CS550 - Machine Learning
Homework #1 - Decision Trees

Ayhan Okuyan
ayhan.okuyan@bilkent.edu.tr
Bilkent University, Ankara, Turkey

CONTENTS

I. INTRODUCTION

This homework investigates the contrasts between constructing and managing decision trees with varying structures. The first part asks us to use a library to train and test trees applying various techniques such as pruning, and data manipulations such as normalization or subsampling. The second part of the homework required us to create our implementation of the decision tree using prepruning and the third part asked us to reiterate our implementation to be cost-sensitive, which considers the cost of each feature before using them. One important note is that the implementation I have performed is not specific to this dataset, it is a general decision tree class structure that can be applied to any type of numeric dataset that doesn't contain missing values. The only modification needed is that the cost array passed to the program should be modified such that the cost for the features that are a combination of multiple features should be a list containing the indices of their components. For implementation, we have chosen to use Python3 for each part, and the source codes are provided in Jupyter Notebook in ipynb format.

II. DATASET

Before we moved on towards the implementation process, we have focused on understanding the data. As given in the process, this is one part of the Thyroid Disease Data Set of UCI Repository [1]. The data we work on is given as "A Thyroid database for training ANNs" and it contains 3772 training, 3428 test instances, with 20 features (costs included), and 3 classes to represent hyperthyroidism, hypothyroidism, and healthy. When we observe the data, we see a highly unbalanced distribution of classes weighing towards the **healthy** class. As also given with the assignment, we see that one of the features given is an extracted version of features 19 and 20. After learning the dataset by observation, we continue the implementation required. Furthermore, we didn't consider the features differently as binary or continuous in our implementation as it didn't physically matter in the implementation. As we know, there can only be one split for the binary case, and by ignoring we lose some efficiency but that doesn't change the outcome.

III. PART 1

This part of the homework asks us to use a framework to define and use decision trees for the given data [1]. For this part, we have used **entropy impurity** for all the trees we're investigating.

A. Pruning vs No Pruning

First, we are asked to use the pruning option to find the best test accuracy and print its training and test results, together with the class-based accuracies. For that, we have used sklearn's 'min_impurity_decrease' option with a threshold value of 0.05, which finds the difference between the parent node and the weighted impurity of its split, then scales this value for the numbers of instances in the node (N_m/N). The algorithm is as follows as described in sklearn API [2].

$$= \frac{N_t}{N}(impurity - \frac{N_{tR}}{N_t} * right\_impurity - \frac{N_{tL}}{N_t} * left\_impurity) = \frac{N_t}{N}\Delta I_s$$

where s refers to the split s of the node. The results with no pruning and pruning are presented in 1, 2 and 3, 4 respectively. We observe that the test accuracy is improved from 99.271% to 99.358%. We see that when we don't use

pruning, the algorithm overfits the training data, reducing its generalizability. Hence, we see a remarkable improvement in terms of test accuracy in the case we use pruning. We can also claim that the model learns the majority class due to the imbalance in the training set and opts to assign poorly the minority classes in both trees.

B. Normalizing the Data

In this part, we have normalized the data and observed the results. For data, normalization can be achieved with,

$$\tilde{X} = \frac{X - \mu_X}{\sigma_X} \tag{1}$$

where

where μ_X, σ_X is the mean and std vectors of the data features. However, we should note that the test data should also be normalized concerning the mean and variance of the training data. The results are presented in Figures 5 and 6. We see that although the threshold values have changed, the overall tree structure has not. Since this is a linear transformation, we didn't expect the decision tree model to react to such a change. However, since the test data is also scaled to the range of training, we expected test results to change, as they did. we cannot be sure of the direction. This time, the result was a change towards more inaccurate decisions.

C. Dealing with Unbalanced Data

In this part, we are asked to use a class-balancing option since this is a highly unbalanced dataset. To deal with it, we have chosen to implement a subsampling algorithm. For each class in the training data, we have randomly sampled the data according to the presence of the minority class. For tractability, we have set the RNG (Random Number Generator) to a particular value. The results are presented in Figures 7 and 8. Here we see that the number of samples from each class is equal to the minimum, which is 93. However, the results were not as good as desired since the number of samples was far too little to learn the variance of the data, especially for class 3. One improvement that could be done was to bootstrap from the minority class instead of decreasing the training size.

IV. Part 2
Decision Tree Implementation with Prepruning

In this part, we have implemented a general, choice-based decision tree class implementation. This implementation considered two options of prepruning: One is the maximum depth, and the second one is impurity decrease prepruning. To build a generalizable tree structure, we have taken input from the user as to which impurity they would like to use. The implementation details and the results on the dataset are provided in this part of the report.

A. Definition of Impurity Metrics

and offered two as in: Gini Index and Entropy. Given the node labels, these are calculated as follows:

$$I_{gini}(m) = 1 - \sum_{i=1}^{N_c} P_m(C_i)^2 \tag{2}$$

$$I_{entropy}(m) = -\sum_{i=1}^{N_c} P_m(C_i) \log P_m(C_i) \tag{3}$$

where N_c is the number of classes and $P_m(C_i)$ is the probability of any sample being in the class. To make one important note, the entropy is measured with **bits** when the logarithm is taken over base 2, **dits** if 10 and **natural units** if e. For our implementation, we have used log2, and will measure impurity with bits. Hence, we can approximate C_i, which is a statistically term originally as,

$$P_m(C_i) = \frac{\sum_{j=1}^{N_m} I(y_j, C_i)}{N_m} \tag{4}$$

$$where \quad I(x,y) = \begin{cases} 1 & if x = y \\ 0 & else \end{cases}$$

This, is ultimately the count of the instances in the node m, that belong to class C_i divided by the total number of samples in the node. After configuring these metrics, we have continued to define the best split of a node. In the implementation, we can set the impurity measure via passing "impurity='gini'" or "impurity='entropy" to the tree.train function.

B. Definition of Best Split

Before, the best split, we need to define the binary split of the data for some feature x_k and its corresponding labels y, which is,

$$n_{left} = \{(x^{[i]}, y^{[i]}) \in m \quad | \quad x_k^{[i]} \leq \theta\} \tag{5}$$

$$n_{right} = \{(x^{[i]}, y^{[i]}) \in m \quad | \quad x_k^{[i]} > \theta\} \tag{6}$$

where θ is the linear decision boundary of the node. The definition of best split is the split that minimizes the weighted impurity of the children nodes. In our implementation, as most frameworks do, we will be defining a binary split for each node, and create a **univariate** structure. Hence the weighted split impurity given left and right nodes as given as follows for the k^{th} feature.

$$I_k(n_{left}, n_{right}) = I_k(\theta) = \frac{N_{left}}{N} I(n_{left}) + \frac{N_{right}}{N} I(n_{right}) \tag{7}$$

where, N_{left}, N_{right} refers to the number of samples in the respective child nodes separated for the k^{th} feature and N is the total number of samples in the parent node. Hence, we can say that,

$$N = N_{left} + N_{right} \tag{8}$$

Hence,

$$I_k(\theta) = \frac{N_{left}}{N} I(n_{left}) + \left(1 - \frac{N_{left}}{N}\right) I(n_{right}) \tag{9}$$

As we have defined the weighted impurity for split s, we can define the objective function for the best split for one feature x_k.

$$\theta^* = arg\min_{\theta} I_k(\theta) \tag{10}$$

Hence, we can define the the best possible split as the best among all K features.

$$(\theta^*, k^*) = arg\min_{\theta,k} I_k(\theta) \tag{11}$$

This completes the implementation of the tree structure, the algorithm we followed here is the ID3 algorithm. Therefore, we can search all the splits sequentially to find the best split and threshold. For the θ values we have chosen to use the respective feature values for each feature.

C. Prepruning

When we grow the whole tree with in training, while we achieve perfect training accuracy, we overfit to the training data. In theory, it wouldn't be a problem if the training data captures all variance over the possible situations, however, since it usually is not the case, we are losing the generalizability of the model when we overfit to the training data. To prevent that, we employed two distinct prepruning algorithms, which stops the training early depending on some conditions. These are given as,

- Max-Depth Prepruning
  Given a maximum depth, stops further growth if the current node's depth is equal to the maximum depth. This is fairly a simple algorithm that only concerns with the depth and acts on the axiom that depth is directly related with overfitting. In the implementation we use "prun = ['depth', maxDepth]" to set the parameters and the pruning method (see the source code).
- Minimum-Impurity Decrease Prepruning
  the impurity decrease is defined as the difference of impurity between the parent node and the weighted impurity of the best split. Here, $I(\theta)$ refers to the weighted impurity of the split given the threshold θ as explained in the previous part.

  $$\Delta I(m, \theta) = I(m) - I(\theta) \tag{12}$$

  Since, we use the log2 as our entropy measure, when entropy is used, we can call this the information gain of the split. That is why we use "prun = ['gain', minGain]" in the implementation (see the source code). This ideally enforces a lower-limit on the gain of information in each split, and considers the split unnecessary if it is below the given threshold.

D. Non-Algorithmic Details

After the training algorithm is prepared, we have prepared functions to traverse the tree and find the results of test data (see tree.predict), to plot the tree (see tree.plotTree), and print the training and test results (see tree.traintestresults). This implementation uses Graphviz to draw the tree, installation of both the software and the python package is required [3].

E. Results

After the implementation is complete, we have investigated to find the best model using a grid search method for each pruning technique. We found that the best test accuracy is found when the gain is 0.05 with test accuracy equal to 99.273% and depth is 5 with 99.329%, which are both improvements from the base case of 99.183%. The tree structures, train, test, and class-based accuracies are provided in the Appendix with the confusion matrices. The result is comparable to the sklearn result that we have found in Part 1, however, it is a little less since we do not scale the impurity decrease for the number of samples in each node. Nevertheless, this is an acceptable result, we see that while the training accuracy decreased the test accuracy improved, which is a sign that the model has become more generalized. However, we should consider that the only reason we are optimizing for the test set is that the training and test are given fixed in the assignment. Under normal circumstances, this is considered an unorthodox method as we are inevitably biasing our model towards the test data.

Furthermore, since the splits are identical when using Gini or entropy, we have moved with using Gini while optimizing since Gini is more efficient in the sense that it doesn't use logarithm operation. Overall, the minimum impurity decrease pruning provided a better result as expected since thresholding the information gain is a more logical option from an information theory perspective than crudely using the depth of the tree.

V. Part 3
Cost-Sensitive Decision Tree Implementation

This part requires us to extend our implementation in Part 2 by incorporating cost-sensitivity to our algorithm. In reality, each feature has a cost of extraction. Whether it be the cost for a medical test or asking the patient his/her age, each feature/action is associated with a cost.

A. Splitting Criteria

Here, we are given the costs of each feature and we extend our implementation to incorporate those features. Here we use the splitting criteria defined by Nunez [4] and Tan [5]. The splitting criteria, also incorporated in the course slides [6] are given as follows.

$$score_{tan}(s, f) = \frac{gain(s, f)^2}{cost(f)} \tag{13}$$

$$score_{nunez}(s, f, w) = \frac{2^{gain(s,f)} - 1}{(cost(f) + 1)^w} \tag{14}$$

$$where \quad gain(s, f) = I_{entropy}(s) - I_{entropy}(f) \tag{15}$$

Furthermore, for a more controllable option, we have used a linear criterion under the name 'linear', which does the following.

$$score_{tan}(s, f) = gain(s, f) - \alpha cost(f) \tag{16}$$

Here, we can see that instead of minimizing the impurity, we are maximizing the information gain. Since we are using

information gain, we will be using entropy (log_2) as the impurity measure to be truthful to the information theory definition. One other important point we should consider is that we are using the cost to scale the gain. Hence, we are searching for a trade-off between the gain of using that split and its cost. The parameters α and w represent the importance of the cost compared with the information gain. This makes the algorithm to opt for a tree that doesn't always return the best split but tries to find a common ground between low cost and good gain. Since we have implemented both impurities (Eq.2-3) and both of these cost-sensitive splitting criteria into the same class structure, to use one, the user should put either "choice=[gini]" or "choice=['entropy']" to use impurities, or "choice=['tan']", "choice=['nunez',w]" or "choice = 'linear', α"to choose the splitting option. Since we opted for the algorithm to minimize the impurity, we have chosen to minimize the negative of the splitting criteria rather than maximizing it (see the source code).

B. Cost of the Extracted Features

As explained in Dataset section, one of the features is extracted from two other features. In order to find the cost of the given feature at the given split, we need to trace back to the root of the tree to see if the cost of the features it's extracted from are paid, as proposed in Algorithm 1.

Algorithm 1: Algorithm to find Cost of the Extracted Feature

```
Result: Cost of the Extracted Feature
cost = 0;
node = currentNode.parent;
listOfFeatures = list of the features the feature is
  extracted from;
costs = zeros vector with the same length as
  listOfFeatures;
while node.parent is not None do
    if node.splitFeature is in listOfFeatures then
        costs[index of node.splitFeature] = 1;
    end
    cost = costVector · costs;
end
```

As we set the new value for the extracted feature for each iteration, we also need to update the cost vector according to the previous splits. The costs for features, if they are previously used for a split in the path from the root to the current node of the tree, should become zero as they are already extracted. Furthermore, if a feature that is a combination of the other features is used, the further nodes should receive zero for both the combination feature and its components.

C. Results

After completing the interpretation, we have grid searched for the best parameters with each choice of splitting criteria, however, the process was intractable hence, I have manually followed a split search algorithm to find the best performance. I was not able to run [5] as we set the costs to zero for the used features, the denominator became zero and the criterion became infinite, caused the algorithm to always choose that feature. For the other two criteria, we have found the test accuracy to be 99.329% with the linear ($\alpha = 0.01$) and Nunez ($w = 0.02$) with information gain pruning 0.1 bits for both. This results in a successful outcome since this is the exact limit of test accuracy we achieved without using the costs in Part 2. The trees and the accuracies together with the class-based accuracies are given in the Appendix. Here, we observe that the trees are highly similar to each other. Also we observe that while TSH and the extracted dominates the first two splits (they are the highest cost metrics), however, the algorithm leans to using either TT4 or T3 since they are paid for when the extracted is used. we can see that after classifying most of the 3 labels with the first split and all of the samples in the second split, the algorithm focuses more on separating the rest, and with the parameters we provide becomes cruder to the changes over iterations. Finally, we decide that although they present the same test accuracy, we selected the model that uses [4], for the best model since it uses one less split.

D. Average Costs of Classes

After finding the best model, we find the average costs for classifying samples from each class. The results are as follows.

```
Average Cost of Class 1: 48.70
Average Cost of Class 2: 50.69
Average Cost of Class 3: 23.61
```

We easily observe that the cost for classifying the third class is nearly equal to the cost Of TSH, since most of them classified in that node. and the cost for the first class is approximately $TSH + extracted$.

References

[1] Quinlan, *UCI machine learning repository thyroid disease data set*, 1987. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/thyroid+disease.

[2] *Sklearn.tree.decisiontreeclassifier*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

[3] *Graphviz graph visualization software*. [Online]. Available: https://graphviz.org/download/.

[4] M. Nunez, "The use of background knowledge in decision tree induction," *Machine Learning, 6, 231–250*, 1998. DOI: 10.1007/BF00114778.

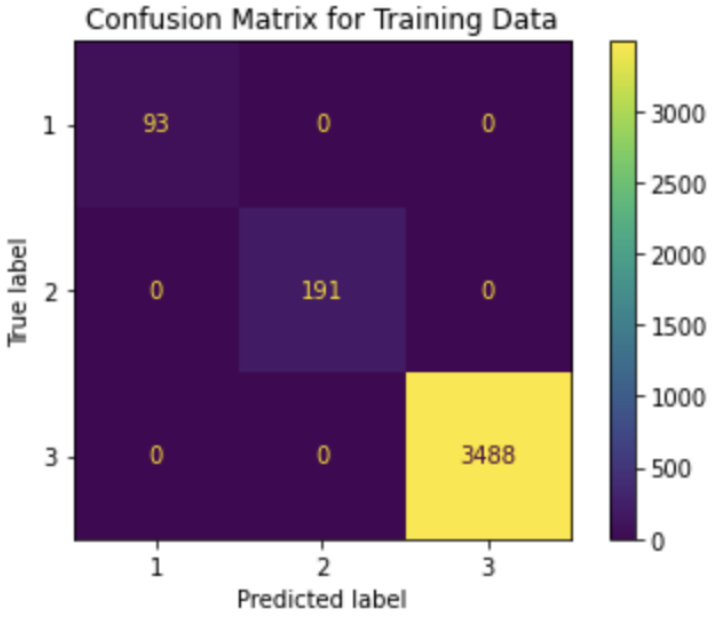
[5] M. Tan, "Cost-sensitive learning of classification knowledge and its applications in robotics," *Machine Learning, 13, 7-33*, 1993.

[6] C. G. Demir, *"decision trees - cs 550: Machine learning"*. [Online]. Available: http://www.cs.bilkent.edu.tr/%E2%88%BCgunduz/teaching/cs550/documents/CS550%20DecisionTrees.pdf.

VI. Appendix

Fig. 1. Base Case with Entropy Impurity [sklearn]

```
Training Accuracy: 100.000%
Class-Based Accuracies:
Accuracy of Class 1: 100.000%
Accuracy of Class 2: 100.000%
Accuracy of Class 3: 100.000%
```




```
Test Accuracy: 99.271%
Class-Based Accuracies:
Accuracy of Class 1: 89.610%
Accuracy of Class 2: 94.086%
Accuracy of Class 3: 99.810%
```

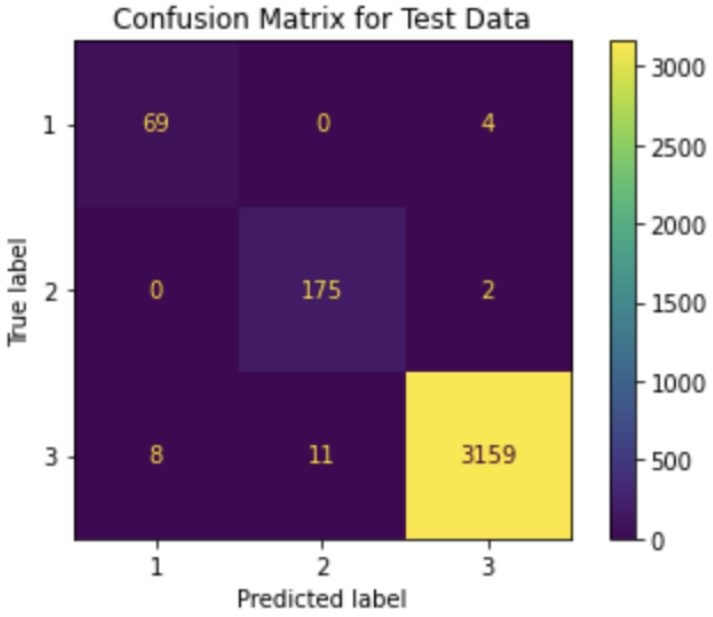


Fig. 2. Classification Accuracies and Confusion Matrices for Training and Test Data for the Tree in Previous Figure

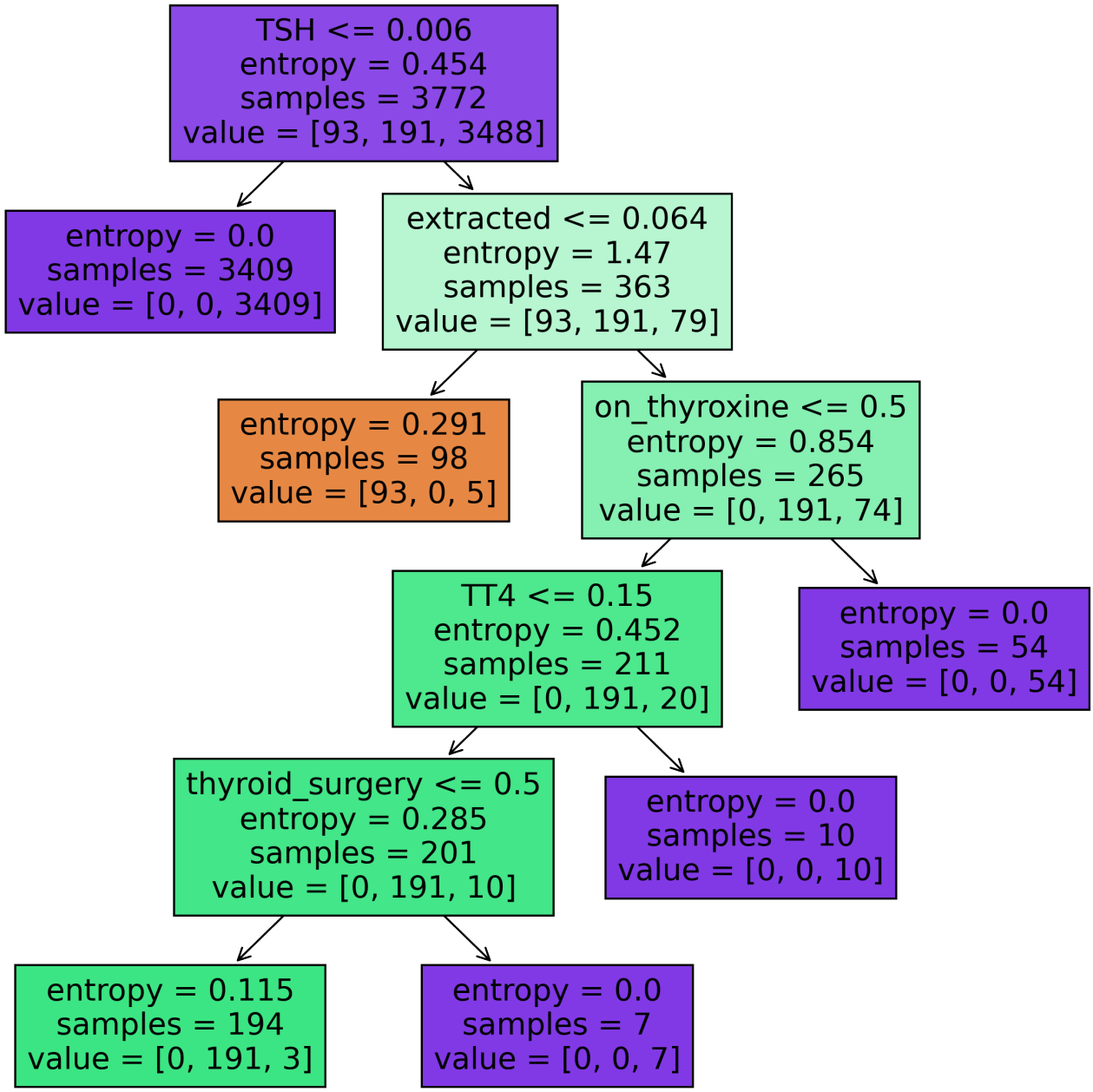
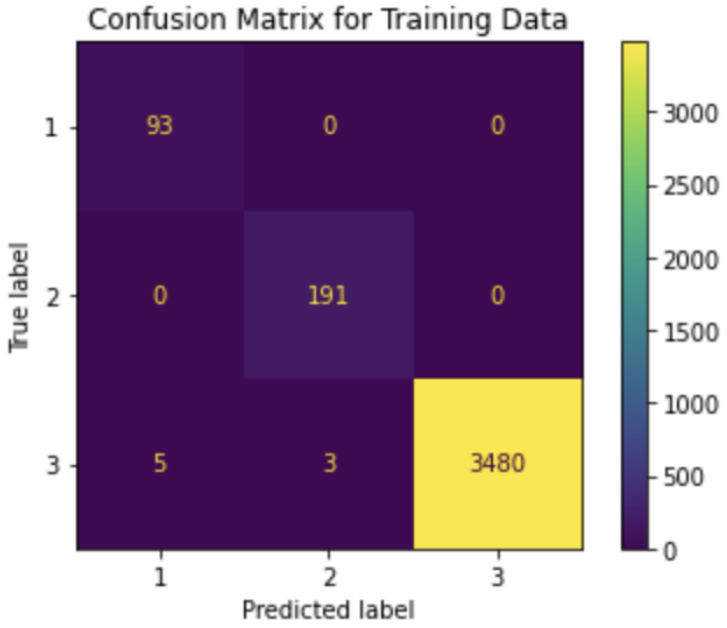


Fig. 3. Tree with Min Impurity Decrease Pruning (0.005))[sklearn]

```
Training Accuracy: 99.788%
Class-Based Accuracies:
Accuracy of Class 1: 94.898%
Accuracy of Class 2: 98.454%
Accuracy of Class 3: 100.000%
```




```
Test Accuracy: 99.358%
Class-Based Accuracies:
Accuracy of Class 1: 86.905%
Accuracy of Class 2: 94.149%
Accuracy of Class 3: 100.000%
```

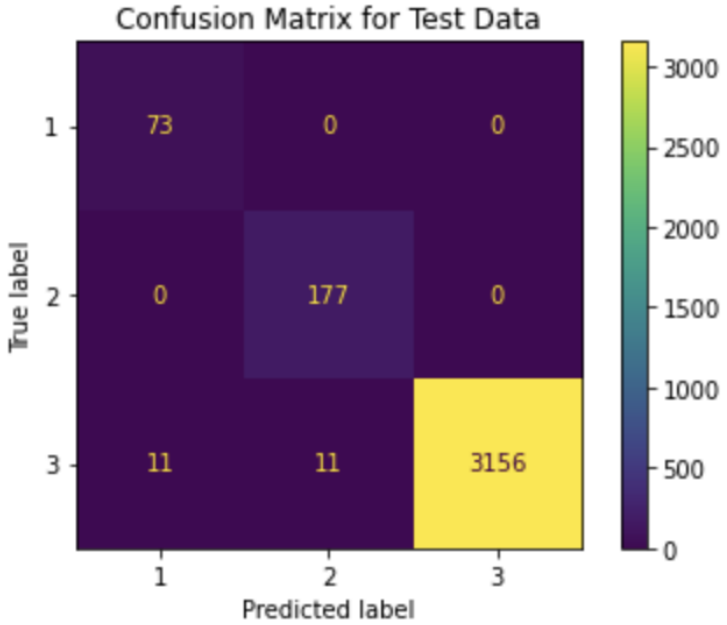


Fig. 4. Classification Accuracies and Confusion Matrices for Training and Test Data for the Tree in Previous Figure

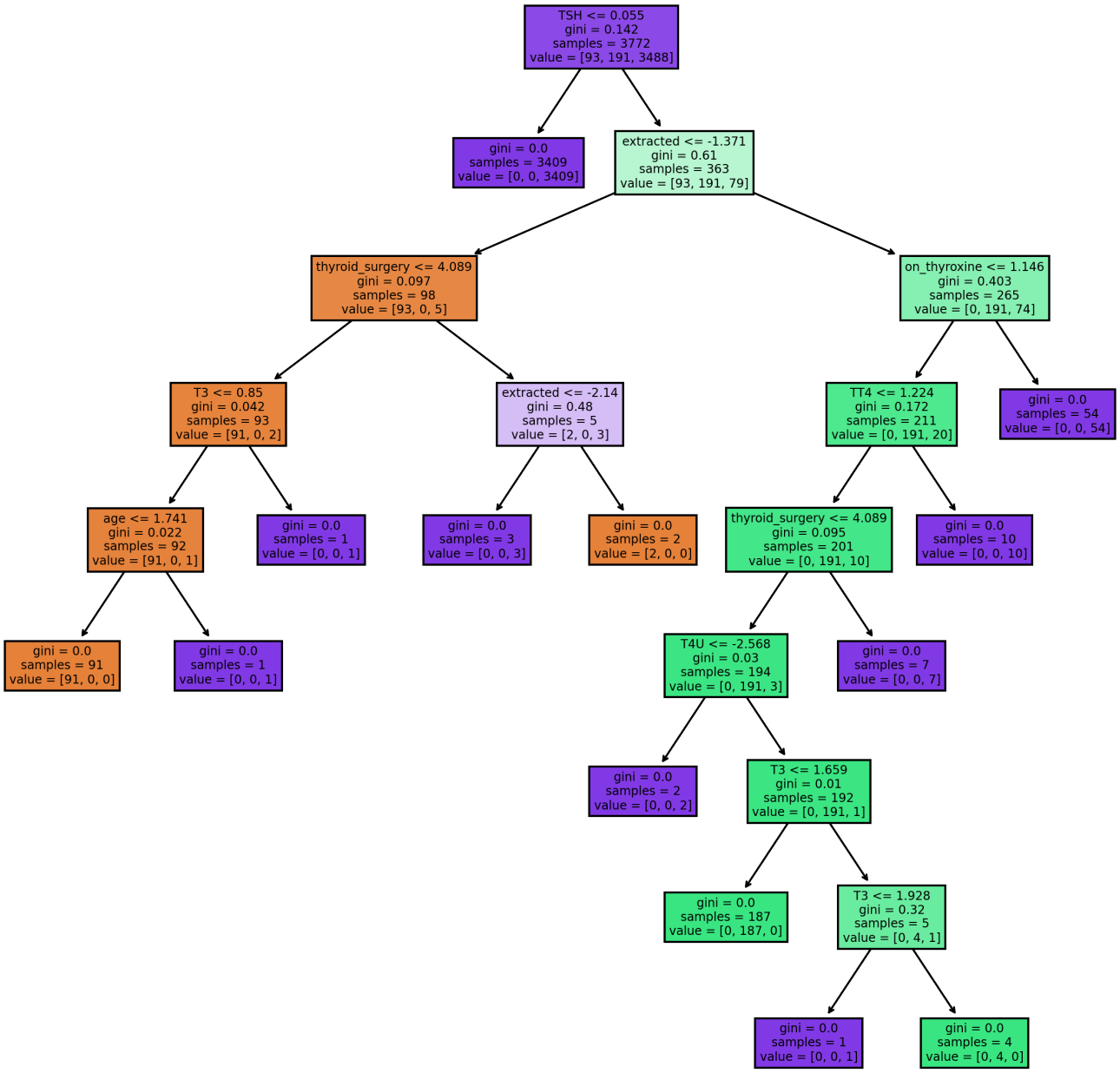
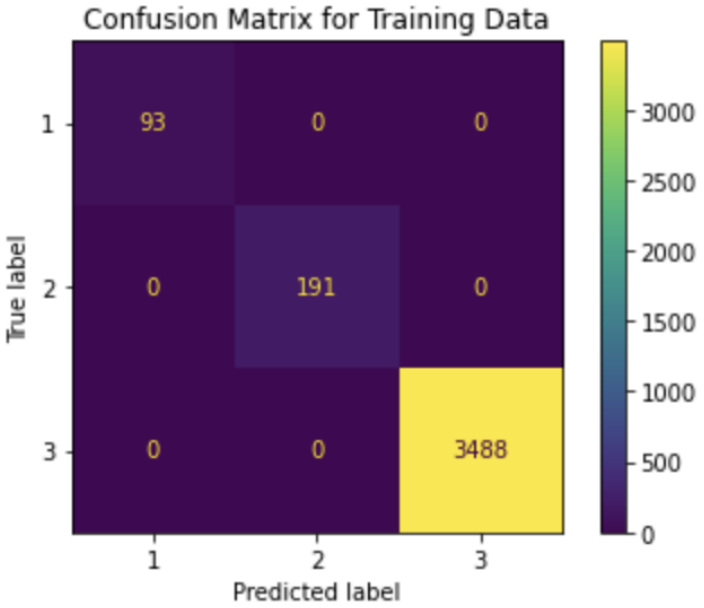


Fig. 5. Tree with Normalized Values [sklearn]

```
Training Accuracy: 100.000%
Class-Based Accuracies:
Accuracy of Class 1: 100.000%
Accuracy of Class 2: 100.000%
Accuracy of Class 3: 100.000%
```




```
Test Accuracy: 99.154%
Class-Based Accuracies:
Accuracy of Class 1: 90.141%
Accuracy of Class 2: 94.086%
Accuracy of Class 3: 99.653%
```

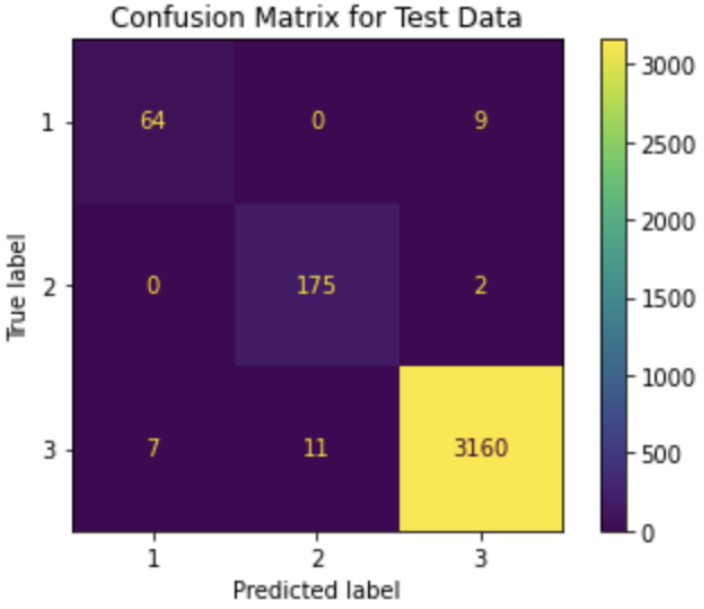


Fig. 6. Classification Accuracies and Confusion Matrices for Training and Test Data for the Tree in Previous Figure

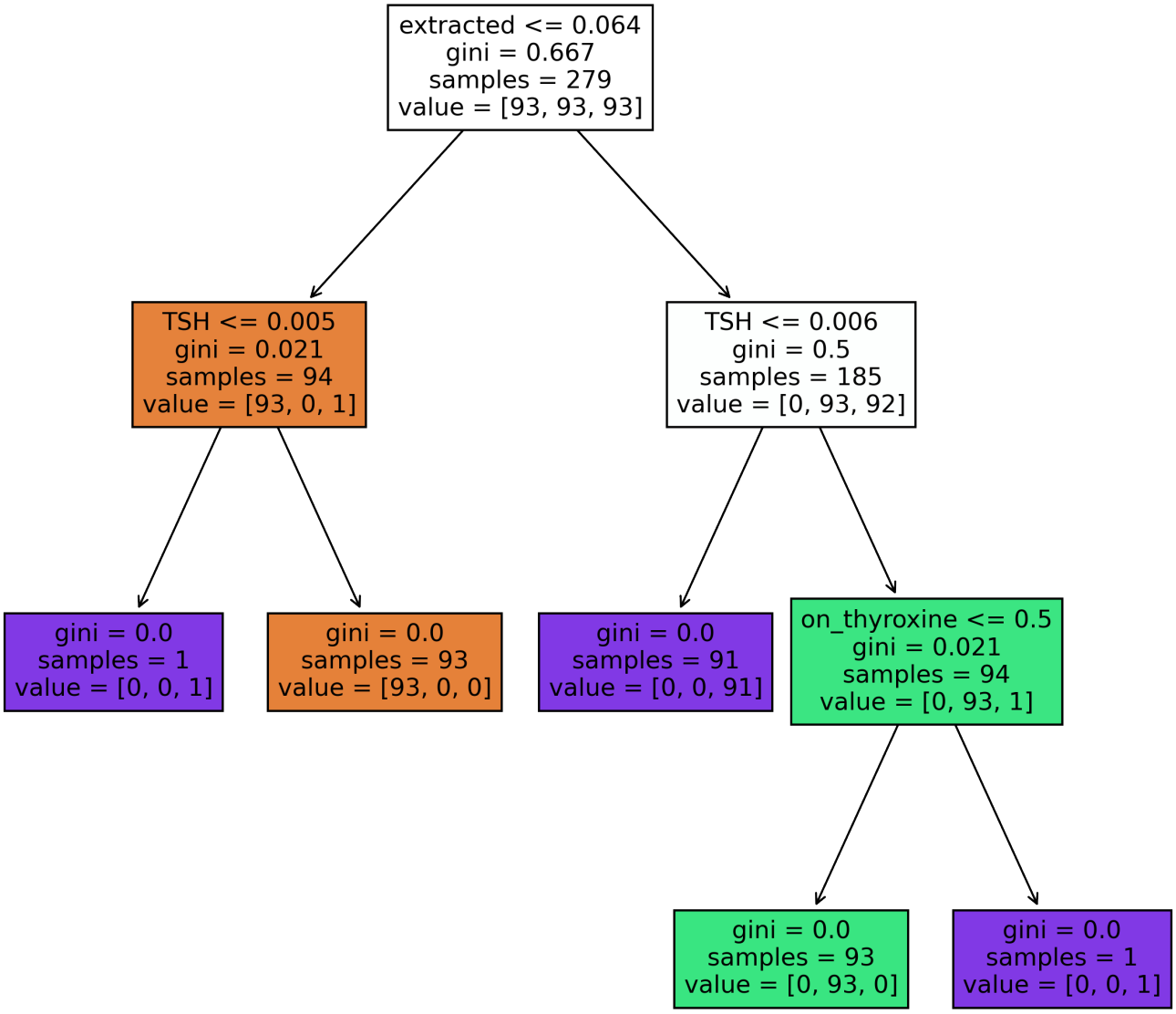
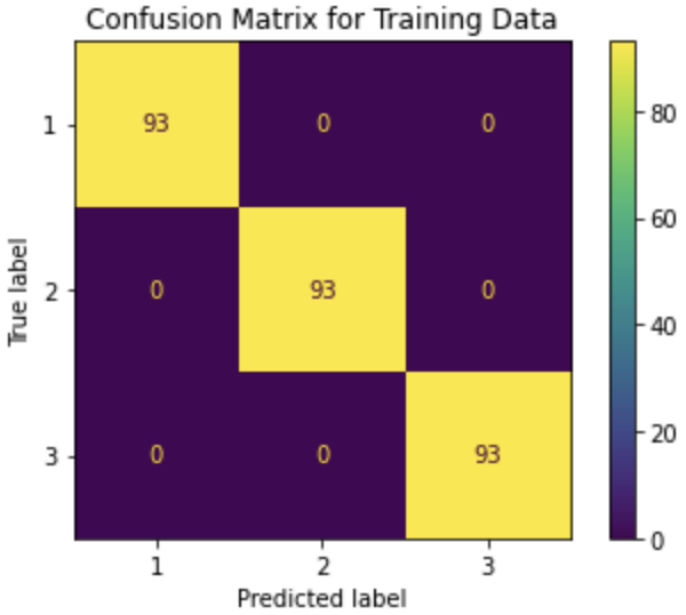


Fig. 7. Tree with the Subsampled Data [sklearn]

```
Training Accuracy: 100.000%
Class-Based Accuracies:
Accuracy of Class 1: 100.000%
Accuracy of Class 2: 100.000%
Accuracy of Class 3: 100.000%
```




```
Test Accuracy: 98.191%
Class-Based Accuracies:
Accuracy of Class 1: 81.111%
Accuracy of Class 2: 79.730%
Accuracy of Class 3: 100.000%
```

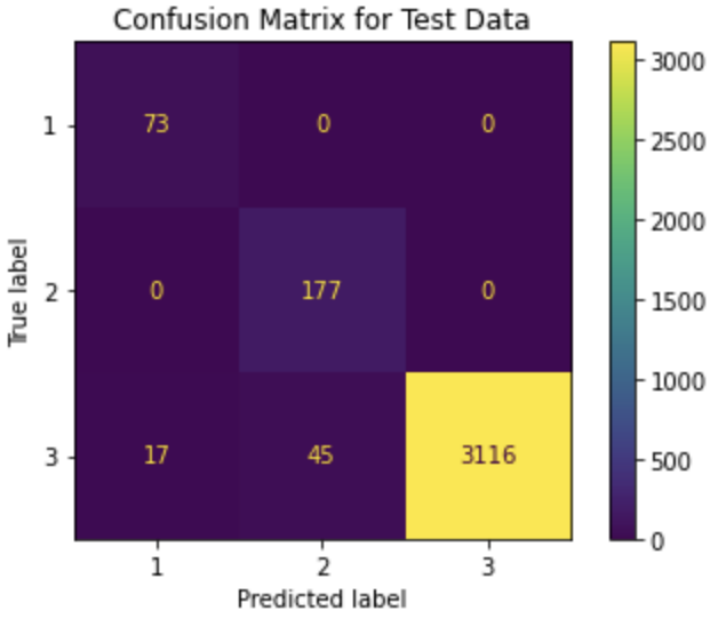


Fig. 8. Classification Accuracies and Confusion Matrices for Training and Test Data for the Tree in Previous Figure

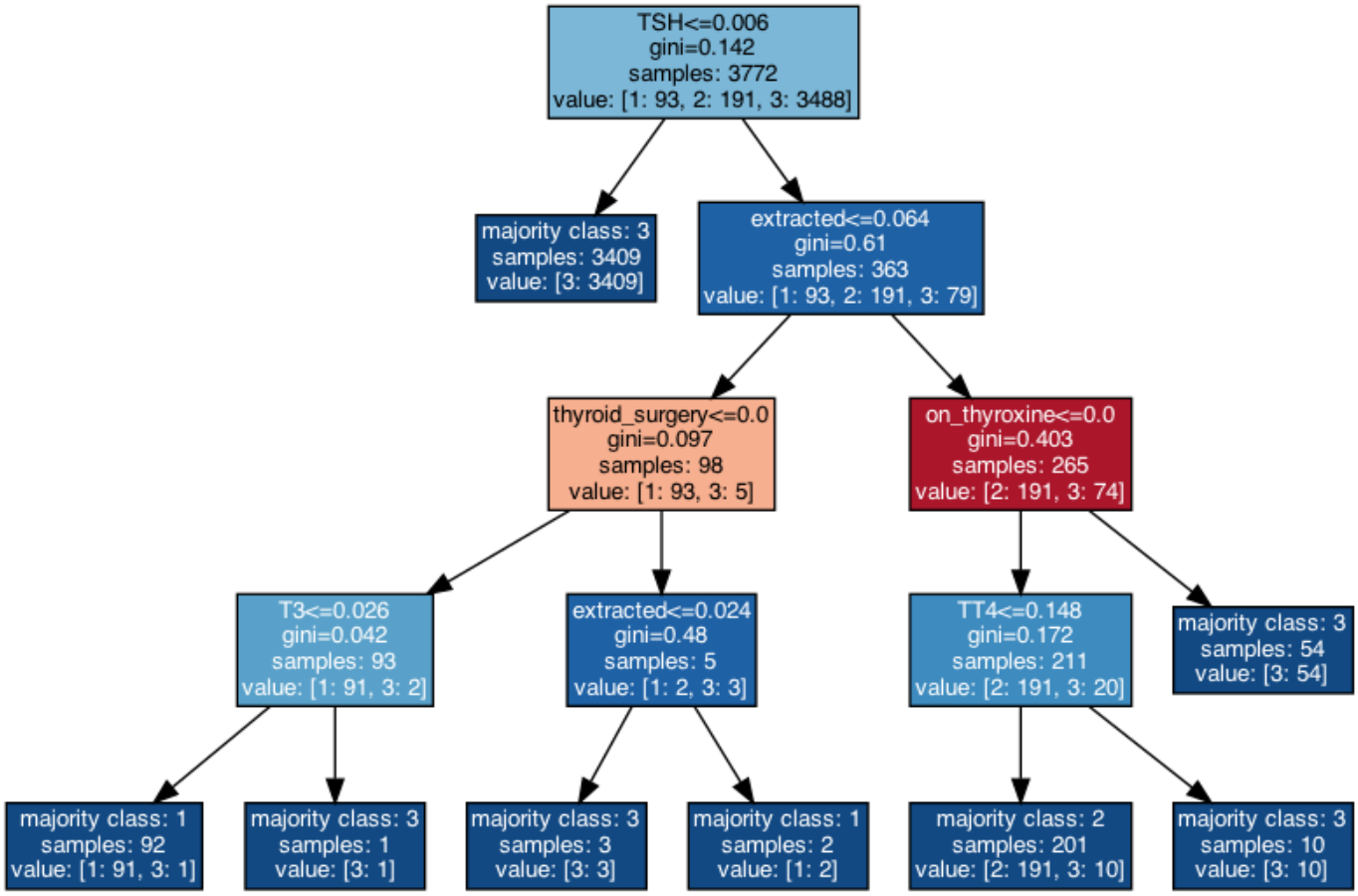
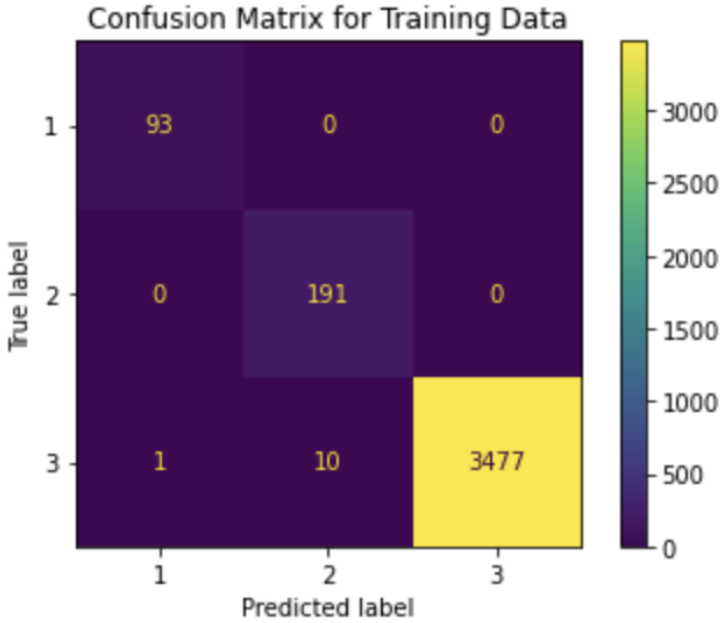


Fig. 9. Tree with Gini Impurity Pruned with Depth = 4

```
Training Accuracy: 99.708%
Class-Based Accuracies:
Accuracy of Class 1: 98.936%
Accuracy of Class 2: 95.025%
Accuracy of Class 3: 100.000%
```




```
Test Accuracy: 99.271%
Class-Based Accuracies:
Accuracy of Class 1: 91.026%
Accuracy of Class 2: 92.147%
Accuracy of Class 3: 99.905%
```

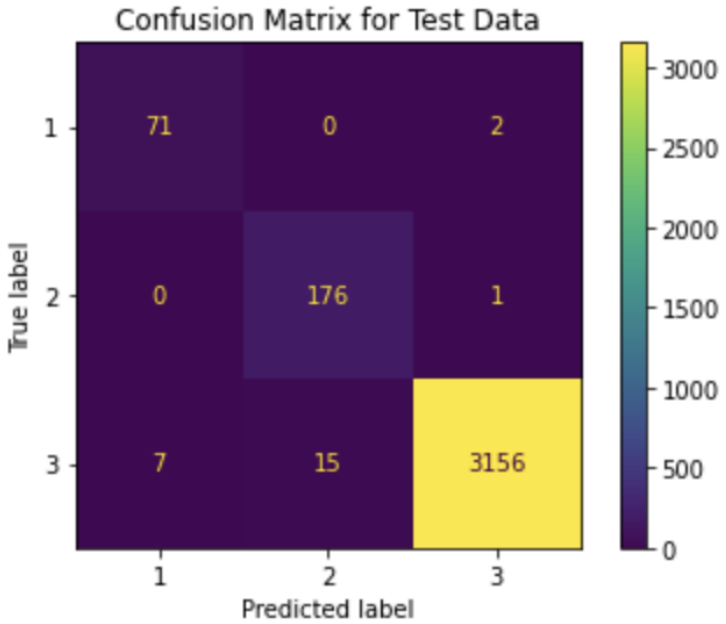


Fig. 10. Classification Accuracies and Confusion Matrices for Training and Test Data for the Tree in Previous Figure

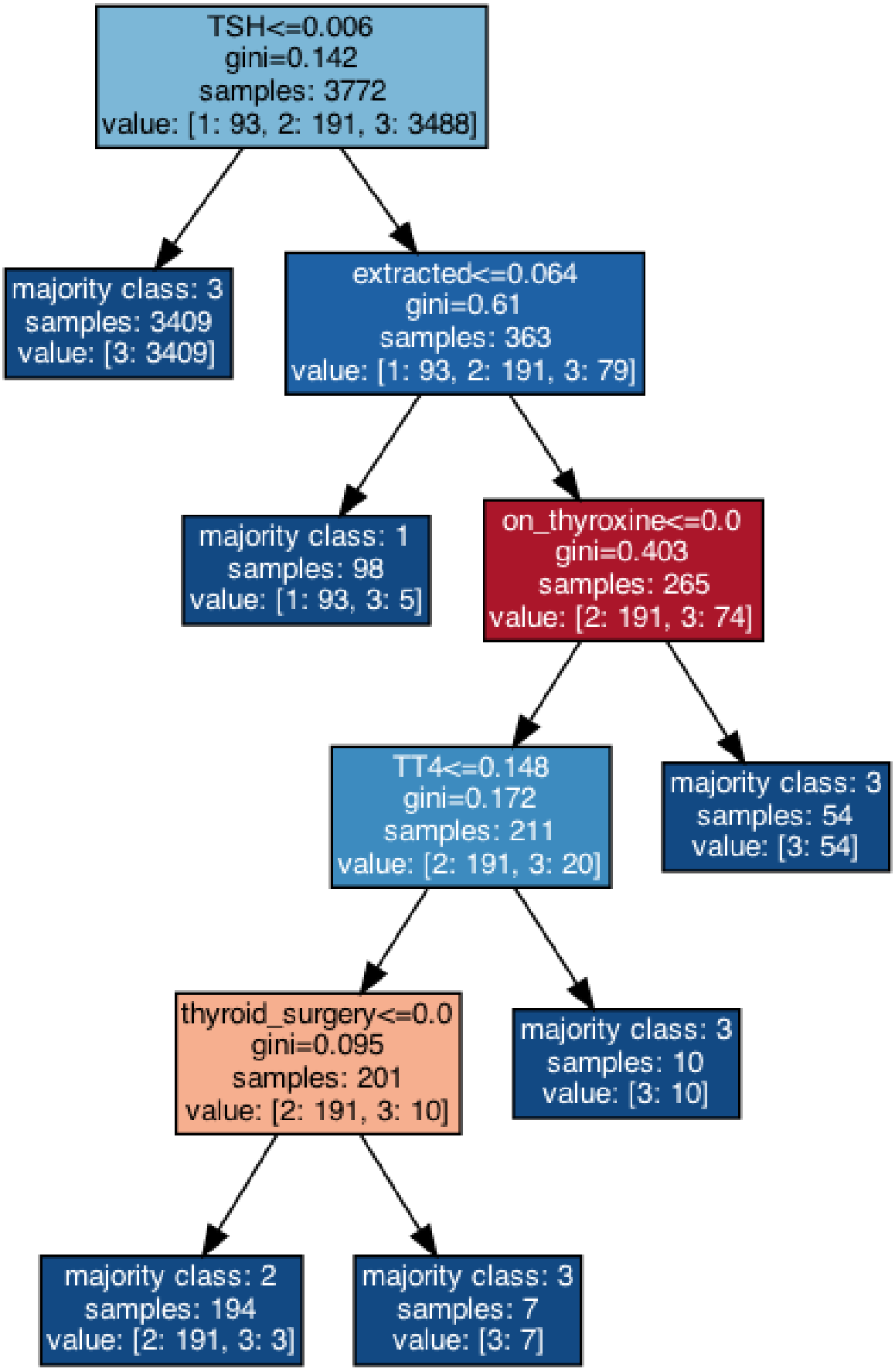
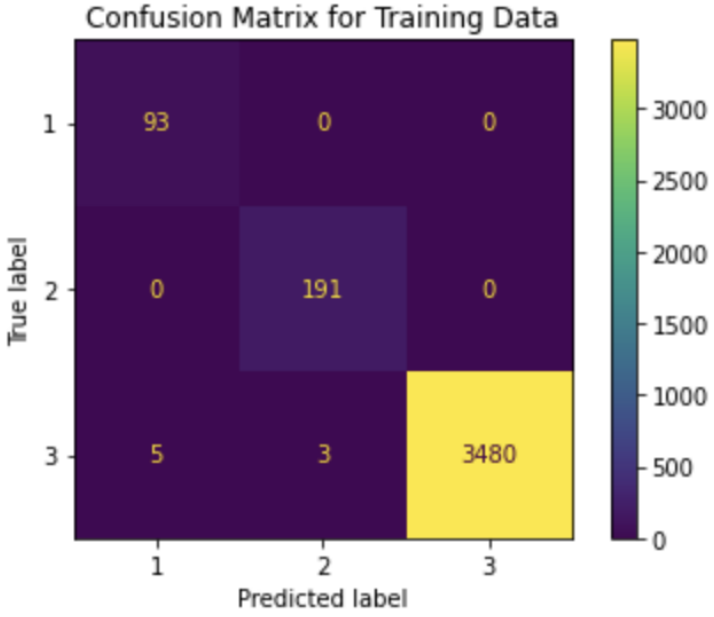


Fig. 11. Tree with Gini Impurity Pruned with Gain = 0.05

```
Training Accuracy: 99.788%
Class-Based Accuracies:
Accuracy of Class 1: 94.898%
Accuracy of Class 2: 98.454%
Accuracy of Class 3: 100.000%
```




```
Test Accuracy: 99.329%
Class-Based Accuracies:
Accuracy of Class 1: 86.905%
Accuracy of Class 2: 94.118%
Accuracy of Class 3: 99.968%
```

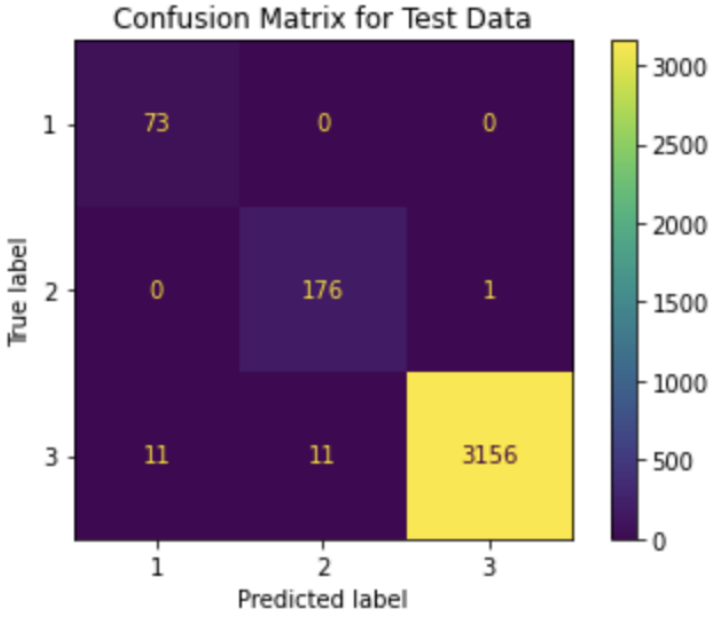


Fig. 12. Classification Accuracies and Confusion Matrices for Training and Test Data for the Tree in Previous Figure

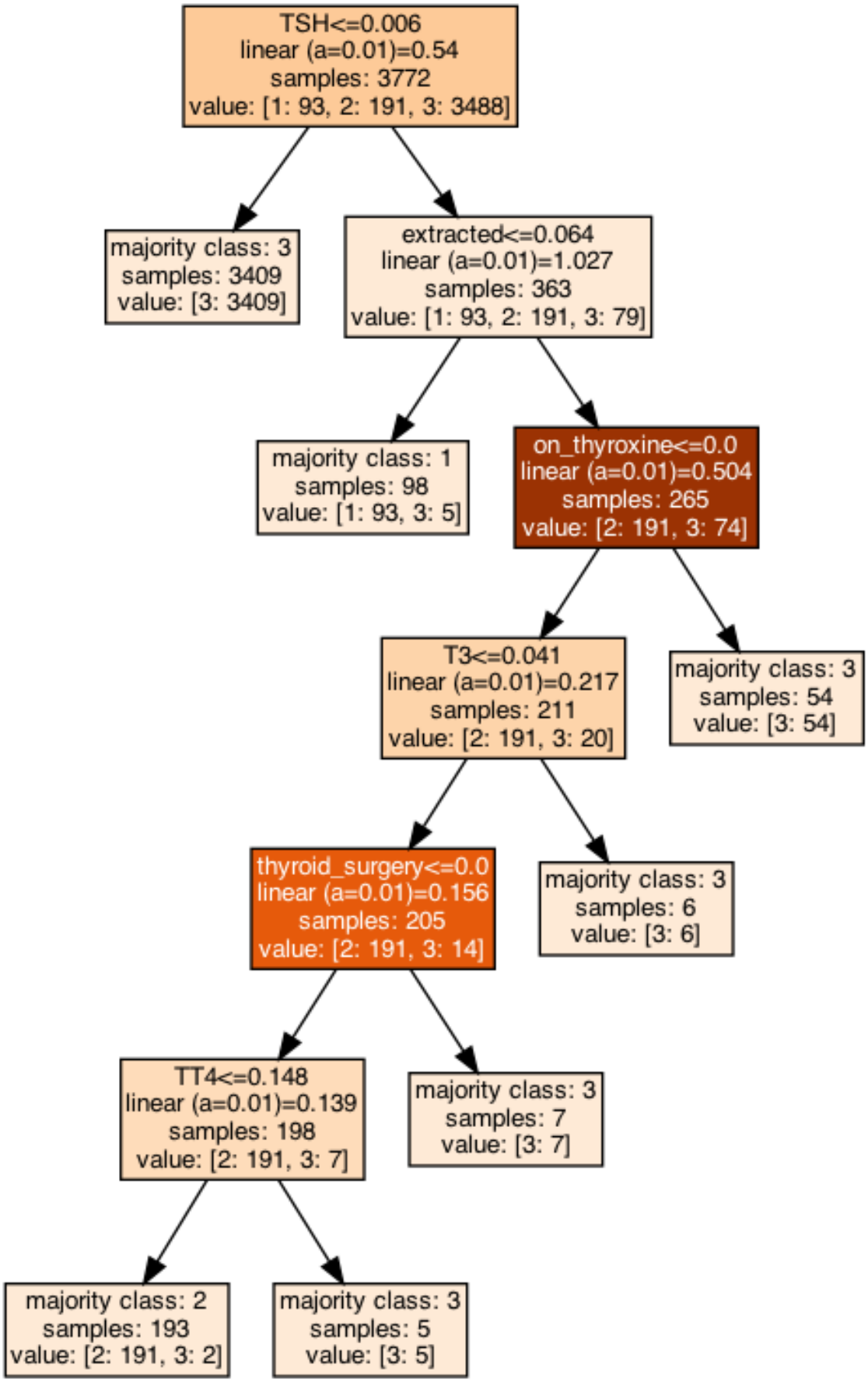
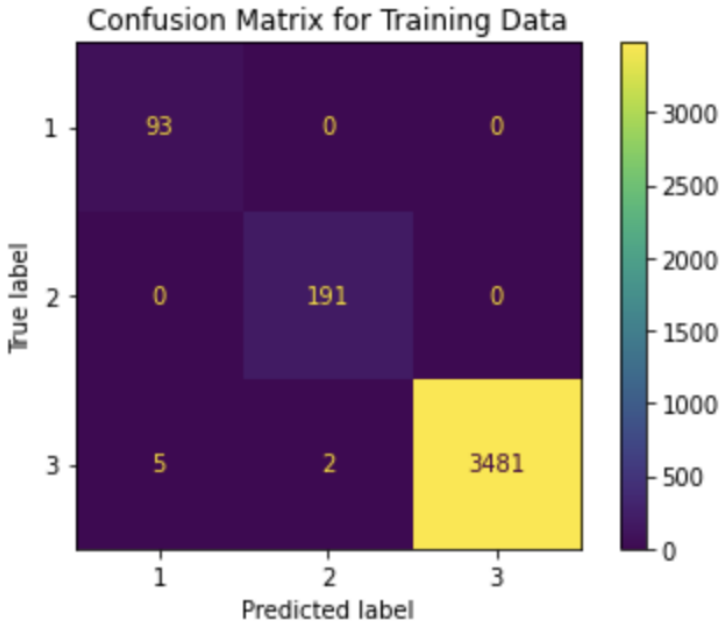


Fig. 13. Tree with Linear Splitting Criterion ($\alpha = 0.01$) with Information Gain Pruning 0.1 bits

```
Training Accuracy: 99.814%
Class-Based Accuracies:
Accuracy of Class 1: 94.898%
Accuracy of Class 2: 98.964%
Accuracy of Class 3: 100.000%
```




```
Test Accuracy: 99.329%
Class-Based Accuracies:
Accuracy of Class 1: 86.905%
Accuracy of Class 2: 94.118%
Accuracy of Class 3: 99.968%
```

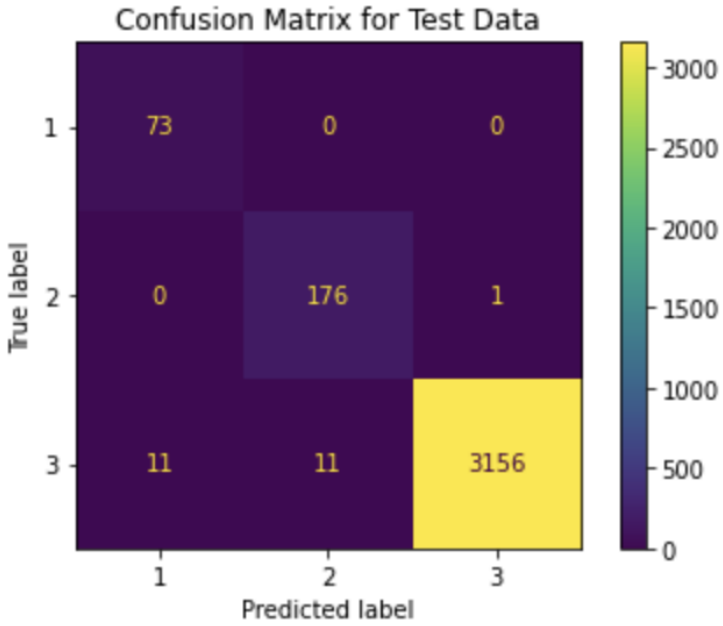


Fig. 14. Classification Accuracies and Confusion Matrices for Training and Test Data for the Tree in Previous Figure

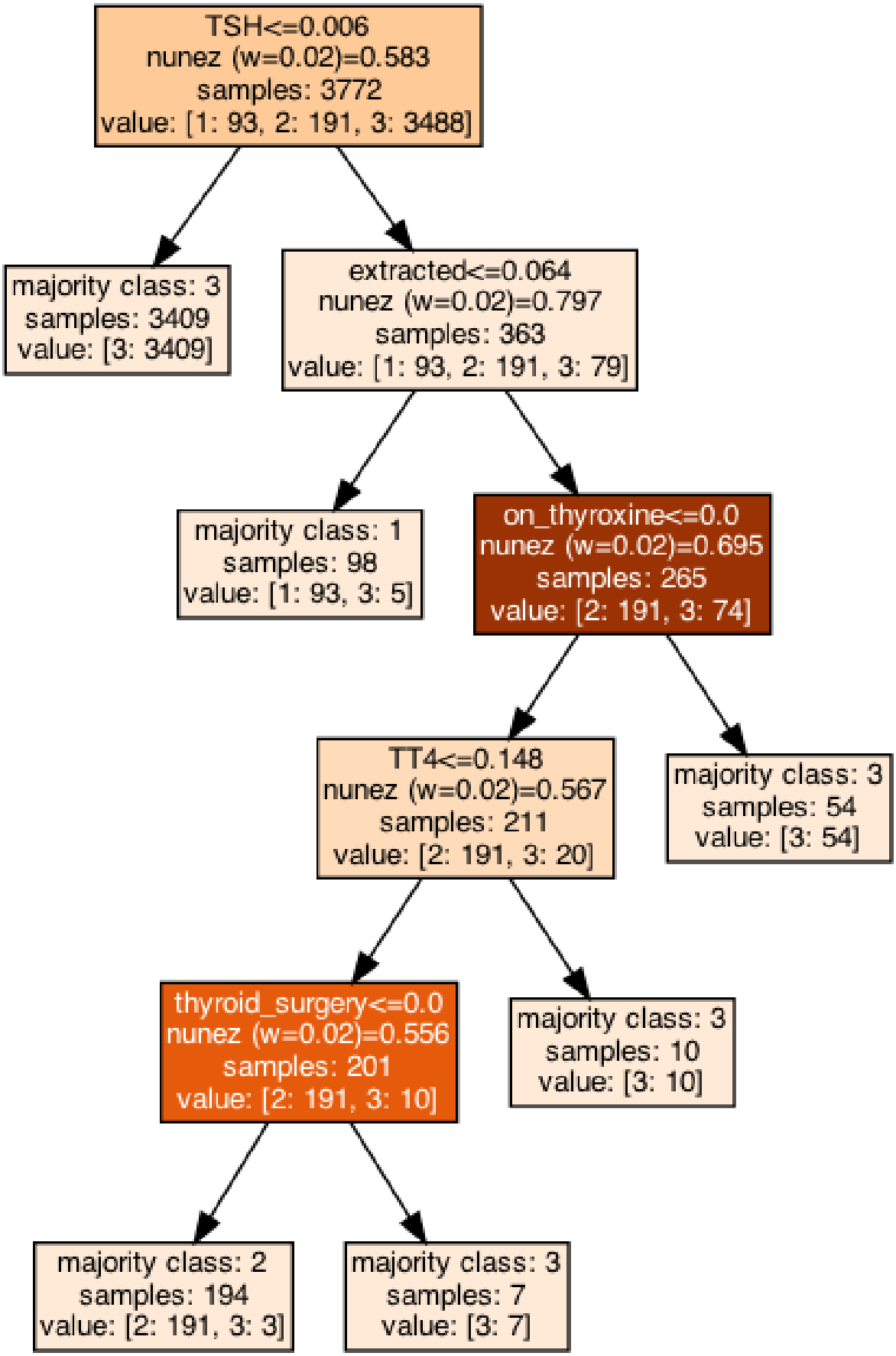
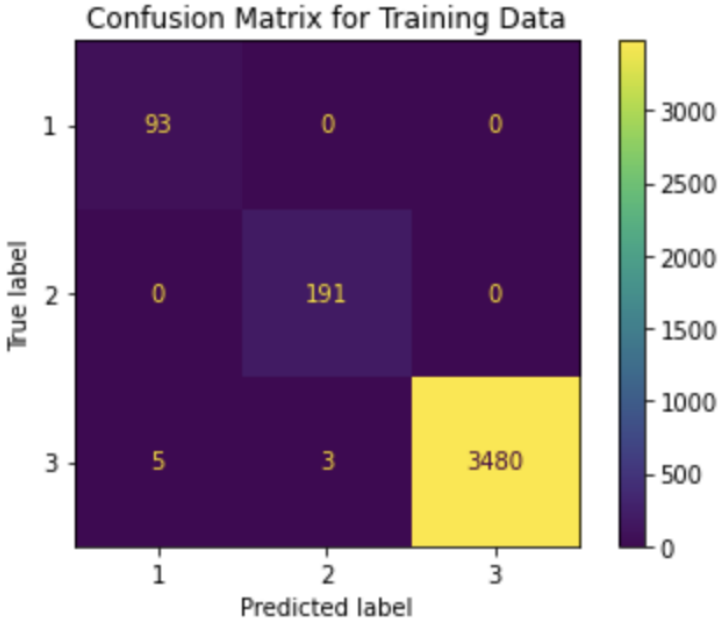


Fig. 15. Tree with Nunez [4] Splitting Criterion (= 0.02) with Information Gain Pruning 0.1 bits

```
Training Accuracy: 99.788%
Class-Based Accuracies:
Accuracy of Class 1: 94.898%
Accuracy of Class 2: 98.454%
Accuracy of Class 3: 100.000%
```




```
Test Accuracy: 99.329%
Class-Based Accuracies:
Accuracy of Class 1: 86.905%
Accuracy of Class 2: 94.118%
Accuracy of Class 3: 99.968%
```

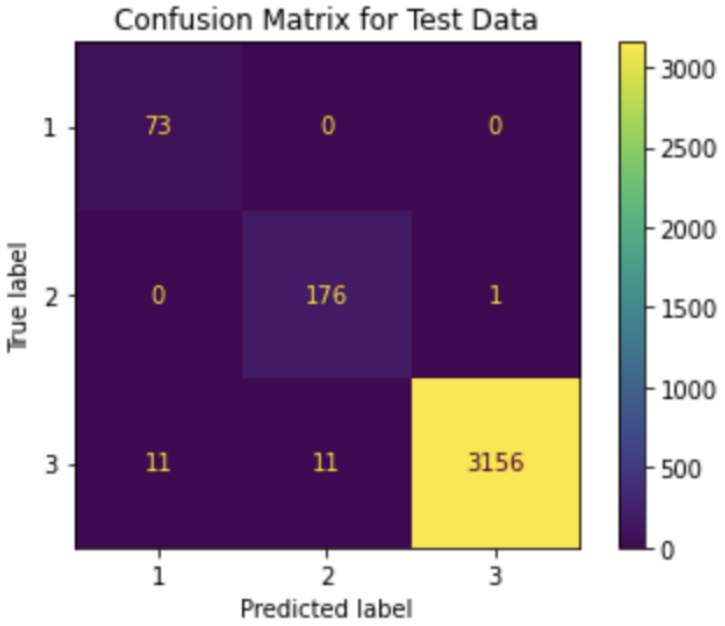


Fig. 16. Classification Accuracies and Confusion Matrices for Training and Test Data for the Tree in Previous Figure