but C4.5 apportions the case probabilistically among the outcomes.

After implementing these three decision tree algorithms on our data-set, we found that CART algorithm predicted rain most accurately with the perfect accuracy of 1.0, ID3 algorithm had the accuracy of 0.99 while the C4.5 algorithm performed with the accuracy of 0.83 . We implemented the same code on two different sized data-sets from the same Australian Weather data-set. We found that the accuracy did not alter with size of the data-set. However, the time complexity varied drastically with the size of the data-set. The models predicted very quickly on smaller data-set , which was expected. But they took several hours to predict for the entire data-set.

After analyzing the prediction for this weather forecast problem, we see that the accuracy of the predicted data is much higher and the cost of using the decision tree is logarithmic to the number of training data-points.

Conclusion

Classification based on Decision tree has been a successful operation. As we see high accuracy rates for all the three methods of the Decision Tree Algorithm. The difference between the true value of the next day's arrival of rain, and the prediction is quite minimal. For the classification, we see that after analysing the results generated we see that CART is the most accurate method, followed by the ID3, and the least accurate is the C4.5 algorithm. The reason for different accuracy, is the different components for branching the decision tree such as the Entropy gain, the information gain ratio and the Gini Ratio. Each method as analysed has some pros and cons, but we learnt that the most feasible is the CART, as it overcomes the limitations of the C4.5 and ID3 algorithms. We also observed that the difference between the time taken to generate the tree and make predictions on a small data set and a large dataset is huge. Since the time taken on a large dataset is very huge for all the three algorithms with a small change in the accuracy. The further scope for this experiment can be to work on different datasets to make broader conclusion on the performance of the algorithms. Also, the ways to reduce the time taken on a large dataset can be studied. There are other decision tree algorithms that work on large datasets like SPRINT and SLIQ. The improvements that are made on these algorithms could be studied.

References

[1] Badr HSSINA, Abdelkarim MERBOUHA, Hanane EZZIKOURI & Mohammed ERRITAL.(2014) *A comparative study of decision tree ID3 and C4.5*. (IJACSA) International Journal of Advanced Computer Science and Applications, Special Issue on Advances in Vehicular Ad Hoc Networking and Applications.

[2] Bhumika Gupta, Aditya Rawat, Akshay Jain, Arpit Arora & Naresh Dhami.(2017) *Analysis of Various Decision Tree Algorithms for Classification in Data Mining*. International Journal of Computer Applications (0975 – 8887) Volume 163 – No 8.

[3] Sayali D. Jadhav & H. P. Channe.(2013)*Comparative Study of K-NN, Naive Bayes and Decision Tree Classification Techniques*. International Journal of Science and Research (IJSR),ISSN (Online): 2319-7064.

[4] W. N. H. W. Mohamed, M. N. M. Salleh & A. H. Omar.(2012) *A comparative study of Reduced Error Pruning method in decision tree algorithms*. IEEE International Conference on Control System, Computing and Engineering, Penang, 2012, pp. 392-397.

[5] A. Navada, A. N. Ansari, S. Patil & B. A. Sonkamble.(2011) *Overview of use of decision tree algorithms in machine learning*. IEEE Control and System Graduate Research Colloquium, Shah Alam,pp. 37-42.

[6] Anuja Priyam, Abhijeet, Rahul Gupta, Anju Rathee & Saurabh Srivastava.(2013) *Comparative Analysis of Decision Tree Classification Algorithms*. International Journal of Current Engineering and Technology, ISSN 2277 - 4106, Vol.3, No.2.

[7] Song YY & Lu Y.(2015)*Decision tree methods: applications for classification and prediction*.Shanghai Arch Psychiatry(2):130–135. doi:10.11919/j.issn.1002-0829.215044.

[8] Hemlata Chahal.(2013) *ID3 Modification and Implementation in Data Mining*. International Journal of Computer Applications (0975-8887), Volume 80-No7.

[9] Murthy, S.K.(1998) *Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey*.Data Mining and Knowledge Discovery-2,345–389.

[10] Songul Cinaroglu,(2016) *Comparison of Performance of Decision Tree Algorithms and Random Forest: An Application on OECD Countries Health Expenditures*. International Journal of Computer Applications (0975 – 8887) Volume 138 – No.1.

[11] Kaggle, data-set: https://www.kaggle.com/jsphyg/ weather-data-set-rattle-package/home

[12](2019)*Machine Learning with Python: Decision Trees in Python https://www.python-course.eu/Decision*$_{T}rees.php$

[13]$ValdecyPereira$, (2019)$https://github.com/Valdecy/C4.5$

[14]$JasonBrownlee$, (2019)$HowToImplementTheDecisionTreeAlgorithmFromScratchInPython.https://machinelearningmastery.com.$

ID3 Implementation

```
#Import packages
import pandas as pd
import numpy as np
from pprint import pprint
from sklearn import metrics
eps = np.finfo(float).eps
from numpy import log2 as log


#load dataset
df = pd.read_csv('Weathersmall6.csv')
df=df.drop('Location',axis=1)
df=df.drop('Date',axis=1)
#df=df.drop('RISK_MM',axis=1)

columns = df.columns.values.tolist()
dataset = df.dropna()

def train_test_split(dataset):
    train_data = dataset.iloc[80:].reset_index(drop=True)#We drop the index respectively relabel the index
    #starting form 0, because we do not want to run into errors regarding the row labels / indexes
    test_data = dataset.iloc[:80].reset_index(drop=True)
    return train_data,test_data

train_data = train_test_split(dataset)[0]
test_data = train_test_split(dataset)[1]

ent = 0
values = df.RainTomorrow.unique()
for value in values:
    fraction = df.RainTomorrow.value_counts()[value] / len(df.RainTomorrow)
    ent += -fraction * np.log2(fraction)

def ent(data, attribute):
    target_variables = data.RainTomorrow.unique()
    variables = data[attribute].unique()

    entropy_attribute = 0
    for variable in variables:
        entropy_each_feature = 0
        for target_variable in target_variables:
            num = len(data[attribute][data[attribute] == variable][data.RainTomorrow == target_variable])  #
            den = len(data[attribute][data[attribute] == variable])
            fraction = num / (den + eps)  # pi
            entropy_each_feature += -fraction * log(
                fraction + eps)
        fraction2 = den / len(data)
        entropy_attribute += -fraction2 * entropy_each_feature

    return (abs(entropy_attribute))

a_entropy = {k:ent(train_data,k) for k in train_data.keys()[:-1]}
a_entropy

def ig(e_dataset,e_attr):
    return(e_dataset-e_attr)

IG = {k:ig(ent,a_entropy[k]) for k in a_entropy}
IG

def find_entropy(data):
    Class = data.keys()[-1]   #To make the code generic, changing target variable class name
    entropy = 0
```

```
    values = data[Class].unique()
    for value in values:
        fraction = data[Class].value_counts()[value]/len(data[Class])
        entropy += -fraction*np.log2(fraction)
    return entropy

def find_entropy_attribute(data,attribute):
    Class = data.keys()[-1]
    target_variables = data[Class].unique()
    variables = data[attribute].unique()
    entropy2 = 0
    for variable in variables:
        entropy = 0
        for target_variable in target_variables:
            num = len(data[attribute][data[attribute]==variable][data[Class] ==target_variable])
            den = len(data[attribute][data[attribute]==variable])
            fraction = num/(den+eps)
            entropy += -fraction*log(fraction+eps)
        fraction2 = den/len(data)
        entropy2 += -fraction2*entropy
    return abs(entropy2)

def find_winner(data):
    Entropy_att = []
    IG = []
    for key in columns[:-1]:
        IG.append(find_entropy(data)-find_entropy_attribute(data,key))
    return data.keys()[:-1][np.argmax(IG)] #columns[:-1][np.argmax(IG)]#


def ID3(data, originaldata, features, target_attribute_name="RainTomorrow", parent_node_class=None):
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]

    elif len(data) == 0:
        return np.unique(originaldata[target_attribute_name])[
            np.argmax(np.unique(originaldata[target_attribute_name], return_counts=True)[1])]
    elif len(features) == 0:
        return parent_node_class
    else:
        parent_node_class = np.unique(data[target_attribute_name])[
            np.argmax(np.unique(data[target_attribute_name], return_counts=True)[1])]
        best_feature = find_winner(data)
        tree = {best_feature: {}}

        features = [i for i in features if i != best_feature]

        for value in np.unique(data[best_feature]):
            value = value
            sub_data = data.where(data[best_feature] == value).dropna()

            subtree = ID3(sub_data, dataset, features, target_attribute_name, parent_node_class)

            tree[best_feature][value] = subtree

        return (tree)

def predict(query, tree, default=1):
    for key in list(query.keys()):
        if key in list(tree.keys()):
            try:
                result = tree[key][query[key]]
            except:
                return default
```

```
            result = tree[key][query[key]]
            if isinstance(result, dict):
                return predict(query, result)

            else:
                return result

def test(data, tree):
    queries = data.iloc[:, :-1].to_dict(orient="records")
    predict = pd.DataFrame(columns=["predict"])
    for i in range(len(data)):
        predict.loc[i, "predict"] = predict(queries[i], tree, 1.0)
    print('Accuracy : ', (np.sum(predict["predict"] == data.RainTomorrow) / len(data)) * 100, '%')

tree = ID3(train_data,train_data,train_data.columns[:-1])

pprint(tree)
test(test_data,tree)
```

```
{'RISK_MM': {0.0: 0.0,
             0.2: 0.0,
             0.4: 0.0,
             0.6: 0.0,
             0.8: 0.0,
             1.0: 0.0,
             1.2: 1.0,
             1.4: 1.0,
             2.2: 1.0,
             3.0: 1.0,
             3.6: 1.0,
             6.4: 1.0,
             10.6: 1.0,
             15.6: 1.0,
             16.8: 1.0}}
The prediction accuracy is:  99.84854823864111 %
```

C4.5 Implementation

```
import pandas as pd
import numpy  as np
from random import randint
from scipy import stats
from copy import deepcopy

def valid_numeric(string):
    for i in range(0, len(string)):
        if pd.isnull(string[i]) == False:
            try:
                float(string[i])
                return True
            except ValueError:
                return False

def valid_numeric_value(value):
    if pd.isnull(value) == False:
        try:
            float(value)
            return True
        except ValueError:
            return False


def prediction_dt_c45(model, Xdata):
    Xdata = Xdata.reset_index(drop=True)
    ydata = pd.DataFrame(index=range(0, Xdata.shape[0]), columns=["Prediction"])
    for j in range(0, ydata.shape[1]):
        if ydata.iloc[:,j].dropna().value_counts().index.isin([0,1]).all():
            for i in range(0, ydata.shape[0]):
                if ydata.iloc[i,j] == 0:
                    ydata.iloc[i,j] = "one"
                else:
                    ydata.iloc[i,j] = "zero"
    data  = pd.concat([ydata, Xdata], axis = 1)
    rule = []
     # Preprocessing - Boolean Values
    for j in range(0, data.shape[1]):
        if data.iloc[:,j].dtype == "bool":
            data.iloc[:,j] = data.iloc[:, j].astype(str)

    dt_model = deepcopy(model)

    for i in range(0, len(dt_model)):
        dt_model[i] = dt_model[i].replace("{", "")
        dt_model[i] = dt_model[i].replace("}", "")
        dt_model[i] = dt_model[i].replace(".", "")
        dt_model[i] = dt_model[i].replace("IF ", "")
        dt_model[i] = dt_model[i].replace("AND", "")
        dt_model[i] = dt_model[i].replace("THEN", "")
        dt_model[i] = dt_model[i].replace("=", "")
        dt_model[i] = dt_model[i].replace("<", "<=")

    for i in range(0, len(dt_model) -2):
        splited_rule = [x for x in dt_model[i].split(" ") if x]
        rule.append(splited_rule)

    for i in range(0, Xdata.shape[0]):
        for j in range(0, len(rule)):
            rule_confirmation = len(rule[j])/2 - 1
            rule_count = 0
            for k in range(0, len(rule[j]) - 2, 2):
                if valid_numeric_value(data[rule[j][k]][i]) == False:
```

```
                    if (data[rule[j][k]][i] in rule[j][k+1]):
                        rule_count = rule_count + 1
                        if (rule_count == rule_confirmation):
                            data.iloc[i,0] = rule[j][len(rule[j]) - 1]
                    else:
                        k = len(rule[j])
                elif valid_numeric_value(data[rule[j][k]][i]) == True:
                     if rule[j][k+1].find("<=") == 0:
                         if data[rule[j][k]][i] <= float(rule[j][k+1].replace("<=", "")):
                             rule_count = rule_count + 1
                             if (rule_count == rule_confirmation):
                                 data.iloc[i,0] = rule[j][len(rule[j]) - 1]
                         else:
                             k = len(rule[j])
                     elif rule[j][k+1].find(">") == 0:
                         if data[rule[j][k]][i] > float(rule[j][k+1].replace(">", "")):
                             rule_count = rule_count + 1
                             if (rule_count == rule_confirmation):
                                 data.iloc[i,0] = rule[j][len(rule[j]) - 1]
                         else:
                             k = len(rule[j])

    for i in range(0, Xdata.shape[0]):
        if pd.isnull(data.iloc[i,0]):
            data.iloc[i,0] = dt_model[len(dt_model)-1]

    return data


def info_gain_ratio(target, feature = [], uniques = []):
    entropy = 0
    denominator_1 = feature.count()
    data = pd.concat([pd.DataFrame(target.values.reshape((target.shape[0], 1))), feature], axis = 1)
    for entp in range(0, len(np.unique(target))):
        numerator_1 = data.iloc[:,0][(data.iloc[:,0] == np.unique(target)[entp])].count()
        if numerator_1 > 0:
            entropy = entropy - (numerator_1/denominator_1)* np.log2((numerator_1/denominator_1))
    info_gain = float(entropy)
    info_gain_r = 0
    intrinsic_v = 0
    for word in range(0, len(uniques)):
        denominator_2 = feature[(feature == uniques[word])].count()
        if denominator_2[0] > 0:
            intrinsic_v = intrinsic_v -
            (denominator_2/denominator_1)*
            np.log2((denominator_2/denominator_1))
        for lbl in range(0, len(np.unique(target))):
            numerator_2 = data.iloc[:,0][(data.iloc[:,0] ==
            np.unique(target)[lbl]) & (data.iloc[:,1]  ==
            uniques[word])].count()
            if numerator_2 > 0:
                info_gain = info_gain +
                (denominator_2/denominator_1)*(numerator_2/denominator_2)*
                np.log2((numerator_2/denominator_2))
    if intrinsic_v[0] > 0:
        info_gain_r = info_gain/intrinsic_v
    return float(info_gain_r)


def split_me(feature, split):
    result = pd.DataFrame(feature.values.reshape((feature.shape[0], 1)))
    for fill in range(0, len(feature)):
        result.iloc[fill,0] = feature.iloc[fill]
    lower = "<=" + str(split)
    upper = ">" + str(split)
```

```
    for convert in range(0, len(feature)):
        if float(feature.iloc[convert]) <= float(split):
            result.iloc[convert,0] = lower
        else:
            result.iloc[convert,0] = upper
    binary_split = []
    binary_split = [lower, upper]
    return result, binary_split


def dt_c45(Xdata, ydata, cat_missing = "none", num_missing = "none",
pre_pruning = "none", chi_lim = 0.1, min_lim = 5):
    name = ydata.name
    ydata = pd.DataFrame(ydata.values.reshape((ydata.shape[0], 1)))
    for j in range(0, ydata.shape[1]):
        if ydata.iloc[:,j].dropna().value_counts().index.isin([0,1]).all():
            for i in range(0, ydata.shape[0]):
               if ydata.iloc[i,j] == 0:
                   ydata.iloc[i,j] = "zero"
               else:
                   ydata.iloc[i,j] = "one"
    dataset = pd.concat([ydata, Xdata], axis = 1)

    for j in range(0, dataset.shape[1]):
        if dataset.iloc[:,j].dtype == "bool":
            dataset.iloc[:,j] = dataset.iloc[:, j].astype(str)

    if cat_missing != "none":
        for j in range(1, dataset.shape[1]):
            if valid_numeric(dataset.iloc[:, j]) == False:
                for i in range(0, dataset.shape[0]):
                    if pd.isnull(dataset.iloc[i,j]) == True:
                        if cat_missing == "missing":
                            dataset.iloc[i,j] = "Unknow"
                        elif cat_missing == "most":
                            dataset.iloc[i,j] =
                            dataset.iloc[:,j].value_counts().idxmax()
                        elif cat_missing == "remove":
                            dataset = dataset.drop(dataset.index[i], axis = 0)
                        elif cat_missing == "probability":
                            while pd.isnull(dataset.iloc[i,j]) == True:
                                dataset.iloc[i,j] = dataset.iloc[randint(0, dataset.shape[0] - 1), j]
    elif num_missing != "none":
            if valid_numeric(dataset.iloc[:, j]) == True:
                for i in range(0, dataset.shape[0]):
                    if pd.isnull(dataset.iloc[i,j]) == True:
                        if num_missing == "mean":
                            dataset.iloc[i,j] = dataset.iloc[:,j].mean()
                        elif num_missing == "median":
                            dataset.iloc[i,j] = dataset.iloc[:,j].median()
                        elif num_missing == "most":
                            dataset.iloc[i,j] = dataset.iloc[:,j].value_counts().idxmax()
                        elif cat_missing == "remove":
                            dataset = dataset.drop(dataset.index[i], axis = 0)
                        elif num_missing == "probability":
                            while pd.isnull(dataset.iloc[i,j]) == True:
                                dataset.iloc[i,j] = dataset.iloc[randint(0, dataset.shape[0] - 1), j]
    unique = []
    uniqueWords = []
    for j in range(0, dataset.shape[1]):
        for i in range(0, dataset.shape[0]):
            token = dataset.iloc[i, j]
            if not token in unique:
                unique.append(token)
        uniqueWords.append(unique)
```

```
    unique = []
label = np.array(uniqueWords[0])
label = label.reshape(1, len(uniqueWords[0]))
i = 0
impurity = 0
branch = [None]*1
branch[0] = dataset
gain_ratio = np.empty([1, branch[i].shape[1]])
lower = "0"
root_index = 0
rule = [None]*1
rule[0] = "IF "
skip_update = False
stop = 2
upper = "1"


while (i < stop):
    impurity = np.amax(gain_ratio)
    gain_ratio.fill(0)
    for element in range(1, branch[i].shape[1]):
        if len(branch[i]) == 0:
            skip_update = True
            break
        if len(np.unique(branch[i][0])) == 1 or len(branch[i]) == 1:
             if "." not in rule[i]:
                 rule[i] = rule[i] + " THEN " + name + " = " +
                 branch[i].iloc[0, 0] + "."
                 rule[i] = rule[i].replace(" AND  THEN ", " THEN ")
             skip_update = True
             break
        if i > 0 and valid_numeric(dataset.iloc[:, element]) == False and
        pre_pruning == "chi_2" and chi_squared_test(branch[i].iloc[:, 0],
        branch[i].iloc[:, element]) > chi_lim:
             if "." not in rule[i]:
                 rule[i] = rule[i] + " THEN " + name + " = " +
                 branch[i].agg(lambda x:x.value_counts().index[0])[0] + "."
                 rule[i] = rule[i].replace(" AND  THEN ", " THEN ")
             skip_update = True
             continue
        if valid_numeric(dataset.iloc[:, element]) == True:
            gain_ratio[0, element] = 0.0
            value = np.sort(branch[i].iloc[:, element].unique())
            skip_update = False
            if branch[i][(branch[i].iloc[:, element] ==
            value[0])].count()[0] > 1:
                start = 0
                finish = len(branch[i].iloc[:, element].unique()) - 2
            else:
                start = 1
                finish = len(branch[i].iloc[:, element].unique()) - 2
            if len(branch[i]) == 2 or len(value) == 1 or len(value) == 2:
                start = 0
                finish = 1
            if len(value) == 3:
                start = 0
                finish = 2
            for bin_split in range(start, finish):
                bin_sample = split_me(feature = branch[i].iloc[:, element],
                split = value[bin_split])
                if i > 0 and pre_pruning == "chi_2" and
                chi_squared_test(branch[i].iloc[:, 0], bin_sample[0]) >
                chi_lim:
                    if "." not in rule[i]:
                         rule[i] = rule[i] + " THEN " + name + " = " +
```

```
                        branch[i].agg(lambda
                        x:x.value_counts().index[0])[0] + "."
                        rule[i] = rule[i].replace(" AND  THEN ", " THEN ")
                    skip_update = True
                    continue
                igr = info_gain_ratio(target = branch[i].iloc[:, 0], feature
                = bin_sample[0], uniques = bin_sample[1])
                if igr > float(gain_ratio[0, element]):
                    gain_ratio[0, element] = igr
                    uniqueWords[element] = bin_sample[1]
        if valid_numeric(dataset.iloc[:, element]) == False:
            gain_ratio[0, element] = 0.0
            skip_update = False
            igr = info_gain_ratio(target = branch[i].iloc[:, 0], feature =
            pd.DataFrame(branch[i].iloc[:,
            element].values.reshape((branch[i].iloc[:, element].shape[0],
            1))), uniques = uniqueWords[element])
            gain_ratio[0, element] = igr
        if i > 0 and pre_pruning == "min" and len(branch[i]) <= min_lim:
             if "." not in rule[i]:
                 rule[i] = rule[i] + " THEN " + name + " = " + branch[i].agg(lambda x:x.value_counts().
                 rule[i] = rule[i].replace(" AND  THEN ", " THEN ")
             skip_update = True
             continue

    if i > 0 and pre_pruning == "impur" and np.amax(gain_ratio) < impurity and np.amax(gain_ratio) > 0:
         if "." not in rule[i]:
             rule[i] = rule[i] + " THEN " + name + " = " + branch[i].agg(lambda x:x.value_counts().inde
             rule[i] = rule[i].replace(" AND  THEN ", " THEN ")
         skip_update = True
         continue

    if skip_update == False:
        root_index = np.argmax(gain_ratio)
        rule[i] = rule[i] + str(list(branch[i])[root_index])

        for word in range(0, len(uniqueWords[root_index])):
            uw = uniqueWords[root_index][word].replace("<=", "")
            uw = uw.replace(">", "")
            lower = "<=" + uw
            upper = ">" + uw
            if uniqueWords[root_index][word] == lower:
                branch.append(branch[i][branch[i].iloc[:, root_index] <= float(uw)])
            elif uniqueWords[root_index][word] == upper:
                branch.append(branch[i][branch[i].iloc[:, root_index]  > float(uw)])
            else:
                branch.append(branch[i][branch[i].iloc[:, root_index] == uniqueWords[root_index][word]]

            rule.append(rule[i] + " = " + "{" + uniqueWords[root_index][word] + "}")

        for logic_connection in range(1, len(rule)):
            if len(np.unique(branch[i][0])) != 1 and rule[logic_connection].endswith(" AND ") == False
                rule[logic_connection] = rule[logic_connection] + " AND "
    skip_update = False
    i = i + 1
    print("iteration: ", i)
    stop = len(rule)

for i in range(len(rule) - 1, -1, -1):
    if rule[i].endswith(".") == False:
        del rule[i]

rule.append("Total Number of Rules: " + str(len(rule)))
rule.append(dataset.agg(lambda x:x.value_counts().index[0])[0])
print("End of Iterations")
```

```
    return rule


df = pd.read_csv('weather_datafil.csv', sep = ',')
X = df.iloc[:, 0:15]
print(X)
y = df.iloc[:, 16]
print(y)
dt_model = dt_c45(Xdata = X, ydata = y, cat_missing = "missing", num_missing = "mean", pre_pruning = "impur
test =  df.iloc[:, 0:16]
print(test)
prediction_dt_c45(dt_model, test)

test =  df.iloc[:, 0:15]
y_pred = prediction_dt_c45(dt_model, test)
y_pred['Prediction']

y_true = df.iloc[:,16]
y_true

pip install word2number

from word2number import w2n
y_npred = [0]*248
for i in range(248):
    y_npred[i] = w2n.word_to_num(y_pred['Prediction'][i])

y_npred
from sklearn.metrics import accuracy_score
accuracy_score(y_true[:248], y_npred)
```

```
In [60]: from sklearn.metrics import accuracy_score
         accuracy_score(y_true[:248], y_npred)

Out[60]: 0.8387096774193549
```

CART Implementation

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:



"""Implementation of the CART algorithm to train decision tree classifiers."""
import numpy as np


class Node:
    def __init__(self, predicted_class):
        self.predicted_class = predicted_class
        self.feature_index = 0
        self.threshold = 0
        self.left = None
        self.right = None
```

```
class DecisionTreeClassifier:
    def __init__(self, max_depth=None):
        self.max_depth = max_depth

    def fit(self, X, y):
        self.n_classes_ = int(sum([len(group) for group in X]))
        self.n_features_ = X.shape[1]
        self.tree_ = self._grow_tree(X, y)

    def predict(self, X):
        return [self._predict(inputs) for inputs in X]

    def _best_split(self, X, y):
        m = y.size
        if m <= 1:
            return None, None
        num_parent = [np.sum(y == c) for c in range(self.n_classes_)]
        best_gini = 1.0 - sum((n / m) ** 2 for n in num_parent)
        best_idx, best_thr = None, None
        for idx in range(self.n_features_):
            thresholds, classes = zip(*sorted(zip(X[:, idx], y)))
            num_left = [0] * self.n_classes_
            num_right = num_parent.copy()
            for i in range(1, m):
                c = int(classes[i - 1])
                num_left[c] += 1
                num_right[c] -= 1
                gini_left = 1.0 - sum((num_left[x] / i) ** 2 for x in range(self.n_classes_))
                gini_right = 1.0 - sum((num_right[x] / (m - i)) ** 2 for x in range(self.n_classes_))
                gini = (i * gini_left + (m - i) * gini_right) / m
                if thresholds[i] == thresholds[i - 1]:
                    continue
                if gini < best_gini:
                    best_gini = gini
                    best_idx = idx
                    best_thr = (thresholds[i] + thresholds[i - 1]) / 2
        return best_idx, best_thr

    def _grow_tree(self, X, y, depth=0):
        num_samples_per_class = [np.sum(y == i) for i in range(self.n_classes_)]
        predicted_class = np.argmax(num_samples_per_class)
        node = Node(predicted_class=predicted_class)
        if depth < self.max_depth:
            idx, thr = self._best_split(X, y)
            if idx is not None:
                indices_left = X[:, idx] < thr
                X_left, y_left = X[indices_left], y[indices_left]
                X_right, y_right = X[~indices_left], y[~indices_left]
                node.feature_index = idx
                node.threshold = thr
                node.left = self._grow_tree(X_left, y_left, depth + 1)
                node.right = self._grow_tree(X_right, y_right, depth + 1)
        return node

    def _predict(self, inputs):
        node = self.tree_
        while node.left:
            if inputs[node.feature_index] < node.threshold:
                node = node.left
            else:
                node = node.right
        return node.predicted_class
```

```
if __name__ == "__main__":
    import sys
    import pandas as pd
    import numpy
    from sklearn.model_selection import train_test_split
    df = pd.read_csv('weather.csv').dropna()
    x=df.drop(columns=['Date','Location','RainTomorrow'])
    y=df.drop(columns=['Date','Location','MinTemp','MaxTemp',
    'Rainfall','WindGustSpeed','WindSpeed9am','WindSpeed3pm',
    'Humidity9am','Humidity3pm','Pressure9am','Pressure3pm',
    'Temp9am','Temp3pm','RainToday','RISK_MM'])
    datax= x.to_numpy()
    datay=y.to_numpy()
    x_train, x_test, y_train, y_test = train_test_split(datax, datay, test_size=0.3, random_state=1)
    clf = DecisionTreeClassifier(max_depth=1)
    clf.fit(x_train, y_train)
    y_pred=clf.predict(x_test)
    print(y_pred)

from sklearn import metrics
Accuracy=metrics.accuracy_score(y_test, y_pred)
print("Accuracy:",Accuracy)
```

```
from sklearn import metrics
Accuracy=metrics.accuracy_score(y_test, y_pred, normalize=True)
print("Accuracy:",Accuracy)
```

```
Accuracy: 1.0
```