ID3 Decision Tree Learning Algorithm

University of Malta

Faculty of Information & Communication Technology

Bachelor of Science in Information Technology (Honours) (Artificial Intelligence) 2nd Year

ICS2207 – Machine Learning: Introduction to Classification, Search and Optimisation

Mr Kristian Guillaumier

Francesca Maria Mizzi

I.D. Card Number: 118201(L)

Table of Contents

ID3 Decision Tree Algorithm

A decision tree is a type of machine learning which makes use of nodes and decisions where each branch node is representing a choice between many possibilities while leaf nodes represent a decision. They are mainly utilized the gathering of data with the intention of decision-making. As is the case with most trees in AI, it starts from the root node on which users take actions. Recursively, according to the tree learning algorithm, each node is split. When each branch is a representation of a likely situation of decision and its outcome, the tree has reached its final result [1].

ID3 (Iterative Dichotomiser 3), is a simple decision tree algorithm with the basic principle being to build tree using a top-down, greedy search using the given sets to check every attribute at every node with the goal being to create the shallowest trees possible. During the method, we select the most valuable attribute, the one with the highest information gain, as the current nodes test attribute. Like this, the data required for the classification of the training sample subset which will be acquired from partitioning later on, will be the smallest [2]

Entropy

ID3 decision tree algorithm builds the tree one interior node at a time, beginning at the root of the tree. At every node, we choose which attributes gives us the most information gain if we were to separate the instances into smaller subset based on the values of that attribute. The most valuable information is determined through entropy reduction.

Entropy is a measure of the amount of uncertainty or disorder. If a data set has a lot of uncertainty, it provides us with very little information. It is also a measure of how heterogeneous a dataset is since depending on whether there are many different class types, and whether each class type has the same odds of happening, the entropy of the data set will be high.

Step by Step

The following is a step by step guide into the logic behind creating an ID3 implementation:

1. Calculate the Prior Entropy of the dataset
    a. The prior entropy is the weighted sum of the logs of all the probabilities of each possible outcome. This is the value which indicates to us how much entropy remains before any splitting occurs.
2. Calculate the Information Gain for each attribute
    a. For every attribute in the dataset:
        i. Calculate the total number of instances
        ii. Initialize a running valued entropy score
        iii. Split the dataset into subsets based on the attribute values.
        iv. For each attribute value, calculate the number of instances, the entropy score using the frequency counts, the weighting of the attribute value by dividing the number of instances by the total instances, and the valued entropy score sum.
    b. The final running entropy score sum indicates how much entropy with remain if we were to divide the current data with that attribute
    c. Calculate the information gain, which is the remaining entropy subtracted from the prior entropy
    d. Record the information and repeat for other attributes
3. Calculate the attribute with the maximum information game
4. Split the dataset based on the value of the attribute with the maximum information gain
5. Repeat steps 1-4 for every branch of the decision tree with the relevant partition

Dealing with Overfitting

The way I chose to deal with overfitting the data set is by performing reduced error pruning. Pruning is a technique which decreased the size and complexity of a decision tree by removing any nodes which can be deemed "unnecessary". It is used to reduce "noise" which include instances which have errors, such as being incorrectly classified, or by having an incorrect attribute value [3].

This is done by doing the following:

- Getting training, test and validation
- Once the tree has been trained and built, the tree is tested on the validation set
- Each node is considered for pruning, which means removing the subtree at that node, turning it into a leaf node and assigning the most common class at the node
- The node is removed if the newly generated pruned tree has a classification accuracy that is as good as the accuracy of the first decision tree
- Pruning stop when no more improvement can be made in the classification accuracy

Dealing with Continuous-Valued Attributes

In both the datasets I chose, "Iris" and "Wine", both sets are made up entirely of continuous-valued attributes. When created a decision tree, the attributes should ideally be discrete. Therefore, the way I chose to deal with the continuous attributes is by placing them into ranges, or "bins" during pre-processing. I use the pandas command pd.cut() in order recategorize the continuous data into my chosen amount of bins, which I chose to be 5. That way, the previously continuous data is now split up into 5 different ranges and can be implemented in the ID3 decision tree much easier.

Datasets

The two datasets I chose to make use of are the "Iris" dataset as well as the "Wine" dataset. The wine dataset is made of 178 instances and 13 columns while the iris dataset is made up of 150 instances and 5 columns.

I chose these datasets due to them having a decent amount of instances, allowing for an appropriately sized test and train set for the decision tree. I also chose them due to the fact that all the variables in both datasets are continuous, allowing me to show my code in handling continuous-valued attributes.

Implementation

Below I will be discussing directly my implementation of the ID3 decision tree, describing the purpose of each method and class used.

Importing relevant libraries

Since the goal was to create the decision tree from scratch, not many libraries were used.

- The pandas library was used in order to handle the data
- The numpy library was used for mathematical calculations
- The matplotlib library was used to plot the graphs
- The math library was used to calculate the log
- The collections library was used to count instances
- The sklearn library was used in order to split the datasets into train and test sets

preprocess(dataset)

This is the function which was used to carry out the pre-processing on the datasets, turning the continuous attributes discrete. In it, I cycle through the columns in the dataset and use the pandas function cut() in order to segment the data into a set amount of bins, which I chose to be 5. This new data is added to the data frame and the old value removed from the frame.

Class Node

This class contains only the constructor for this class which are used to assign the attributes of a node in the tree. It contains the values of 8 attributes:

- label = the class label for the node
- attribute = the attribute itself
- values = the values of the attribute
- children = keeps track of the node's children
- pruned = Boolean to see if the node is pruned or not
- instances_labeled
- pAttribute = a reference to the node's parent node
- pValue = the value of the parent node

most_informative_attribute(instances)

This function chooses the attribute which gives the most information if you were to split the dataset based on that specific attributes value. The first instance in instances for key is extracted, value iterates through each key-value pair in the first instance in instances. By the end, it creates a new list containing the names of the attributes.

mode_class(instances)

This function returns the name of the most common class by looping through each instance in the list, appending each one to the list of classes and using the counter library to find the most common class.

prior_entropy(instances)

This function calculates the entropy with respect to the previous class before segmenting the data. Every instance in the list of instances us added to the end of the classes list and the weighted sum of the logs of the probabilities of every probable result is computed. If the instances all belong to the same class, the entropy is 0.

entropy(instances, attribute, value)

The entropy for the data filtered by the attribute value is calculated in this function. Similarly to the previous function, every instance in the list of instances is added to the classes list and the weighted sum of the logs of the probabilities of every probable result is computed. If all the instances are from the same class, the entropy is 0.

gain_ratio(instnaces, attribute)

In this function, the gain ratio is calculated if the data set were to be segmented based off of the values of this specific attribute. The prior entropy of the combined instances is found. A list holding the attribute values of each instances is created and the frequency counts are stored.

accuracy(trained_tree, test_instances)

Calculates the classification accuracy of the ID3 Decision Tree

predict(node, test_instance)

This function is a recursive function which returns to the user the name of the class. This is done by recursively calling the function until we get to a leaf node. We first get the name of the attribute from the node and store that value for the test instance into the variable attr_value. Assuming it is an unpruned tree, we continue to follow the branch for this value and recursively call until we reach the stopping condition.

prune(node, val_instances)

This is the function used to prune the tree as discussed earlier. It first check if the node is a leaf node. If there is no improvement in accuracy, pruning does not take place, however, if the accuracy does improve, the tree is recursively pruned.

ID3(instances, default)

This is the main function for the ID3 decision tree algorithm. The function starts by creating an empty “classes” list. For each instance in the list of instances the value of the attribute is appended to the “classes” list. If all instances have the same label or only one instance is present, a leaf node is created and labelled with that class. The counter then creates a tally of each element in the list. If not, the best attribute is found and the attribute that maximizes the gain ratio of the data and set it to be the next decision node. A list is then created with the list of the values of the unique best attributes. The instances are then split according to the instances in order to create the child nodes. The subset is generated and as it cycles through the instances, if the instance has the best attribute value, it is added to the list. A new subtree is created recursively, and the values initialized. We then add the variables to keep track of the state of the subtree’s parent node and the subtree is assigned to the appropriate branch.

Unfortunately, when trying to run all the above methods together, I came across an issue involving data handling were variables weren’t being passed the way they should and the data inside of the data frame was getting distorted.

Statement of Completion

Item	Completed (Yes/No/Partial)
Dataset selection and import	Yes
ID3	Partially, it doesn't run as intended
Support continuous attributes	Yes
Overfitting management	Partially, implemented but doesn't work
Good discussion on an alternative method	No
Experiments and evaluation	No

References

[1] W. Peng, J. Chen, and H. Zhou, "An Implementation of ID3-Decision Tree Learning Algorithm."

[2] C. Jin, L. De-lin, and M. Fen-xiang, "An improved ID3 decision tree algorithm," in *2009 4th International Conference on Computer Science Education*, 2009, pp. 127–130, doi: 10.1109/ICCSE.2009.5228509.

[3] Y. Zhang, Z. Chi, and D. Wang, "Decision Tree's Pruning Algorithm Based on Deficient Data Sets," in *Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT'05)*, 2005, pp. 1030–1032, doi: 10.1109/PDCAT.2005.111.