Flowchart for writing a decision tree algorithm using the ID3 algorithm :

1. Start.

↓

2. Define a Node class to represent decision tree nodes.

↓

3. Define Decision Tree class.

↓

4. Intialize Decision Tree with hyperparameters such as max_depth and min_samples_split

↓

5. Define an entropy() function to calculate the entropy of a given set of samples.

↓

6. Define an information_gain() function to calculate the information gain of a given split.

↓

7. Define a build_tree() function to recursively build the decision tree.

↓

8. Define a ~~best~~ get_best_split() function to find the best split for a given set of samples.

↓

9. Define a split() function to split a set of samples based on given features and threshold value.

↓

10. In built_tree() function, check if stopping conditions are met (eg. min_samples_split or max_depth reached)

↓

11. If stopping conditions are not met, use get_best_split() to find the best split.

↓

12. If information_gain is positive, recursively call build_tree() on the left and right subtrees.

↓
13. Create a leaf node if stopping conditions are met or information gain is negative
↓
14. Return the root node of the decision tree
↓
15. End.

Pseudo code.

1. Node class

```
START
  CLASS Node.
  FUNCTION __init__ (self, feature_index = None, threshold = None
                     left = None, right = None, info_gain = None
                     Value = None)
  SET self.feature_index to feature_index #Index of feature that
                                           node splits on
  SET self.threshold to threshold # the threshold value used to
                                    split the data at this node.
  SET self.right to right # right child of this node
  SET self.left to left # left child   "   "
  SET self.info_gain to info_gain # the information gain obtained
                                    by splitting at this node.
  SET self.value to value # predicted value of target variable
                            at this node (leaf node)
  END FUNCTION
 END CLASS.
END.
```

2. Define Decision Tree Class

```
START.
 CLASS Decision_Tree__Classifier.
 FUNCTION __init__ (self, min_samples_split = 2, max_depth = 2)
 # initialize the root of the tree.
 SET self.root to None.
```

stopping conditions.

SET self.min_samples_split to min_samples_split. # min sample req to split the node.

SET self.max_depth to max_depth. # max depth of tree.

END FUNCTION

END CLASS

END.

3. Define Entropy()

$$\text{Entropy}(P) = -\sum_{i=1}^{N} P_i \log_2 P_i$$

START

FUNCTION entropy (y)

compute the unique class in y

SET class_labels to the unique elements in y.

initialize entropy to zero.

SET entropy to 0.

iterate over each unique class_label.

FOR each cls in class_labels.

compute the proportion of samples with class label cls. (Probability)

SET p_cls to the number of samples in y with class label cls divided by the total no. of samples.

SET entropy to entropy + $\underbrace{-\text{p\_cls} * \log_2(\text{p\_cls})}_{\text{class entropy}}$

END FOR.

return the computed entropy

RETURN entropy.

END FUNCTION

END.

4. Information_gain ()

$$\text{Information gain} = \text{Entropy (Parent)} - \begin{pmatrix}\text{Weighted}\\ \text{average}\end{pmatrix} \text{entropy (children)}$$

$$= I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

START

FUNCTION information_gain (parent, l_child, r_child)

calculate the entropy of the parent node

SET parent_entropy to the result of calling the entropy fn with parent.

```
# Calculate the entropy of the left child node.
  SET leftentropy to the result of calling the entropy function
                                                with l-child.
# calculate the entropy of right child node.
  SET right-entropy to the result of entropy (r-child)
                                   entropy
# Calculate the weighted avg ^ of the child nodes.
  SET child_entropy to (len (l-child) / len (parent)) * left_entropy
                  + (len (r-child) / len (parent)) * right_entropy

# calculate informn gain.
  SET gain to parent_entropy - child_entropy
# return the computed infmn gain
  RETURN gain
END FUNCTION
END.
```

5. Define build-tree()

```
START
                              dataset,
  FUNCTION buit-tree ( ^ curr-depth):
    X, Y = get-features & target from dataset.
    num-samples, num_features as shape of X. # get no: of samples & features
    # check stopping condition.
      IF num-samples ≥ min-sample-split & curr-depth <= max-depth
        # find best split.
        best_split = get-best_split (dataset, num-samples, num_features)
        # check if infmn gain is positive.
        IF. best-split ["info-gain"] > 0
          left-subtree = build-tree (best_split ["dataset_left"], curr-depth+1)
          right-subtree = build-tree (best-split ["dataset-right"], curr-depth+1)
          # return decision node

          RETURN Node
    # compute ~~leaf~~ leaf node
    leaf value = calculate_leaf-value (y)
    return Node
END
```

6. get_best_split ( )

START

FUNCTION get_best_split (dataset, num_samples, num_features)

~~define~~ Initialise dictionary to store best split & variable to store maximum infmⁿ gain.

best_split = { }

max_info_gain = ~~-infinity~~ .float ( )

loop over all the features & their possible threshold.

FOR feature_index in range of num_feature

feature_values = dataset (:, feature_index)

possible threshold as unique values of feature_values

Split dataset into left & right subset using the split ( ) method.

FOR threshold in possible_threshold.

dataset_left, dataset_right = split (dataset, feature_index, threshold)

compute information gain.

current_information_gain as information_gain (y, left_y, right_y)

If the infmⁿ gain is greater than the maximum information gain seen so far, update the best split with current split & new max infmⁿ gain

IF curr_info_gain > max_info_gain :

best_split ["feature_index"] = feature_index.

best_split ["threshold"] = threshold

best_split ["dataset_left"] = dataset_left

best_split ["dataset_right"] = dataset_right

best_split ["Info_gain"] = curr_info_gain

max_info_gain = curr_info_gain.

RETURN best split # returns the best split.

7. Split ( ) :- split dataset into two subset.

START

FUNCTION split (dataset, feature_index, threshold)

```
# Initialise 2 ~~to~~ empty arrays
    dataset_left = empty numpy array
    dataset_right = empty numpy array
  # Iterate/loop over each row in the dataset & append
    them to left subset if the feature value is less
    than or equal to the threshold, and to the
    right subset otherwise.

FOR ~~IF~~ each row in dataset
      IF row[feature_index] <= threshold
         append row to dataset_left
      ELSE
         append row to dataset_right.
   # returent left & right subsets
   RETURN dataset_left, dataset_right.
```

Commit