Homework # 1

CS 550 - Machine Learning

Yusuf Dalva, Bilkent ID: 21602867
yusuf.dalva@bilkent.edu.tr

Introduction

In this assignment, the decision tree classification algorithm has been inspected in detail, including its different variations. Here, at the first stage a toolbox has been used both as a baseline model for personal implementation and to get stable results that is tested by several individuals. Considering the programming language used for this assignment, python, scikit-learn package has been used. For the remaining discussion on decision trees on Part 2 and Part 3, a personal implementation for decision trees has been used. Explanatory details about the implementation are given in the corresponding sections. In this assignment, to understand the given dataset better, a data exploration stage has been performed to observe the class imbalance issues. This part introduces the data management module implemented and explains the observed trends in the dataset. Figures that are provided in his report can also be found on Appendix in full size, for clarity purposes. Sections are numbered in a way that is consistent with the part numbers in assignment description (Section numbered as I corresponds to Part 1 etc.).

Data Exploration

Before the implementation of the algorithm, a data exploration stage has been performed. At the first stage, the data files that are given imported with the help of a helper class named **Dataset**. This class is a personal implementation and can be found as the file *dataset.py*. As stated in the assignment description, the $1^{st}, 17^{th}, 18^{th}, 19^{th}, 20^{th}, 21^{st}$ features are observed as continuous and the other features are observed as binary. As the dataset has been observed, it is identified that the dataset is applicable for feature-wise decision boundaries with specific features, this can be seen on the implementation file. Following this step, the class imbalance problem has been addressed. This stage uses the frequencies of different classes in the dataset. It is observed that training and test sets show similar qualities in terms of label (state of nature or class) distribution. The observations can be summarized as follows:

1) Class 1: 2.47% of training set, 2.13% of test set
2) Class 2: 5.06% of training set, 5.16% of test set
3) Class 3: 92.47% of training set, 92.71% of test set

The implementation details of this analysis can be found at IPython notebook file named *Data_Exploartion.ipynb*.

I. Using scikit-learn package for training a classifier

As specified at the previous parts, for the first part of the assignment DecisionTreeClassifier class has been used. Since the library provides options for easy configuration and visualization, this option has been followed.

A. *Finding the Best Classifier*

In order to find the best decision tree possible for the given training and test sets, grid search methodology has been used. The main motivation behind this approach is to achieve the optimal set of parameters, rather than making a greedy choice in parameter selection. In this parameter selection step, the parameters that are optimized are given as a list below:

- Splitting Criterion: Gini impurity or entropy impurity
- Splitting type: Best split or random split
- Impurity limit at split: The split will be performed only if the impurity value is above a certain threshold
- Maximum tree length: The limit for the height of the tree

Possible values for the variables are tried out in a four dimensional grid and then best parameters are reported in the implementation. Here, in order to determine a tree length limit, first the tree has been grown without any pruning and then the possible values for the tree length has been determined. Also for the impurity limit at split, the limit has been selected as a number in the interval [0,1]. With all of these different selections (as a grid search), the optimum point has been found to obtain the optimal classifier for the given test set. By trying out all possible combinations, the choice has been made more like a dynamic programming approach rather than a greedy approach. Lastly, to determine the scores of the classifiers for evaluation class weighted accuracy has been used to deal with class imbalance [1]. Here, the class weights are inversely proportional with the frequencies of the classes in the corresponding dataset. The equations for this calculation is as follows:

$$acc_{balanced} = \sum_{c=1}^{C} \omega_c \times acc_c \, , \sum_{c=1}^{C} \omega_c = 1 \tag{1}$$

Here the weights are determined in a way such that they are inversely proportional with the frequency of the class in the corresponding dataset. This modification ensures that each of the classes make equal contribution to the calculation of the predictive score. In equation 1, acc represent the accuracy score calculated. The algorithm for this weighting operation is given as Algorithm 1. The results of this grid search together with the values tried out are given below:

- Splitting Criterion: ”entropy” is selected among (”entropy”, ”gini”)
- Splitting Type: ”best” is selected among (”best”, ”random”)

Algorithm 1: Class Weighted Accuracy

```
$freq \leftarrow$ Frequencies of each class in dataset
for $c \leftarrow 1$ to $C$ do
    $\omega_c = \prod_{i \neq c} freq_i$
$\omega \leftarrow \omega \div \sum_{c=1}^{C} \omega_c$
```

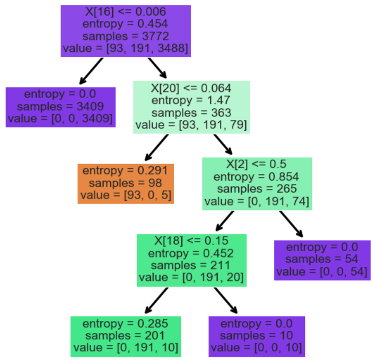

Fig. 1. Decision Tree plotted with best parameters for part 1

- Impurity limit at split: 0.01 is selected among (0, 0.01, 0.05, 0.2, 0.5, 0.7)
- Maximum tree length (depth): 4 is selected among (1, 2, 3, 4, 5, 6, 7, 8, 9)

The resulting tree is shown at Fig. 1 and the confusion matrix for training set and test set is shown at Fig. 2.

According to the confusion matrices given in Fig. 2, the accuracy values are as follows:

- Overall Accuracy: 99.24% for test set and 99.60% for training set
- Class 1 Accuracy: 100% in both training and test sets
- Class 2 Accuracy: 100% in both training and test sets
- Class 3 Accuracy: 99.18% for test set and 99.57% for training set

With the effect of pruning while constructing the classifier, it was possible to get near to completely consistent results while evaluating training set and testing sets. Here the prepruning algorithm has two main hyper-parameters which is optimized at the grid search. These parameters were impurity limit at split and maximum tree depth. As specified before, the optimal parameter for this prepruning algorithm were 4 for maximum tree depth and 0.01 for minimum impurity increase (no split if below).

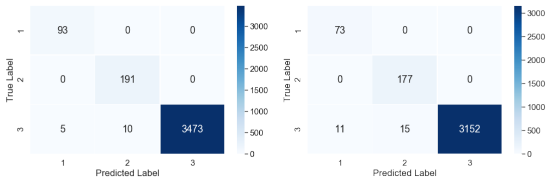

Fig. 2. Confusion Matrix for training (left) and test sets(right)

B. Decision Tree without pruning

For the case of where no prepruning is applied, the training and test set accuracies are given below:

- Overall accuracy: 100% on training set, 99.32% on test set
- Class 1 accuracy: 100% on training set, 94.52% on test set
- Class 2 accuracy: 100% on training set and test set
- Class 3 accuracy: 100% on training set, 99.40% on test set

When the results when prepruning and no pruning have been applied, the effect of overfitting is clear when the training set accuracies are observed. As the training continues until all leaves are pure when these is no pruning, the decision tree constructed overfits the training set. This finding is validated by comparing the training set accuracies where it is 100% at the overfitting case (no pruning). However, since the linear separability of the training set and test set are very similar, even the overfitting case gave high performance, which is assessed in terms of accuracy. Even though the performance drop is small, it can be observed that the decision tree that is constructed with prepruning performs better than the overfitting tree, which involves no pruning.

C. Normalizing feature values

To inspect the effects of feature normalization, a standard score normalization has been performed. This normalization is performed on the training samples and test samples. Formulation is given as follows:

$$X = \frac{X - \mu_{train}}{\sigma_{train}} \tag{2}$$

Here, in order to perform a consistent normalization that maps the features in a way that their mean will be 0 and standard deviation will be 1. After evaluating the results on the decision tree where the feature values are normalized, it is observed that the confusion matrix is identical with the one for the optimal decision tree. In order to focus only on the effect of normalization, the comparison is performed on the decision tree with optimal hyper-parameters. The only difference is observed on the training set, which is minor. The results for the training set and test set are as follows:

- Overall Accuracy: 99.24% for test set and 99.60% for training set
- Class 1 Accuracy: 100% in both training and test sets
- Class 2 Accuracy: 100% in both training and test sets
- Class 3 Accuracy: 99.18% for test set and 99.57% for training set

These results effectively show that the normalization does not have any effect in terms of testing performance as expected. Since the splits are found in a way that tries out different alternatives, normalized features will only form different possible splits without changing the separation of the data. Due to this fact and these findings, it is concluded that data normalization makes no difference in terms of model performance for a decision tree model.

D. Testing With Balanced Classes

In order to deal with the class imbalance problem and performing a test with same number of samples for each class, under-sampled is the used methodology here. In order to conduct an experiment, first the samples are shuffled. Then from each class samples are drawn without replacement from the dataset in the amount of the number of samples for the minority class, which is the number of instances for class 1 in this assignment. This operation is performed for selecting the training samples to conduct the decision tree model with optimal hyper-parameters. The results of training and testing accuracies are given as follows:

- Class 1 Accuracy: 100% on both training and test set
- Class 2 Accuracy: 100% on both training and test set
- Class 3 Accuracy: 100% on training set and 98.17% at the test set

As the obtained results shows, the predictive performance of the model with the optimal hyper-parameters for the whole data, benefited the predictions made for Class 1 and Class 2. However, the majority class where only a small proportion of the samples are used showed poor performance. Even though the tree depth is kept constant after sampling the data, it seems that the sampled dataset shows more basic characteristics that was not able to capture all the details in the training set (for Class 3). Due to this fact, a drop in the accuracy metric for class 3 is observed. Even though this drop is around 1% in terms of accuracy, considering the amount of test samples available, the decrease in the generalization of the environment when under-sampling is used is significant. The change in the confusion matrix is also significant for class 3 (missing approximately 30 more samples from class 3). These findings can be explained with the lower level of details identified with the under-sampled dataset.

All of the demonstrations for the first part can be found at file "*Part 1 - Decision Tree With A Toolbox.ipynb*"

II. Implementing a Decision Tree with Prepruning

Following the instructions provided, for this part a decision tree classifier has been implemented originated from [2] and [3]. Just like in the data exploration stage, the implementation has been performed on a separate file named $decision\_tree.py$. Following the binary tree approach, this implementation is composed of a tree and its corresponding nodes.

A. Expressing A Decision Tree Mathematically

In order to implement the decision tree fro scratch, the first step was determining the mathematical methodology behind the classifier. In order to do that firstly the two impurity measures used are formulated.

$$impurity_{entropy} = \sum_{c=1}^{C} \hat{P}(Y = C_c|x) \times log\hat{P}(Y = C_c|x) \tag{3}$$

$$impurity_{gini} = 1 - \sum_{c=1}^{C} \hat{P}(Y = C_c|x)^2 \tag{4}$$

Then for a selected impurity measure, the best split for a given data distribution will be searched. If the optimal parameters are defined as i(feature which the split will be applied), t(threshold value for the split), the optimal split can be identified as follows:

$$\begin{aligned} \arg\min_{i,t} \hat{P}(left) * impurity(left) \\ + \hat{P}(right) * impurity(right) \end{aligned} \tag{5}$$

Here left indicates the left node and right indicates the right node. The variable that is minimized is the entropy of a split which weights the entropy values of the nodes. The weighting is performed by using a probability estimation here. Considering that the samples in n^{th} node are shown as D_n and the samples included in the left child and right child according to the splitting rule $x_i \leq t$ (i is the splitting feature and t is the split threshold) are $D_{n,left}$ and $D_{n,right}$ are estimated with the following estimator:

$$\hat{P}(left) = \frac{D_{n,left}}{D_n}, \hat{P}(right) = \frac{D_{n,right}}{D_n} \tag{6}$$

Here $D_{n,left}$ includes the samples satisfying $x_i \leq t$ and $D_{n,right}$ includes the samples satisfying $x_i > t$. Considering the introduced theory, there are various hyper-parameters in model construction and the main task here after implementation is to find the optimal decision tree trained on the training data that performs best on the training set.

B. Implementation Details

The decision tree implemented is designed as a univariate tree and also implements a prepruning strategy. The decision tree classifier implemented gives several options for the individual for performing pruning and changing how the decision tree operates. The provided choices are as follows:

- Splitting Criterion: As a splitting criterion either entropy impurity or gini impurity can be used, just like the classifier using scikit-learn package.
- Prune Length: Considering the level of the root node as level 0, the decision tree applies prepruning by not allowing any node that has depth than the specified prune length parameter.
- Prune Threshold: As another idea adapted from the decision tree classifier used from scikit-learn package, as another prepruning strategy the algorithm considers increase in score. Here if the difference of evaluation scores (impurity measure) before and after the split are smaller than the specified threshold, the tree is pruned.

For validation of the implemented decision tree algorithm, the fully grown trees by the toolbox and the personal implementation compared and found as nearly identical. The only difference is caused by selecting features in a random order in the toolbox, whereas the provided implementation does it sequentially.

C. Finding the optimal hyper-parameters

Just like the one performed on the first part of the assignment, the optimal hyper-parameters for the classifier are also

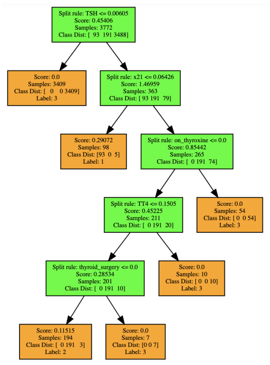

Fig. 3. Decision Tree Constructed for Part 2

searched by a grid search approach for not performing a greedy selection while selecting hyper-parameters. For assessment of each classifier, the weighted accuracy approach is followed which is provided in Algorithm 1. The optimal hyper parameters is given as follows:

- Splitting Criterion: The best splitting criterion is selected as entropy among ("entropy", "gini")
- Prune Length: The prune length is selected as 5 among (1, 2, 3, 4, 5, 6, 7, 8, 9)
- Prune Threshold: The pruning threshold is selected as 0.1 among (0, 0.01, 0.003, 0.1, 0.3, 0.5, 0.8, 1.0)

When the hyper-paramereters are compared with the one found for the toolbox, there is a minor difference that is identified. The main difference results from the pruning threshold that is selected. In order to examine the reason this difference, it is identified that the algorithm provided by the scikit-learn package implements this pruning by threshold strategy by weighting the impurity gain. Considering this fact the decision tree algorithm written is consistent with the one provided by scikit-learn package. Considering these hyper-parameters selected by using the grid search approach, the constructed three is given in Fig. 3. In order to visualize this tree, a tool named graphviz is used which renders trees from dot files. After drawing the decision tree, the confusion matrices and the class based accuracy values are reported. Confusion matrices are given with Fig. 4. The accuracy values are as follows:

- Overall Accuracy: 99.35% in test set and training set
- Class 1 Accuracy: 100% in test set and training set
- Class 2 Accuracy: 100% in test set and training set
- Class 3 Accuracy: 99.31% in test set and training set

As it can be observed from the training set and test set accuracies, with the prepruning technique applied, the constructed decision three does not overfit to the training samples. This observation shows the success of the prepruning algorithm applied. The high values of class based accuracies can be easily explained with the similarity of the training and testing data in terms of feature distributions. When the decision trees on Fig. 1 and 3 are compared, the consistency of the algorithm implemented can also be observed. Overall, with the results obtained for training and testing set the implemented algorithm can be considered as competitive in comparison with the one provided by scikit-learn package.

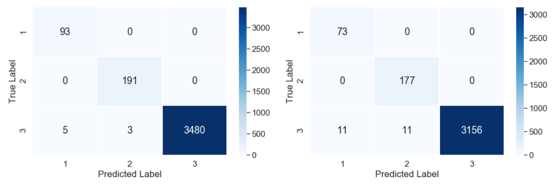

Fig. 4. Confusion Matrices for Training(left) and Test sets(right) for Part 2

The complete implementation can be found at file named *Part 2 - Decision Tree Classifier Without Feature Cost.ipynb*.

III. Implementing a Decision Tree considering cost metrics

In the last part of the assignment, the decision tree classification algorithm is extended in a way that considers the feature costs. In order to reflect the cost of feature extraction to the decision made to construct the tree, two algorithms are provided from [3]. The equations are as follows:

$$criterion_{tan} = \frac{Gain(S,F)^2}{Cost(F)} \tag{7}$$

$$criterion_{nunez} = \frac{2^{Gain(S,F)} - 1}{(Cost(F)+1)^{\omega}} \tag{8}$$

These two equations are named after the authors that provided them in the implementation (tan and nunez respectively). These equations are now also considered as a part of the hyper-parameters where the criterion will be maximized at every split. As a modification, the prepruning algorithm only considers the tree depth in this version of the algorithm where the information gain is already prioritized with the criterion function. To determine the feature cost, Algorithm 2 is followed. As it can be observed from the formulation of the algorithm, if a feature is already used the cost 0 is assigned to it. In the actual implementation, this value is taken as 10^{-8} in order not to face with any problems as dividing by zero. This indicates that if a feature is used, the cost of using it again is a negligible number, compared with the other cost values. Also for the 21st feature in the feature set, the condition of extracting both features 19 and 20 is added. If the 21st feature is extracted, feature 19 and 20 will be extracted first, if not already extracted. As an assumption the combination cost for combining features 19 and 20 is assumed as 1.0.

A. Finding the optimal Decision Tree

In order to find the best parameters for the decision tree that implements feature costs, a grid search approach is followed like the previous part. The set of hyper-parameters used for forming the grid of parameters is as follows:

- Impurity Measures: Entropy or Gini Index
- Possible Pruning length: For applying prepruning to the model. The possible values are integers in the range [3,12].
- Cost Measure Used: If any cost measure is used, it will be either the "nunez" metric or "tan" metric. The equations for the cost criterion were provided before.

Algorithm 2: Cost Determination For a Feature

```
if $feature \in used\ features$ then
    return 0
else
    if $feature$ is the $21^{st}$ $feature$ then
        $cost \leftarrow 1$
        if $feature\ 19 \notin used\ features$ then
            $cost \leftarrow cost + costs[feature\ 19]$
        if $feature\ 20 \notin used\ features$ then
            $cost \leftarrow cost + costs[feature\ 20]$
    else
        $cost \leftarrow costs[feature]$
    return cost
```

- Cost Weight: For Nunez criterion, there is a weight involved which determines the effect of the feature cost respective to the information gain with the split. The values are selected from the set of values (0.01, 0.05 ,0.1, 0.2, 0.5, 0.75, 1, 2)

After performing the grid search for optimizing the hyper-parameters in an effort to optimize the class weighted accuracy value, the following hyper-parameters were obtained:

- Impurity Measure: Entropy
- Pruning Length: 4
- Cost Measure: Nunez
- Cost Weight: 0.75

This optimization shows that when the weight of the cost is selected in a way that the contribution of the cost is smaller than the contribution of the information gain (denoted as Gain(S,F) in the formulation), splits prioritizing performance can be performed. In additional experiments where the cost weight is high ($\omega = 2$), the decision tree grows too much without pruning and tends to overfit. This issue proves both the fact that expensive features giving more information and the effect of pruning in a considerably large decision tree. In these experiments the tree fails to achieve accuracy based performance that is achieved with low weights of cost.

The training and test set accuracy for the optimized tree is given as follows:

- Overall Accuracy: 99.21% for test set, 99.62% for training set
- Class 1 Accuracy: 98.63% for test set, 100% for training set
- Class 2 Accuracy: 100% for both training and test set
- Class 3 Accuracy: 99.18% for test set, 99.59% for training set

Even though training set on the classifier performs slightly better than the test set, the model does not show any signs of overfitting in general. The constructed decision tree can be seen in Fig. 5 and confusion matrices for training and test sets can be seen in Fig. 6.

B. Average Prediction Cost

As the final stage of evaluation, the average cot of making a prediction is assessed. In order to perform this analysis, Algorithm 2 is applied for each prediction where each prediction has its own used features. After performing this analysis, the average cost of a prediction using the optimal classifier (shown in Fig. 5) is reported as **25.59**. Considering the given cost values in the file *ann-thyroid.cost*, this cost value indicates that the expensive features will be used on average. This finding also shows that the expensive features give more information compared to the cheaper options. In the decision tree constructed it is also observed that after the 21^{st} feature is extracted, the features TT4 and T4U are used almost immediately since they are not limited by their extraction costs. Here note that TT4 and T4U are used to extract the 21^{st} feature.

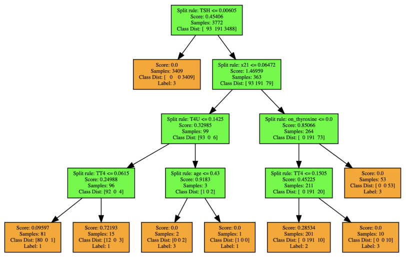

Fig. 5. Optimal tree constructed for Part 3

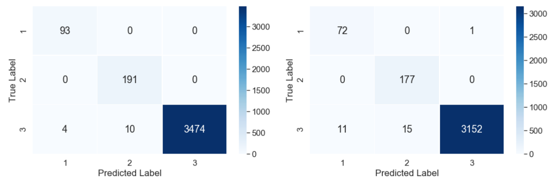

Fig. 6. Confusion Matrices for Training(left) and Test sets(right) for Part 3

As mentioned before, when the cost weight (ω) parameter grows, the tree size also grows. This issue is mainly caused due to the penalty applied by $criterion_{nunez}$ when an expensive feature is used. As it can also be seen as the result of hyper-parameter search, models with high cost parameter performed in a poorer way than the selected model. This issue clearly identifies that the use of expensive features tends to (does not guarantees for every selection) maximize the information gain for this particular dataset.

Complete implementation for the third part can be found at the file *Part 3 - Decision Tree With Cost*.

References

[1] Ç. Gündüz Demir, *Algorithm independent issues - cs 550: Machine learning*, URL: http://www.cs.bilkent.edu.tr/~gunduz/teaching/cs550/documents/CS550_AlgorithmIndependentIssues.pdf.

[2] E. Alpaydın, *Introduction to Machine Learning*. Cambridge, Massachusetts, London, England: The MIT Press, 2014.

[3] Ç. Gündüz Demir, *Decision trees - cs 550: Machine learning*, URL: http://www.cs.bilkent.edu.tr/~gunduz/teaching/cs550/documents/CS550_DecisionTrees.pdf.

Appendix A
Data Exploartion Stage

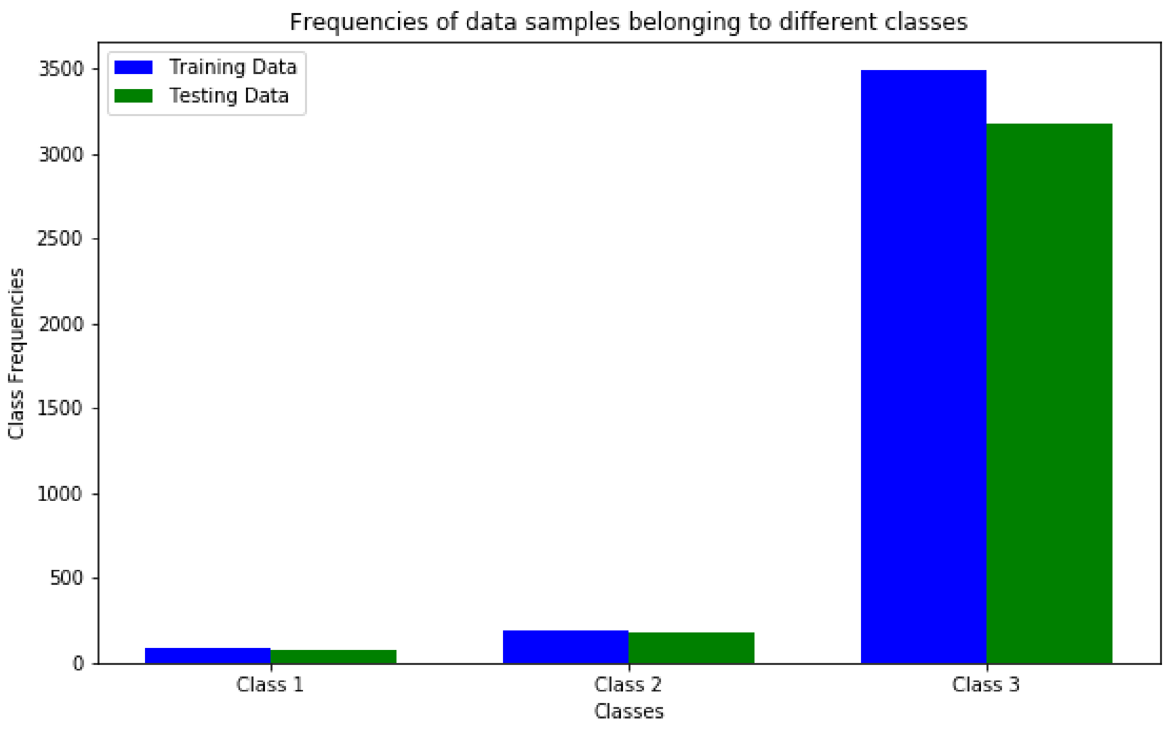

Fig. 7. Class Frequency Distribution of the given Dataset

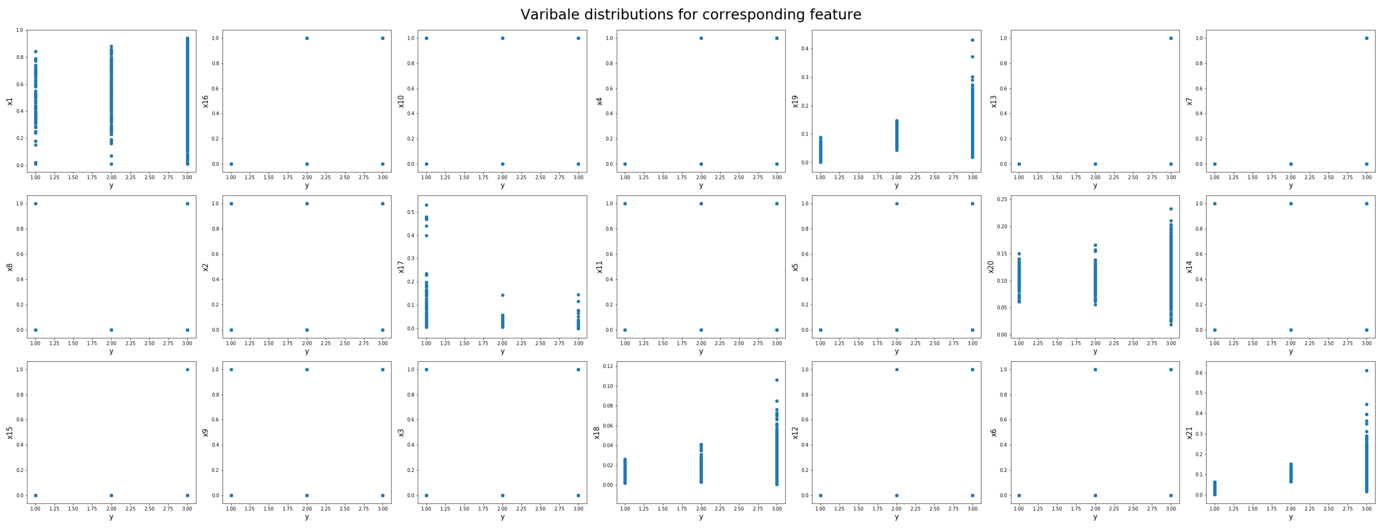

Fig. 8. Distributions of various variables in the dataset for the training set

Appendix B
Part 1 - Supplementary Plots

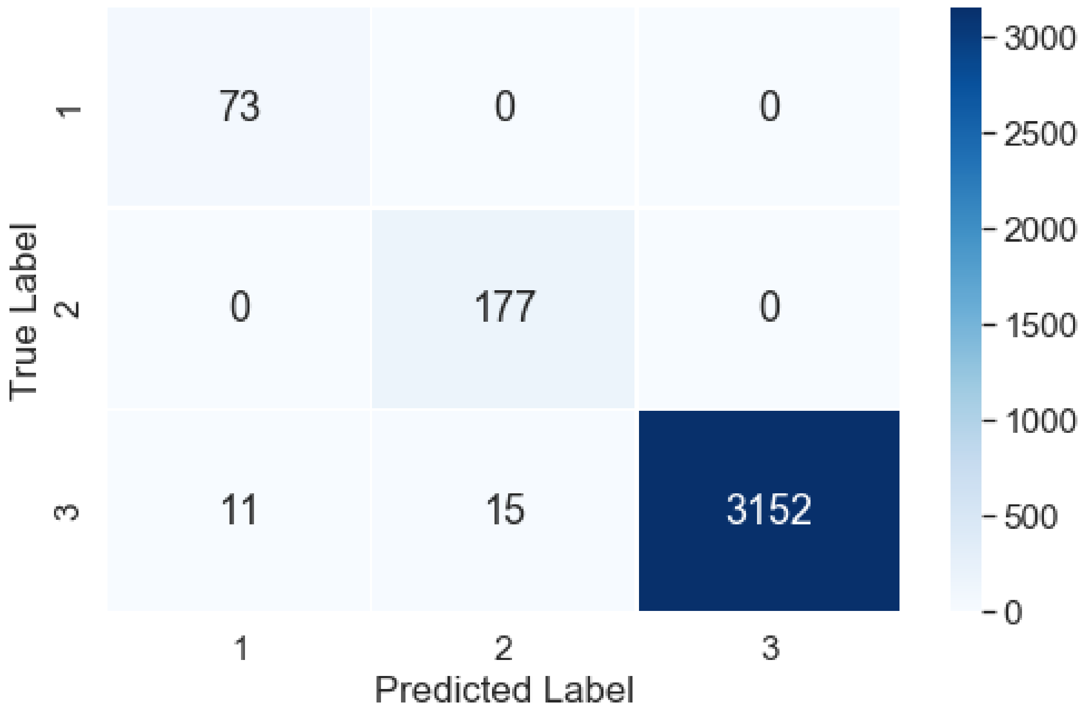

Fig. 9. Confusion matrix for the model with best hyper-parameters for test set

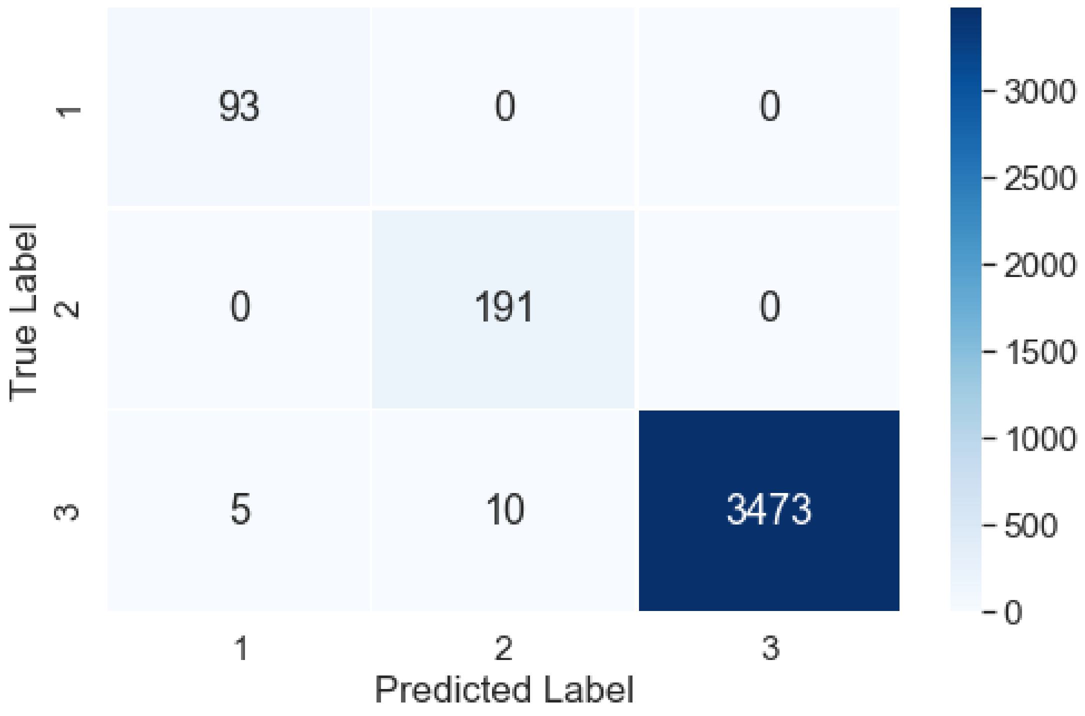

Fig. 10. Confusion matrix for the model with best hyper-parameters for training set

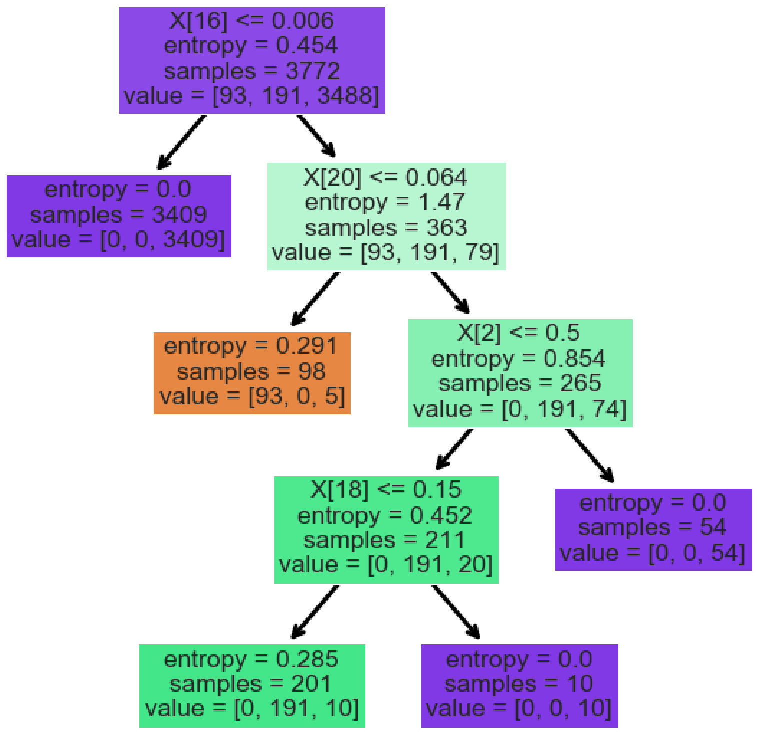

Fig. 11. Decision tree with the best hyper-parameters for Part 1

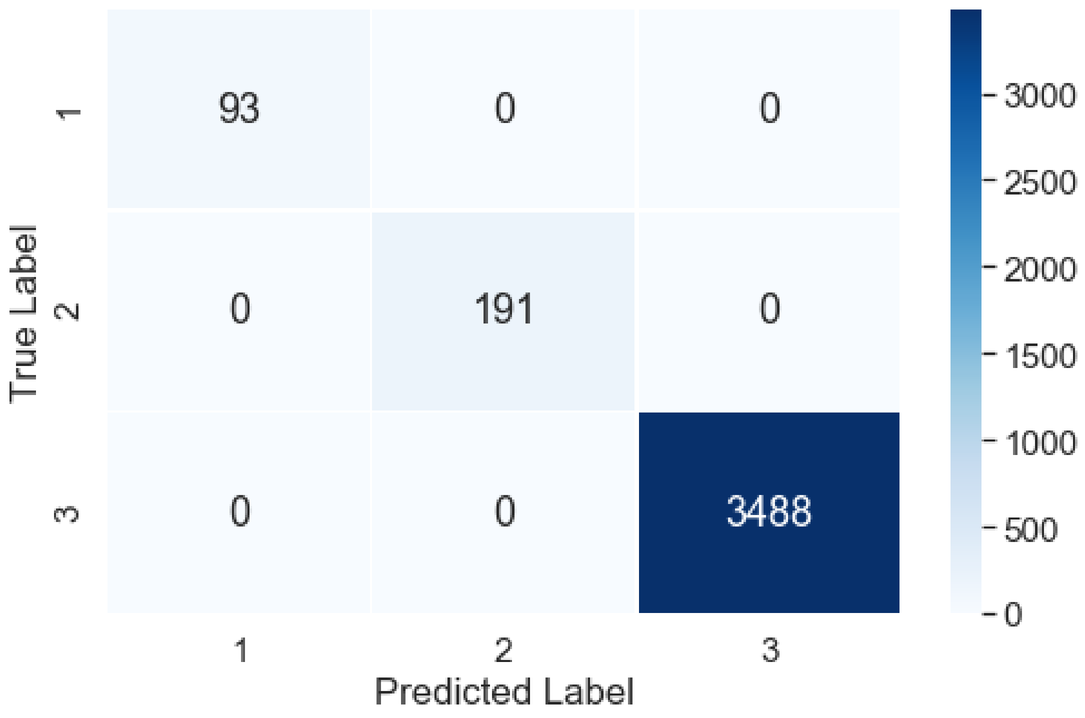

Fig. 12. Confusion matrix for training set where there is no pruning (Part 1)

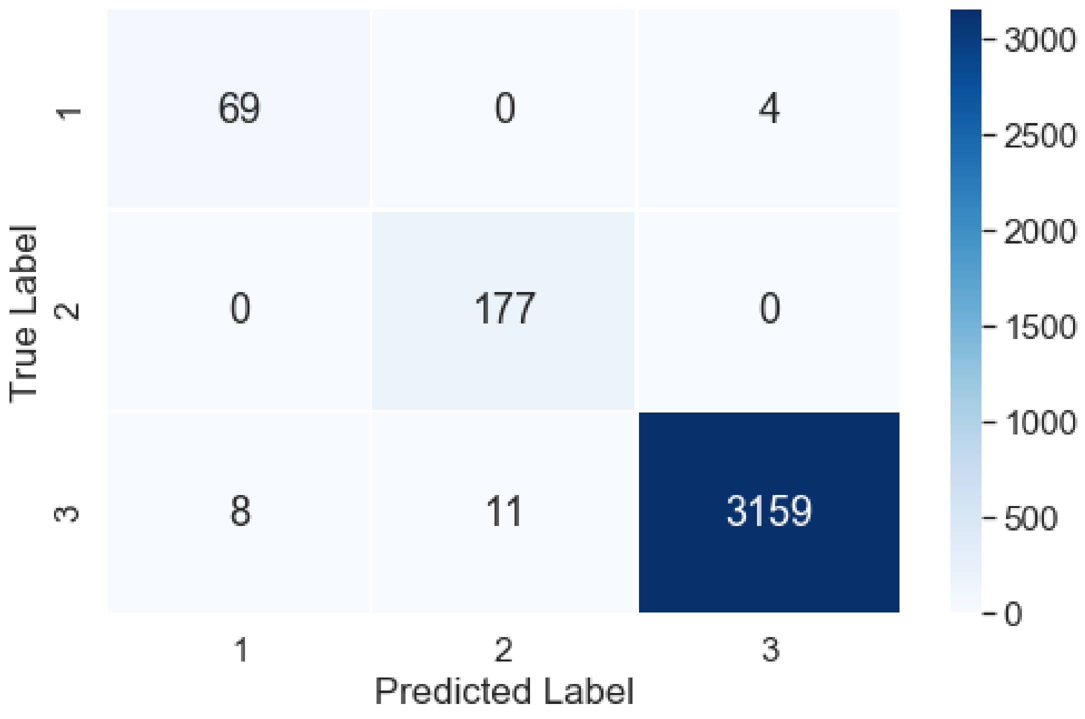

Fig. 13. Confusion matrix for test set where there is no pruning (Part 1)

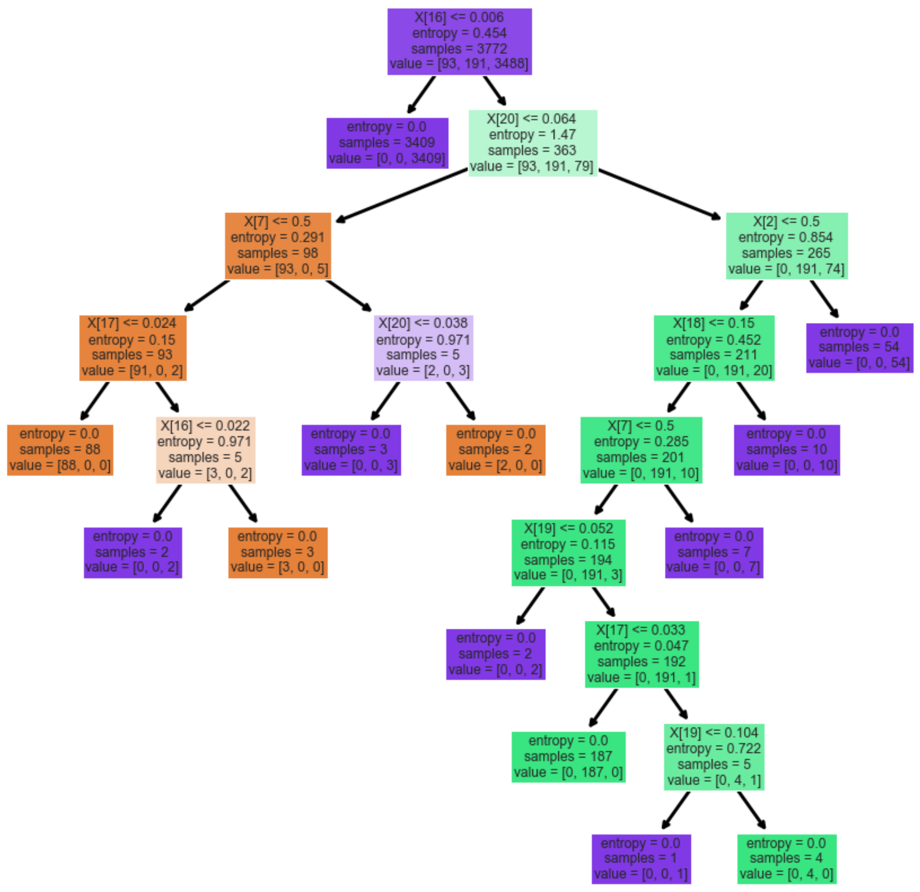

Fig. 14. Fully grown tree where there is no pruning (Part 1)

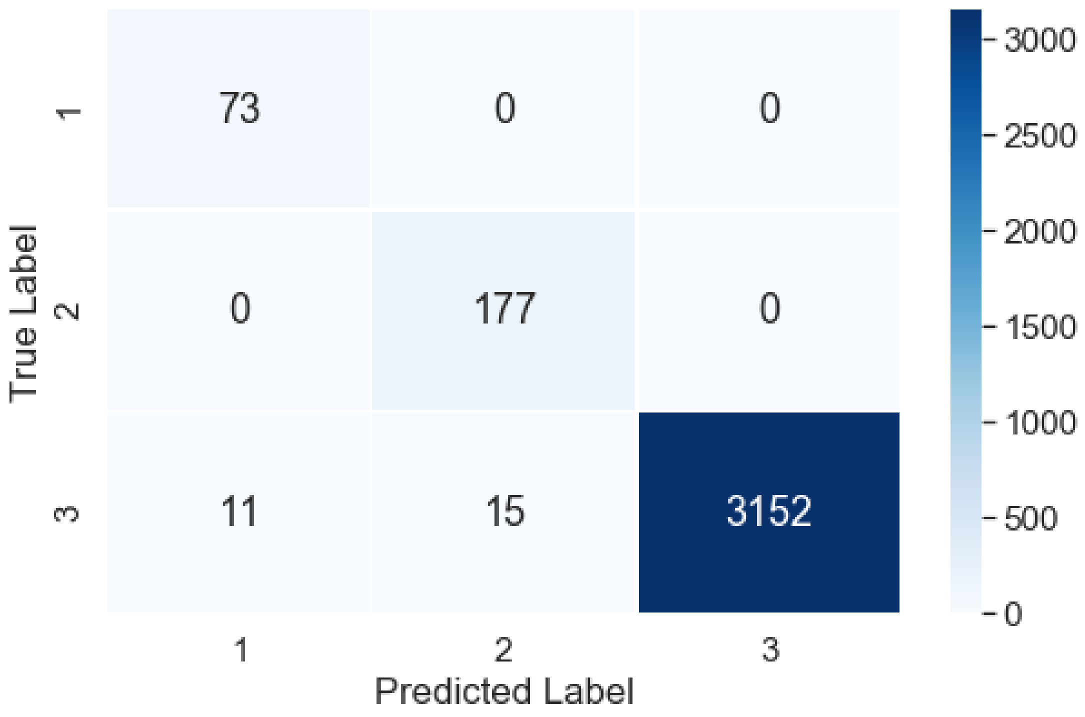

Fig. 15. Confusion matrix for test set where normalization is applied (Part 1)

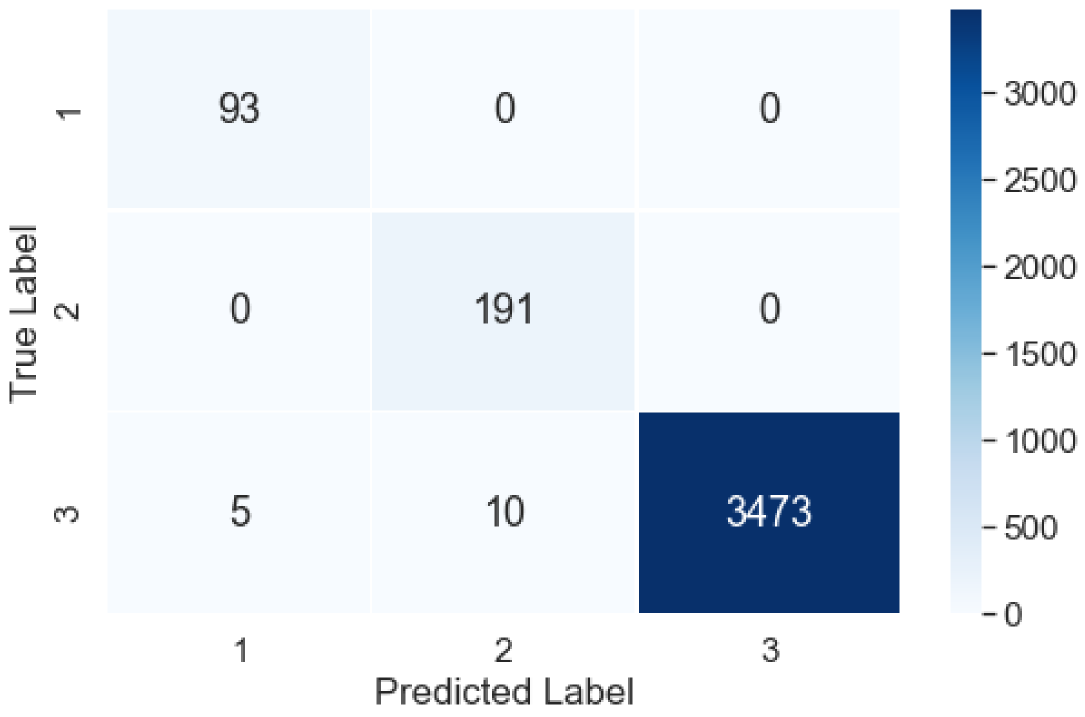

Fig. 16. Confusion matrix for training set where normalization is applied (Part 1)

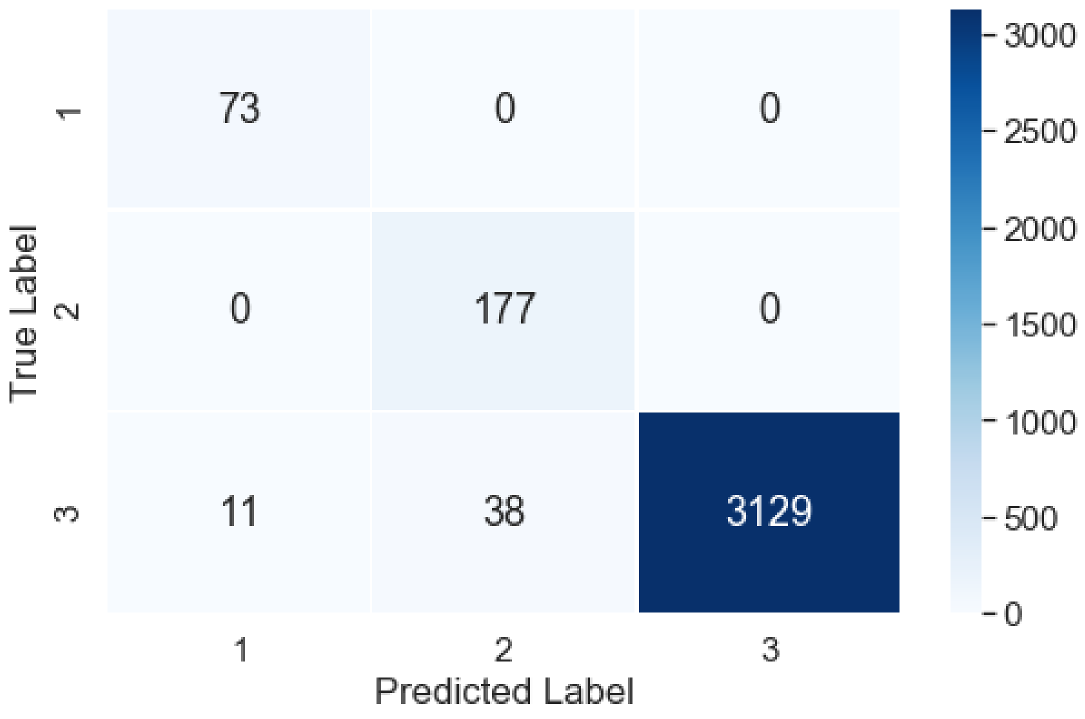

Fig. 17. Confusion matrix for test set where undersampling is applied (Part 1)

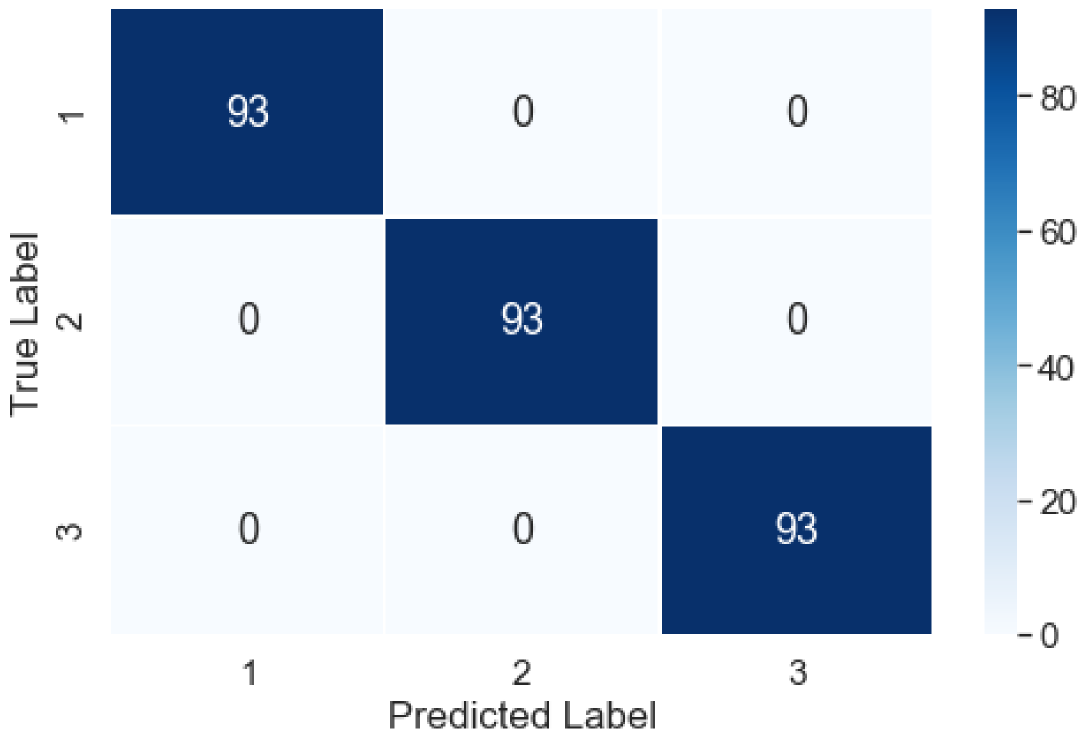

Fig. 18. Confusion matrix for training set where undersampling is applied (Part 1)

Appendix C
Part 2 - Supplementary Plots

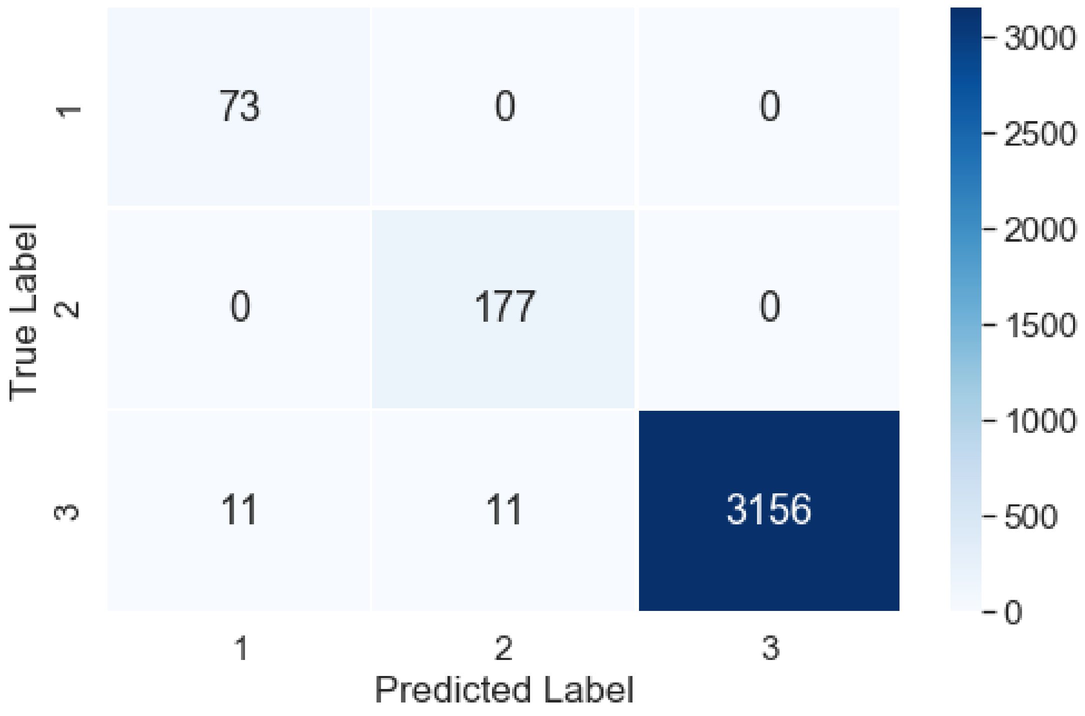

Fig. 19. Confusion matrix for test set with optimal parameters (Part 2)

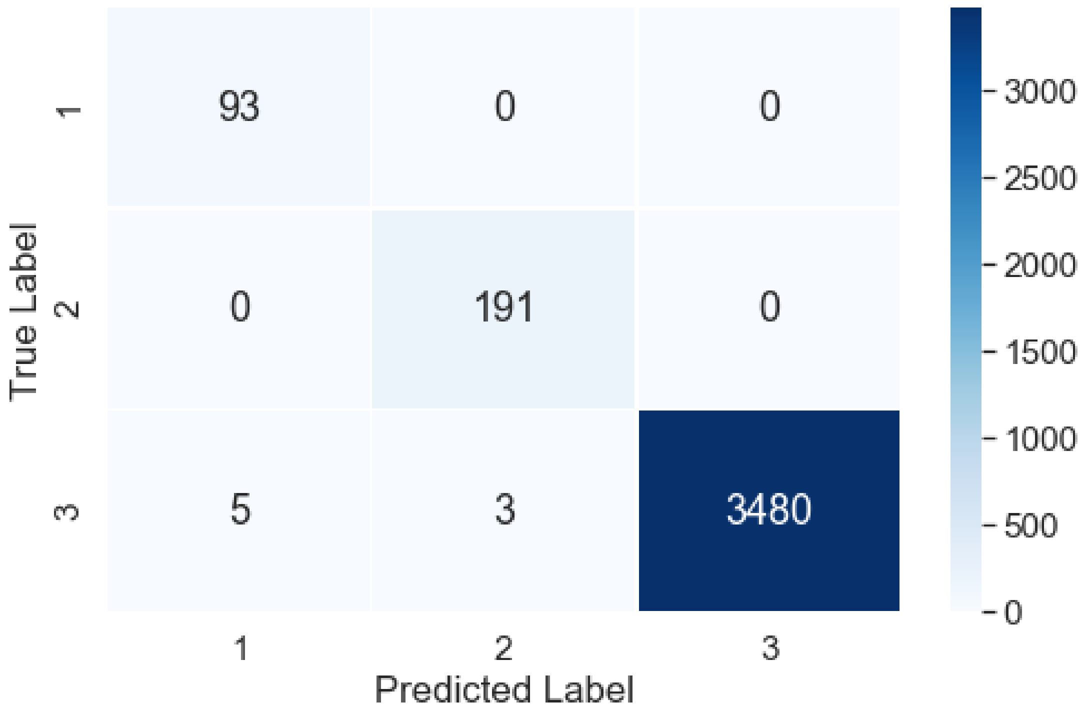

Fig. 20. Confusion matrix for training set with optimal parameters (Part 2)

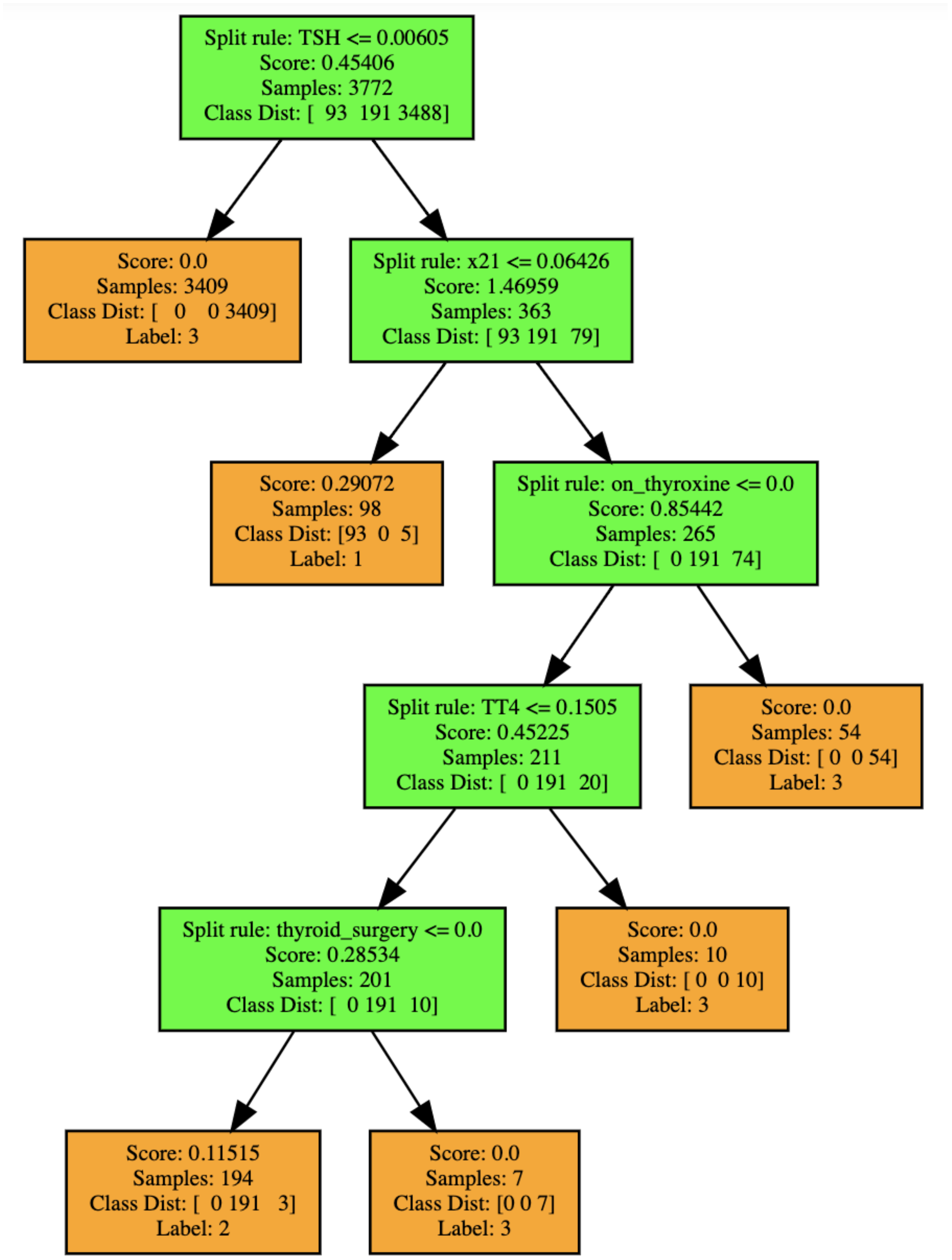

Fig. 21. Decision Tree with optimal parameters (Part 2)

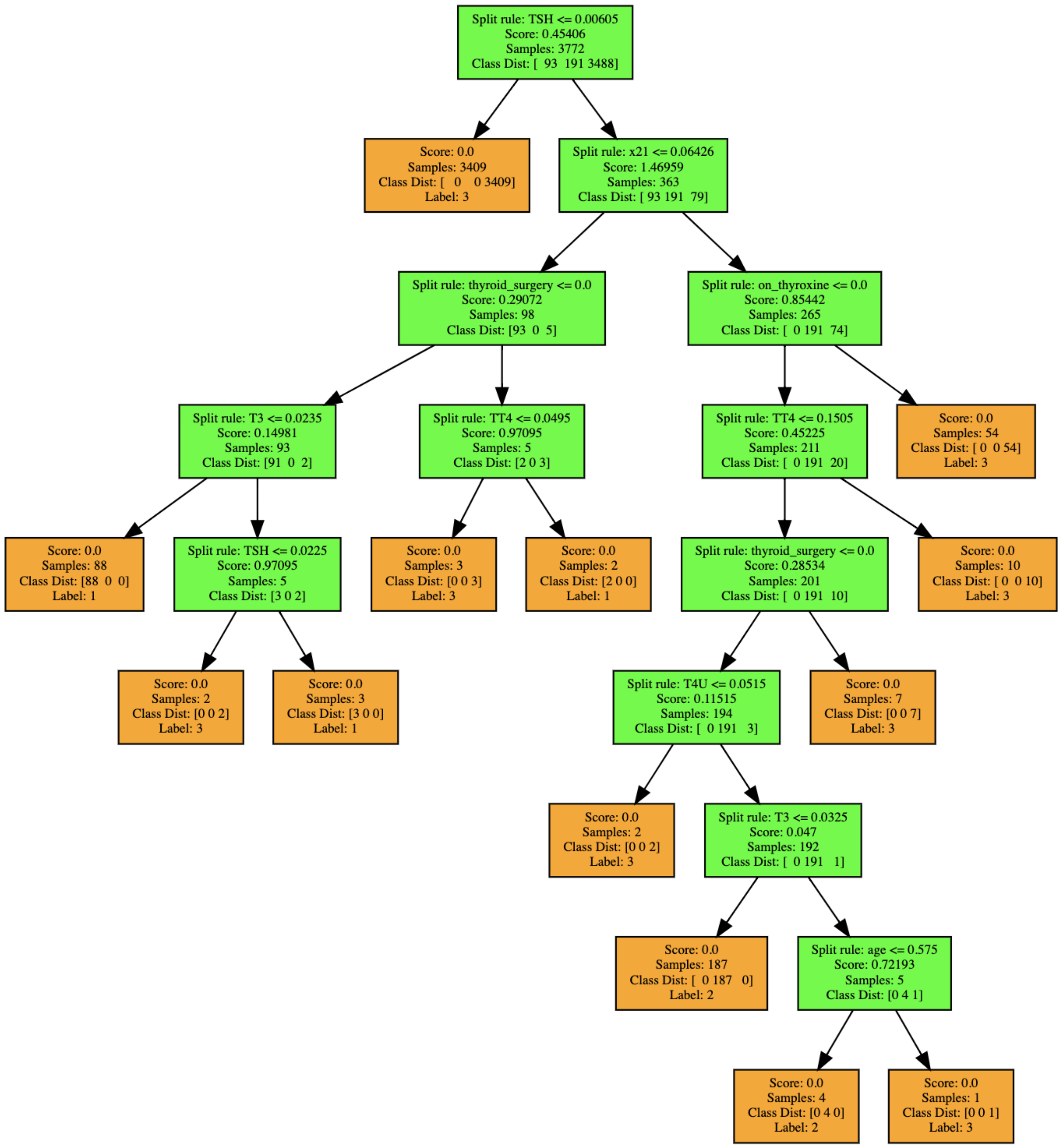

Fig. 22. Decision Tree without any pruning (Part 2)

APPENDIX D
PART 3 - SUPPLEMENTARY PLOTS

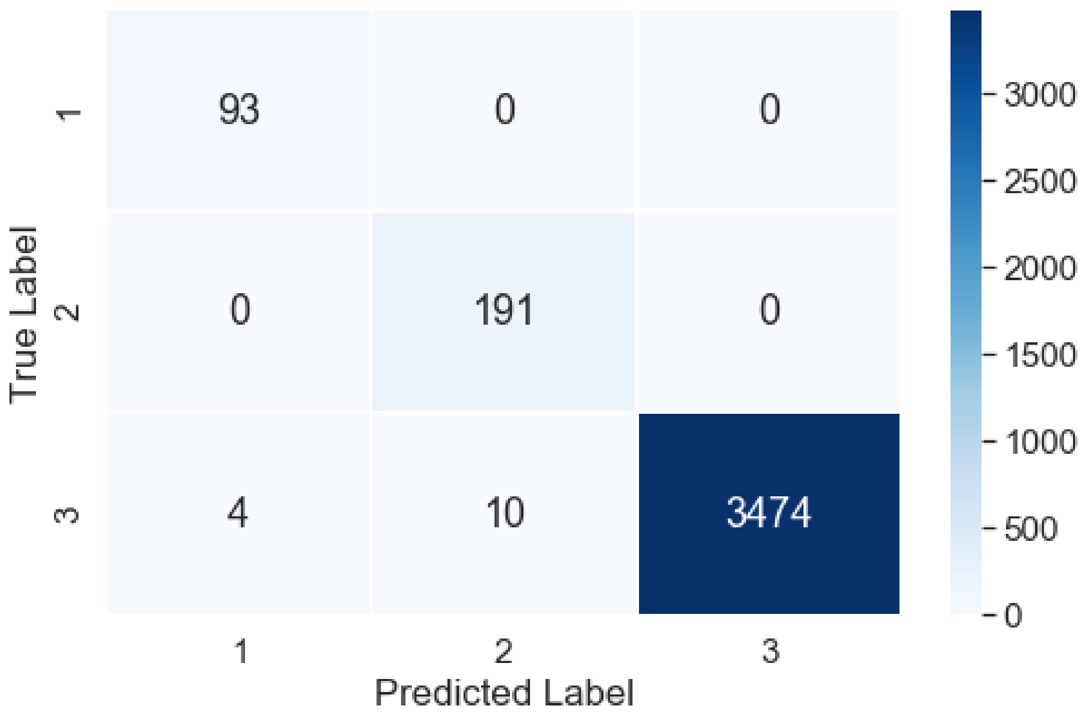

Fig. 23. Confusion Matrix with optimal hyper-parameters (Part 3)

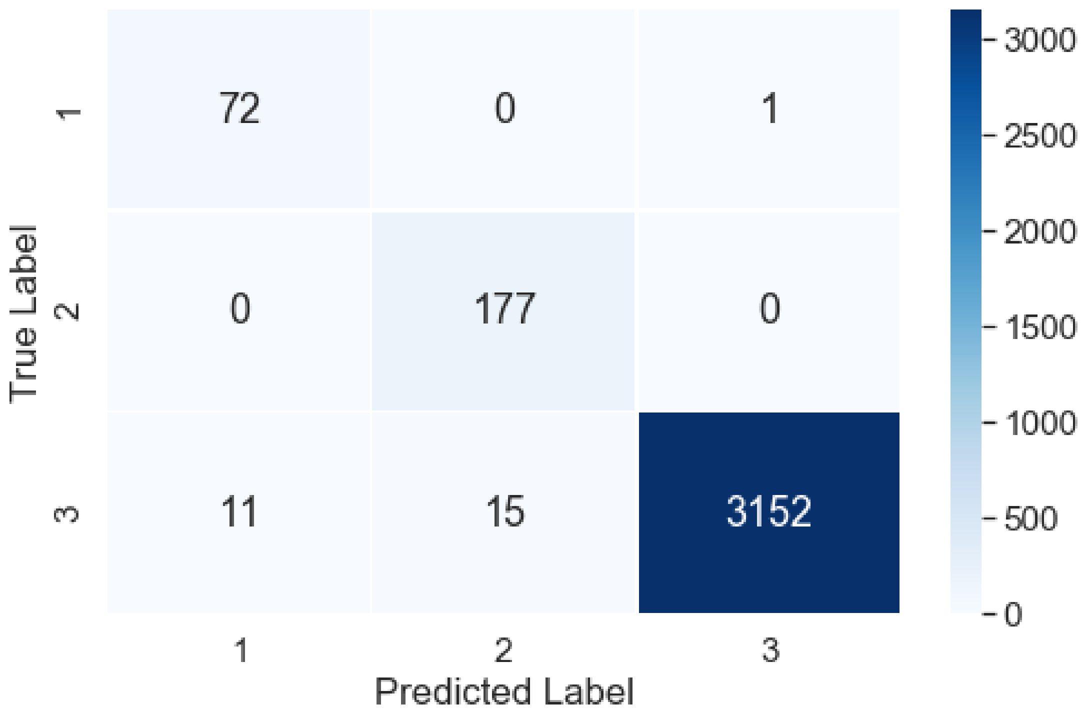

Fig. 24. Confusion Matrix with optimal hyper-parameters (Part 3)

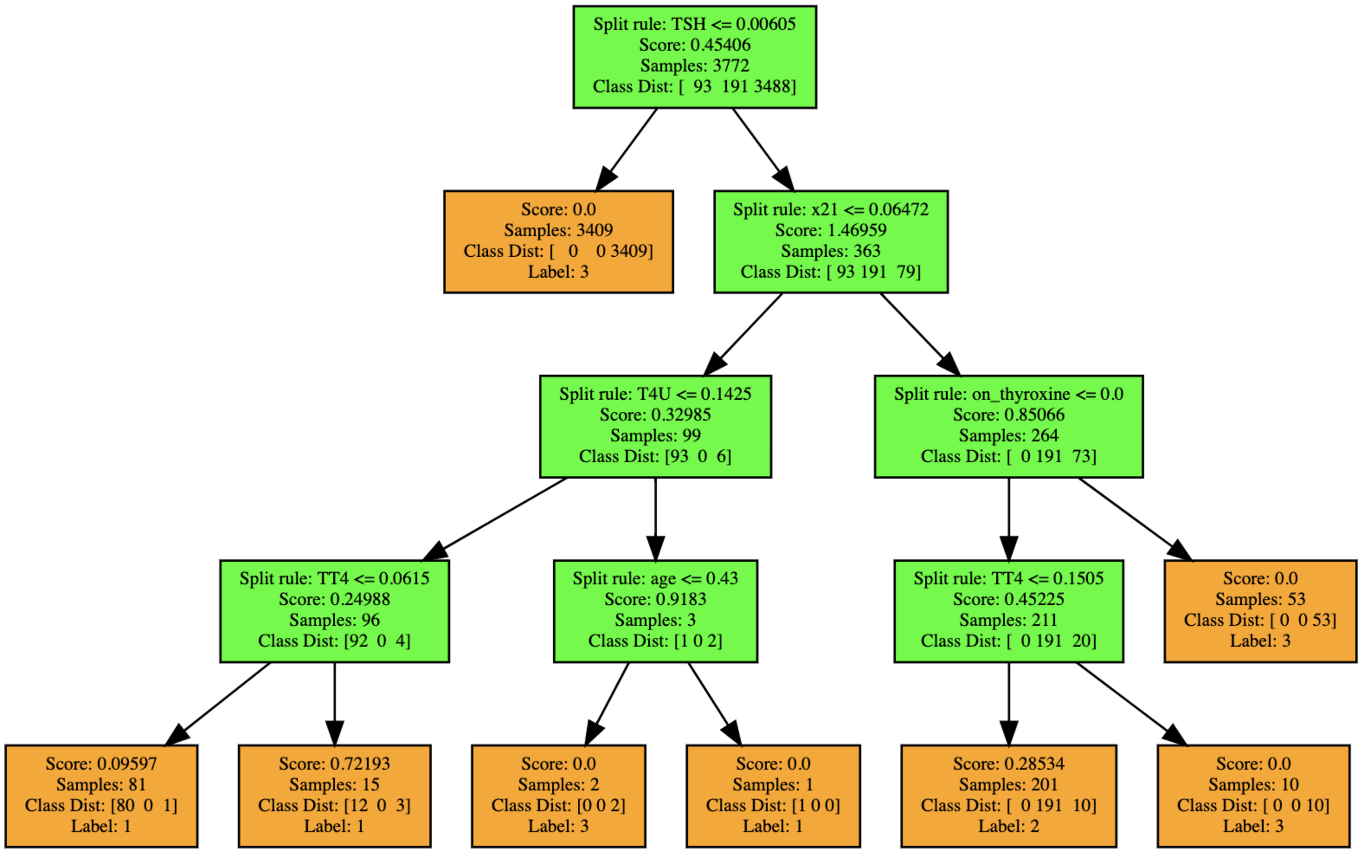

Fig. 25. Decision Tree with optimal hyper-parameters (Part 3)