Implementation of DT_Train_Binary

DT_Train_Binary was created that had three parameters: X, 2D numpy array which correspond to the given training data, Y, a 2D numpy array which corresponds to the label of the samples of the data set, and maximum depth, a hyperparameter chosen by the programmer.

Inside the function, a list was made to keep track of the feature names of the original dataset. Then, another function was created. ID3, named after the algorithm that was implemented, was made in order to keep track of the depth for the recursions. ID3 has the following parameters:

samples: A 2D array of the training data - might be seperated later on

originalLabel: A 2D array of the original training labels - used to keep the original, untouched dataset. This was to use for the case where the training data was empty.

label: A 2D array of the labels. These labels can be separated later on, as well

headers: A list of the feature names

max_depth: The maximum amount of questions that can be asked, essentially

depth: The current depth of the tree

Inside ID3, stopping conditions were set to stop the recursion. There were three stopping conditions: if the depth of the current tree was equal to the max depth, if the dataset was empty, and if there were only one type of samples in the label set. If any of those were true, we would return a leaf node.

Then the information gain was calculated using each feature and those gains were stored in a list. A function that helped was the calc_entropy function which takes in the label for that dataset, and returns the entropy. The calc_attr_entropy function was used to calculate the information gain of each feature. A tree dictionary object with the best feature index as the root was created. The header for feature was removed from the list of headers.

A split function was also created which separates the data into the left and right subsections with the corresponding left and right labels. This is necessary for the recursion step in which we call the ID3 on the right and left subsections. The ID3 is called on the left and right subsections, and the depth is increased.

Finally, the final decision tree dictionary object is returned.

Implementation of DT_make_prediction

The implementation of DT_make_prediction was simple. It takes in as parameter a single sample (a row), and DT dictionary object, and returns the prediction that the decision tree comes up with.

In a for loop for every key in the DT tree dictionary, whose DT.keys() value is the root node, the branch holds the value from the sample which corresponds to the index of the root or given node. The tree is now reduced to the tree (or leaf) that branches off from that value.

The type of the tree is checked, and if it is still a tree, a recursive call is made to the remaining tree, with the sample. If it is not a tree, it returns the prediction.

Implementation of DT_test_binary

In DT_test_binary, there are three parameters:

X: data to do testing on

Y: the label for the testing

DT: the decision tree object

An empty list is created to hold the predictions made using the decision tree. Then, in a for-loop, in which it iterates through each row in the X array, the predictions of each sample are appended to the list by using the DT_make_prediction function.

The prediction list is then converted to an array and a comparison test is used to see if the predictions match the labels. If they do, then the count variable is incremented.

After the loop, the number of correct over the total number is calculated.

Implementation of DT_train_binary_best

DT_train_binary_best holds 4 parameters:

Training data and labels and validation data and labels.

An empty list is created that will hold the decision trees. For each feature in the training data, it appends the decision tree produced for each possible depth to the list of decision trees.

Another empty list will be created to hold the accuracies for the decision trees. The accuracies are calculated by using the test_binary function on the validation data.

The index with the highest percentage accuracy is chosen and the function returns the value of the decision tree list with the index, which is a tree.

Implementation of DT_train_real

In DT_train_real, there are three parameters:

X: data to do testing on, which in this case are real values

Y: the label for the testing, which in this case is binary

DT: the decision tree object

The implementation of this algorithm/decision tree is similar to the binary one, but in this example, we have real values and we need to find a threshold in the range of each feature value that gives the best split and highest information gain. We used a greedy algorithm on each feature column that evaluates all feature values in a loop and finds the feature value with the highest information gain and will take that value as the threshold for splitting each column/feature.