Assignment 2 - CT4101 Machine Learning

Oisín Hodgins 4BMS
17480216
o.hodgins1@nuigalway.ie

1.0 Team Members

I undertook this project alone.

2.0 Description of Algorithm and Design Decisions

2.1 Choosing an Algorithm

In the previous assignment I chose K-NN and naïve bayes, this time around I decided to use a decision tree based algorithm, to broaden my experience with this module. As such I chose the C4.5 algorithm.

2.2 Brief Description of Algorithm

I will keep this section short, as the reader of this report knows far more about C4.5 than I do!

Similar to Ross Quinlan's ID3 algorithm, which was discussed in class, we will use the concepts of entropy and normalized information gain from information theory to create a decision tree. We calculate the normalized information gain for each feature, based in the reduction in entropy if we were to partition the data using that feature. A node is created for the feature with the highest normalized information gain.
Each node in the tree corresponds to a feature in the feature space of the data, with child nodes associated with each value of the feature.
During testing, we pass an instance of the test data to the decision tree. Each node will specify either a class to assign to this instance, or will send the instance to one of its child nodes based on the node's associated feature.

2.2 Advantages/Disadvantages of C4.5

I do not intend to repeat any unnecessary information here, however the C4.5 algorithm has some important advantages when compared to the ID3 (Also developed by Ross Quinlan) algorithm, and to the other recommended algorithms for this assignment.

Notably, when compared to ID3, the C4.5 algorithm can easily handle continuous attributes by splitting the attribute based on some threshold value[1]. This useful aspect of the algorithm was a major factor in my decision to use it, as all of the features in the data provided are continuous.

One notable disadvantage of the C4.5 algorithm is it's high run time and demand for computational power, thus it is most suitable for smaller datasets (the data must be stored in memory to create the tree , and the data is passed over many times in the creation of the tree). As the dataset provided with the assignment is relatively small, I felt that C4.5 would be appropriate

3.0 Design Decisions

3.1 Multi-Class C4.5

The C4.5 algorithm is intended to be used for binary classification, however there are 3 classes in the training data. After some initial research, I decided to transform the multi-class data into binary data using a one-vs-one approach.

In one-vs-one multi-class classification, we split the training data into pairs of classes, and create a model for each pair. In this assignment, we had the following three pairs:

'ale' vs 'stout'
'ale' vs 'lager'
'lager' vs 'stout'

If there are k classes, we will create k*(k+1)/2 models. When the models are complete and we are testing the data, each model 'votes' for one of its two classes to be the final prediction. We take the final prediction as the class with the highest number of votes.

One notable downside here, is the case where two classes receive an even number of votes, in this case when each of the three models votes for a different class. I did not address this downside directly, as there is an inherent randomness in the algorithm when it is partitioning the data and counting the number of unique classes. Ideally my implementation should choose a class randomly in this case.

3.2 Visualisation of the Decision Tree

While I would have liked my implementation to include a visualisation, I ran into several hurdles trying this and decided not to include it in the end. However, if one is willing to inspect the console output thoroughly, they could piece together the decision tree based on the unique node ID's shown. This is discussed further in section 4.

3.3 Overview of the Code

Finally we can discuss the code I used to implement C4.5, this section should serve as a guide to the code, as my variable naming and functions are not optimised/easily-legible!

The language used is python.

3.3.1 Inputting data

First, the user is prompted to input their data which will be split (randomly) later on into training and test data. If so desired, the user can input their own training and test data (in separate files) however I have not extensively tested this so I cannot guarantee it will work as expected.

The user is prompted to enter in some more details, including an output file. Please note now that a brand new file should be specified, as the contents of the file inputted will be deleted!

3.3.2 Creating the Decision Tree

Next we will run each of the one-vs-one models, only using the relevant classes each time.

A node class is used to create the decision tree, and to classify test data. The details are specified in the comments of the code, and I will repeat the important ones here:

Regular (non-leaf) nodes are created with a specific index corresponding to one feature of the training data, this feature cannot be used again in that branch of the tree. A feature may appear

in different branches of the tree however, I made the decision to allow this behaviour as it resulted in a higher classification accuracy.

Regular nodes will have some children associated with them, and when classifying test data they will evaluate the test data based on the node's feature. They will then call one of their children nodes and pass the test data to them accordingly.

Eventually a leaf node is reached, and a class value is returned back through each node of the tree.

When building the tree, we evaluate the ratio between the normalized information gain for each feature, and create a node with the highest gain.

For continuous features, in the gain_ratio_ continuous() function, we attempt threshold split at each value, and select the threshold which provides the greatest normalized information gain.

After we have created the root node, we recursively call the algorithm (specifically the c_algorithm() function) to determine the children of the root node. This will iterate until a leaf node is reached in each branch.

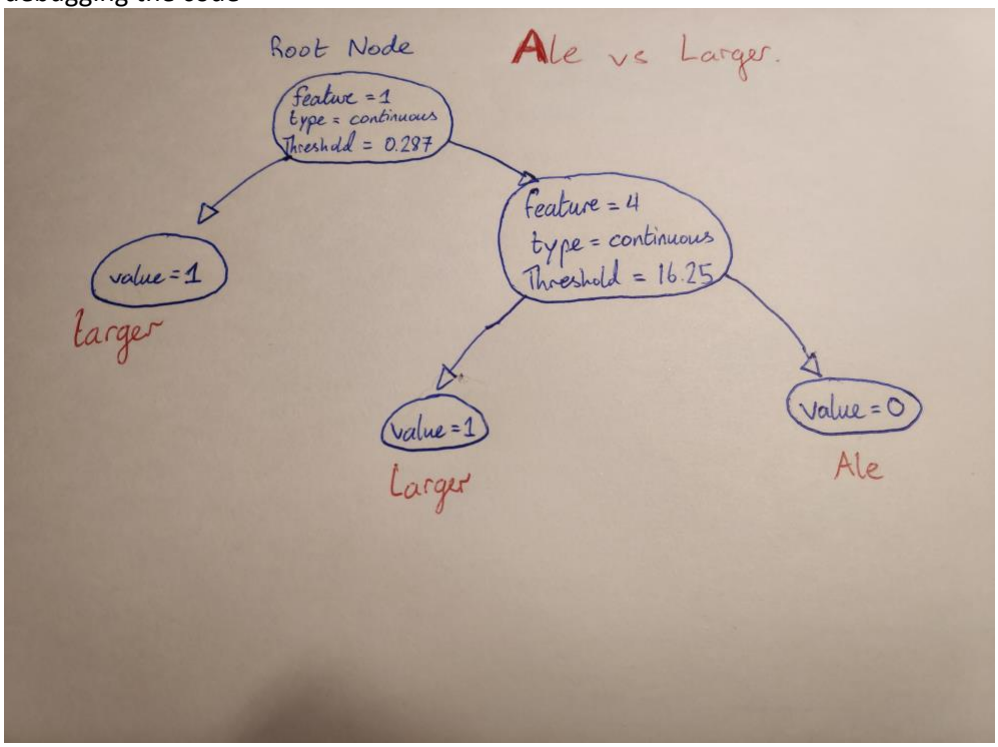
3.3.3 How Data is Tested & Output

Tests occur for each model immediately after creation. The predicted classes are stored in one array along with the actual classes, and after all models are finished we choose a final prediction for each instance in the test data by majority vote.

This is all outputted to the specified file, following the assignment instructions the algorithm will write ten large arrays of predictions, however the details are summarised at the end of the file and also explained there.

3.3.4 Visualisation

Although I did not get a visual implementation working, I have included one of the outputted trees from my algorithm. Using the unique object IDs in python one can easily check the statistics of each node, to better understand the decision rule. Doing rough sketches like theses helped with debugging the code



4.0 Comparison with a ML package

My code is very inefficient, with many unnecessary variable assignments and iterations of particular functions. Unsurprisingly the C4.5 algorithm from the chefboost ML package ran much quicker.
The ML package, over several iterations, achieved a rough average of ~85%, which is very similar to my own algorithm.
However, this may be due to overfitting on my part, as my implementation does not undertake any pruning. As a result the created tree may be overfitted to the test data.

4.1 Comparison with Assignment 1

In assignment 1 I used R instead of python, which for me made plotting and visualising the data much easier. More importantly, in assignment 1 my code was much slower than it is in this current implementation. Previously I used K-NN and Naïve Bayes, both of which gave ~90% prediction accuracy, however they were implemented from a public open source reliable package. I do not believe that this is any indication that those are superior algorithms in this context, only that my implementation was subpar.

5.0 Conclusion and Observations

Overall this implementation is slow and inefficient, if I were to do it again I'm sure it could be done in many fewer lines. I believe C4.5 algorithm is a great improvement over ID3, and in this situation where computation time was low, proved highly effective.

6.0 References

Paper (J. R. Quinlan):

[1] "Improved use of continuous attributes in c4.5". *Journal of Artificial Intelligence Research, 4:77-90, 1996*

Available at:

https://arxiv.org/pdf/cs/9603103.pdf

5.0 Appendix of Code