NEW YORK INSTITUTE OF TECHNOLOGY

New York Institute of Technology

Machine Learning Project Assignment 1 - Report

DTSC 710/ Fall 2021 - M01

Course Instructor: Dr. Kiran Balagani

Title: Classification Project

Hui (Henry) Chen

Dataset

Throughout the project, the “Car Evaluation Data Set” from the UCI machine learning repository (URL: http://archive.ics.uci.edu/ml/datasets/Car+Evaluation) was utilized. In the dataset, there are 7 attributes, such as "buying", "maint", "doors", "persons", "lug_boot", "safety", and "class" and 1728 records. Some attributes, "buying", "maint", "lug_boot", "safety", and "class", are encoded as a discrete string representation with a ranked ordering between values. Therefore, they are ordinal variables.

Data Exploration

After exploring the dataset, there are 0% missing values. All of the attributes are discrete values with a ranked ordering between values. For instance, “buying” and “buying” attributes have the same 4 discrete values (vhigh, high, med, and low). “Doors” attribute has 4 discrete values of 2, 3, 4, and 5more. “Persons' ' attributes have three discrete values of 2, 4, and more. The “Lug_boot" attribute has 3 discrete values of small, med, and big. Lastly, the “safety” attribute has 3 discrete values of low, med, and high. All of the attributes are balanced, except the “class” attribute. “class” attribute has imbalanced values of unacc as 1210 records, acc as 384 records, good as 69 records, and vgood as 65 records. Therefore, the dataset is imbalanced (see Fig. 1 for further detail).

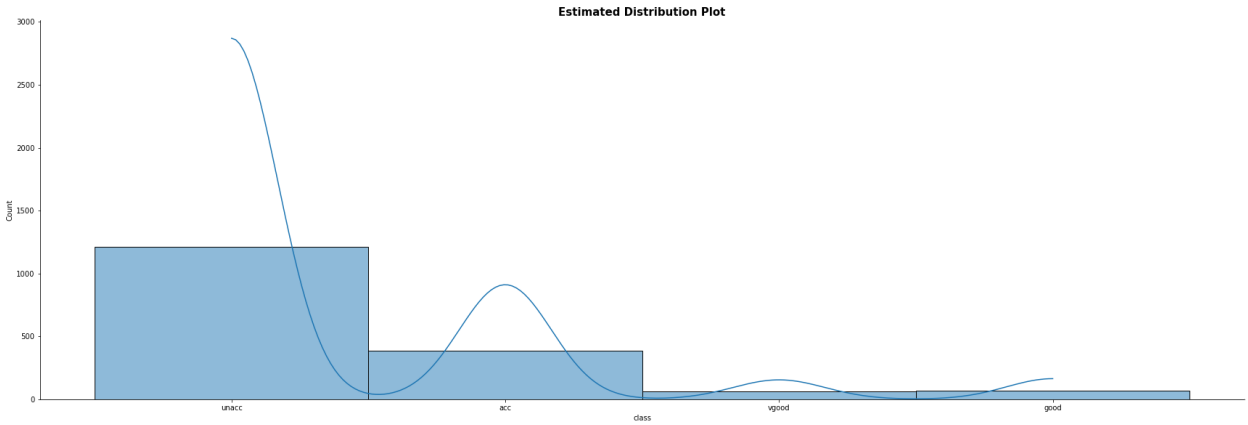

Fig.1: An estimated distribution plot of “class” attribute

As we can see from Fig.1, most of the records belong to the “unacc” class. Then follow by “acc” class. Lastly, “vgood” and “good” classes shared the same class ratio.

Data Preprocessing

Since there are no missing values, we will not consider that in the preprocessing step. Since the data is ordinal, we will apply ordinal encoding to all attributes through the “OrdinalEncoder” class with the help of the scikit-learn Python library. After applying the ordinal encoding, the string representation attributes will be encoded into ranked-numerical

values. For instance, for the “class” attribute, it will encode the "unacc" as 0, "acc" as 1, "good" as 2, and "vgood" as 3. The numerical transformation allows the machine learning model to perform computation easily. Since the ordinal encoding is different from one hot encoding method, therefore, we will not have a very sparse input feature matrix.

Once the data is encoded into numerical values, splitting the dataset into the training sets and the test sets are needed. In this process, all features, except the “class” feature, are input features matrix denoted as X. Then, the “class” feature is the output feature denoted as Y. After the X and Y split, we further split the dataset into the following test set ratios: 20%, 40%, 50%, 25% and 10%. During the splitting, we set the parameter, random state to zero so that the splits are deterministic during permutations calculation. In this project, we set the random state to zero in order to make out permutations calculation consistency. The split will generate an input feature matrix denoted as X_{train}, X_{test}, y_{train}, and y_{test}. The split result will be stored in a dictionary array where each index is corresponding to the different split ratios in the order we mentioned on the test set ratios.

Models

Throughout the entire project, there are three classifiers applied: Naïve Bayes, Decision Tree, and Random Forest. Firstly, the Naïve Bayes classifier, the Categorial Naïve Bayes model was utilized since the dataset is categorically distributed data. Also, the Categorical model performs around 20% more accuracy than the Multinomial model. All parameters of the Categorial Naïve Bayes model are set to default. Secondly, the Decision Tree model, and we set the class weight as balanced and measure the quality of a split (criterion) as entropy. In that way, the model automatically adjusts weights inversely proportional to class frequencies in the input data. Thirdly, in the Random Forest model, we experimented with the different number of ntree parameters in the range of 1 - 50 in order to find the optimal number of trees in this model. It turns out that regardless of how many trees we have, it will not affect the accuracy of the model. Therefore, we set 4 trees and the rest of the parameters as the decision tree parameters - since the random forest is an ensemble method of the decision tree. Once all input features matrices are fitted and transformed on the model, we store the result, such as $\hat{y}_{train}$, $\hat{y}_{test}$, precision, recall, f1-score, support, accuracy, and the confusion matrix into the respective dictionary array that is associated with different test ratios. Once all the models are trained, we utilize the model results to further derive metrics. E.g., Correct Classified per class.

Model Analysis 1: Naïve Bayes

After training the Naïve Bayes model, the accuracy is as follows:

Accuracy\ Test Ratio	0.2	0.4	0.5	0.25	0.1

	0.8237	0.8035	0.8032	0.8056	0.8555

Fig.2: Accuracy table of the Naïve Bayes model with respect to different test ratios.

As we can from Fig.2, the best accuracy occurred at test ratio split 0.1 with around 0.8555 accuracies. The lowest accuracy occurred at a 0.5 test split ratio of 0.8032 accuracies. The mean accuracy is 0.821475; therefore, 0.2 is the optimal test split in terms of accuracy metric.

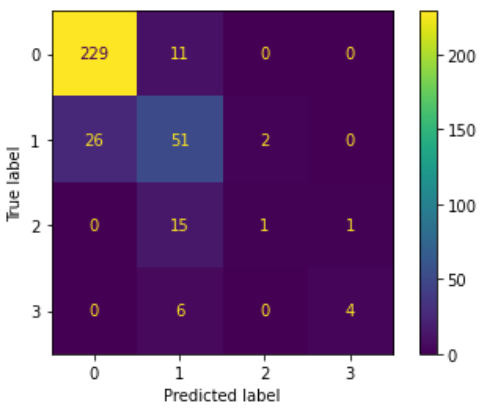

Fig.3: confusion matrix with test ratio of 0.2

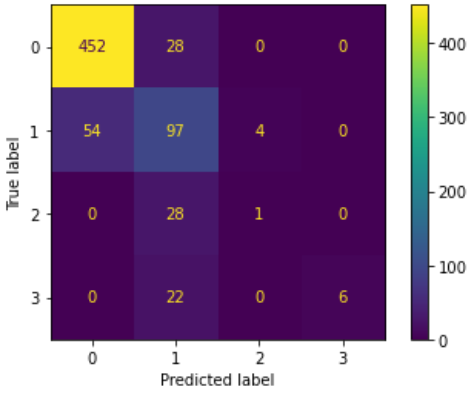

Fig.4: confusion matrix with test ratio of 0.4

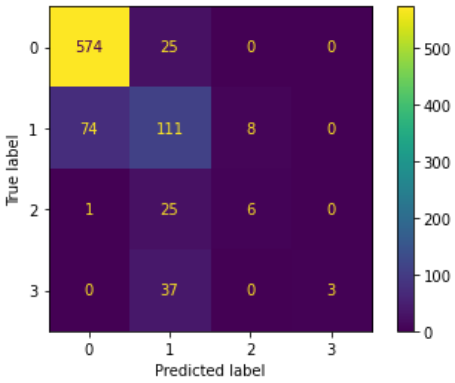

Fig.5: confusion matrix with test ratio of 0.5

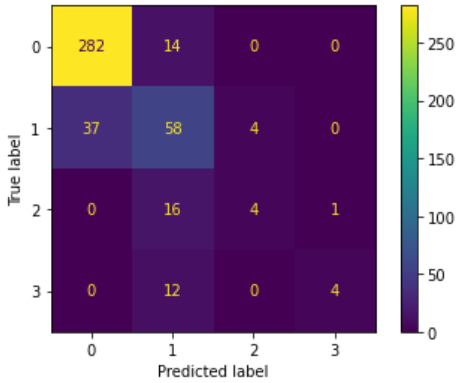

Fig.6: confusion matrix with test ratio of 0.25

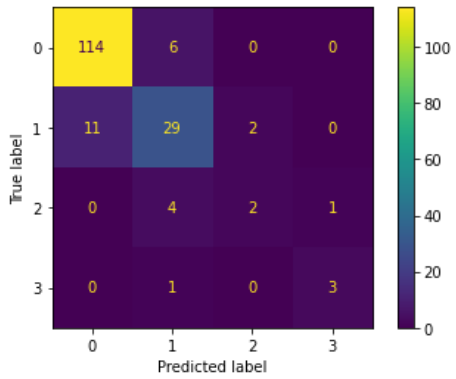

Fig.7: confusion matrix with test ratio of 0.1

With the help of Fig.3 - Fig.7, we can further derive the per class classification accuracy with respect to test ratios for the Naïve Bayes model.

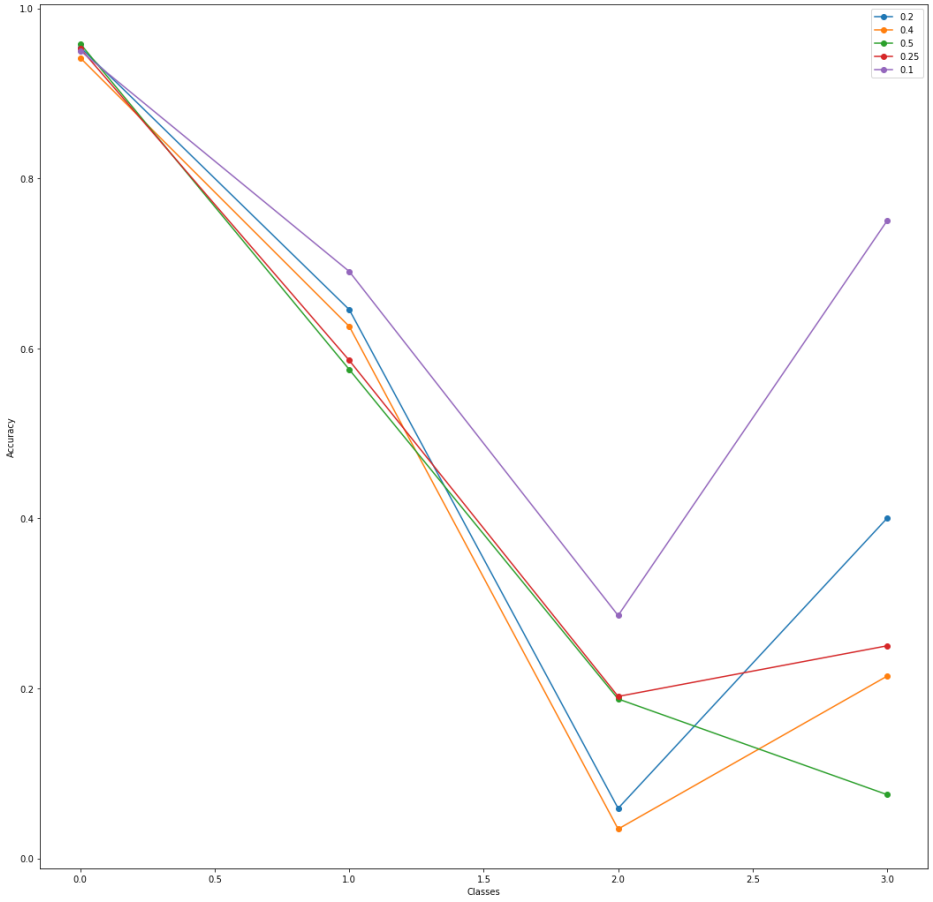

Fig.8: Naïve Bayes Metrics: per class classification accuracy

In Fig.8, the x-axis is the class label and the y-axis is the accuracy score in the range of 0 - 1. The class labels are represented as follows throughout the project: unacc denoted as 0, acc denoted as

1, good denoted as 2, and vgood denoted as 3. As we can see, 0.1 test split gives the best per class classification accuracy, which is true since the split has the highest accuracy amongst other test split ratios in the same model. However, because the dataset is imbalanced, class good has, and always, the lowest accuracy, followed by vgood and acc class label.

Model Analysis 2: Decision Tree

After training the model, the accuracy metrics are as follows:

Accuracy\ Test Ratio	0.2	0.4	0.5	0.25	0.1
	0.9624	0.9783	0.9641	0.9722	0.9538

Fig.9: Accuracy table of the Decision Tree model with respect to different test ratios

By looking at the accuracy table, the best accuracy occurred at the split of 0.4 with 0.9783 accuracies. While the lowest accuracy occurred at the split of 0.1 with 0.9538 accuracies. The mean accuracy is 1.2077, which automatically implies that the model is overfitting.

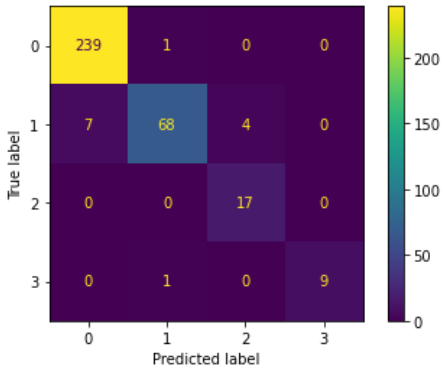

Fig.10: confusion matrix with test ratio of 0.2

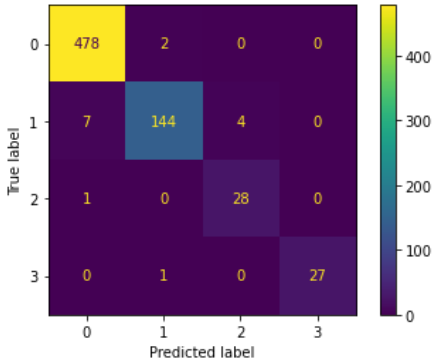

Fig.11: confusion matrix with test ratio of 0.4

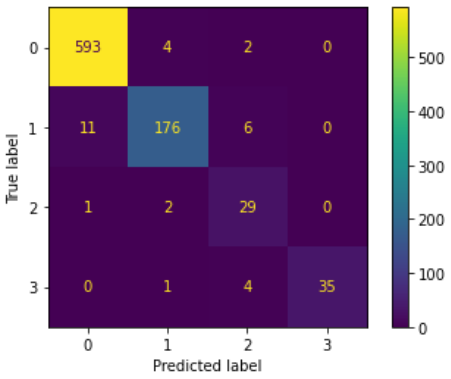

Fig.12: confusion matrix with test ratio of 0.5

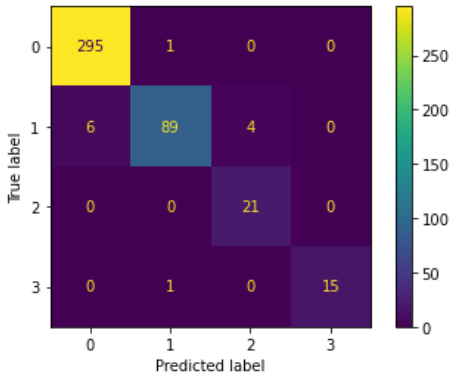

Fig.13: confusion matrix with test ratio of 0.25

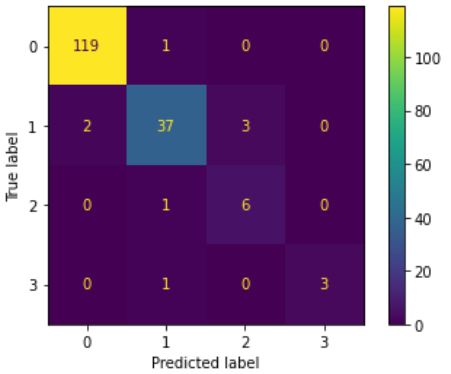

Fig.14: confusion matrix with test ratio of 0.1

With the help of Fig.10 - Fig.14, we can further derive the per class classification accuracy with respect to test ratios for the Decision Tree model.

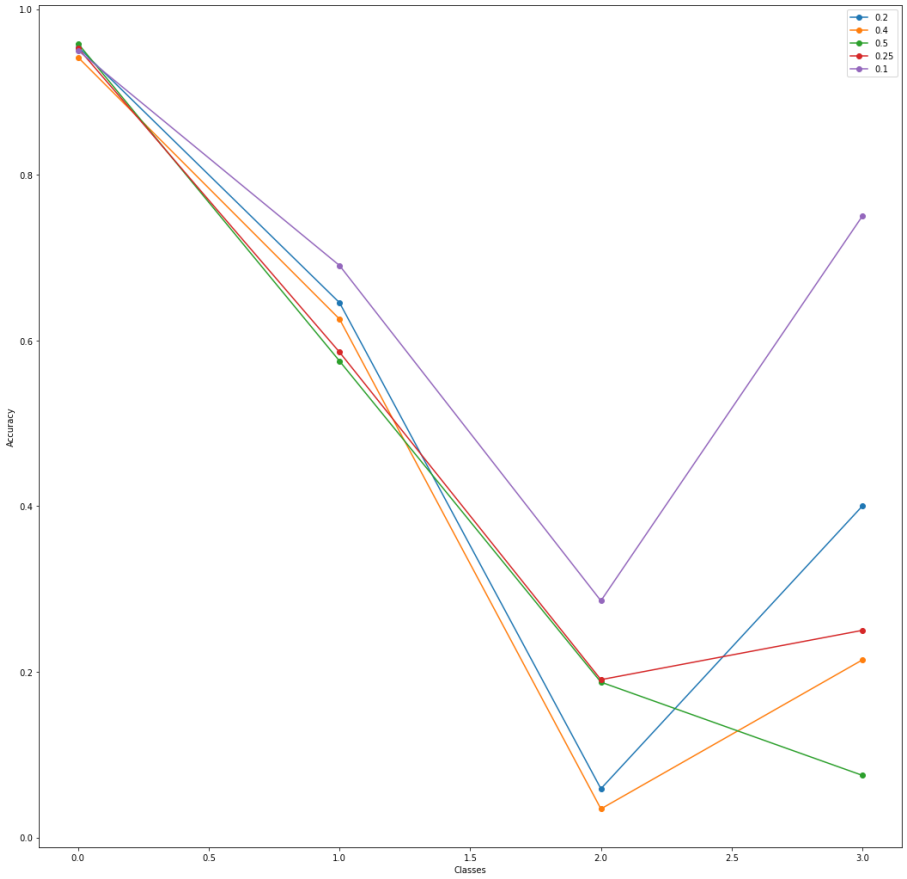

Fig.15: Decision Tree Metrics: per class classification accuracy

As we can see, 0.1 test split gives the best per class classification accuracy, which is true since the split has the highest accuracy amongst other test split ratios in the same model. However, because the dataset is imbalanced, class good has, and always, the lowest accuracy, followed by vgood and acc class label. Furthermore, the model with a 0.4 test split ratio will not be recommended since it performs poorly on acc, good, and vgood classes, which may result in misclassifying the data in the production environment.

Since we already detect the model is overfitting, we would like to find out how much it overfits. With the help of the input feature matrix, we further derive the overfitting graph with respect to different test ratios.

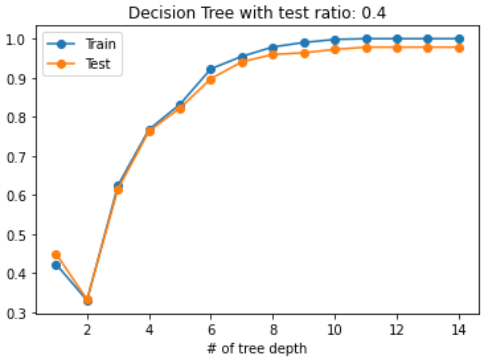

Fig.16: D. Tree overfitting at 0.4 test ratio

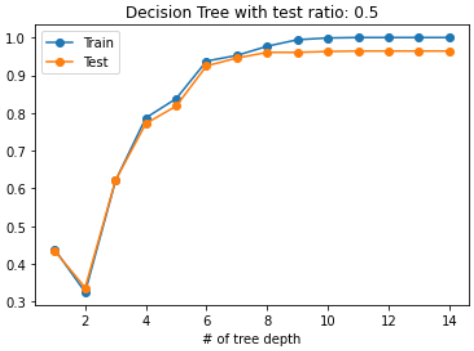

Fig.17: D. Tree overfitting at 0.5 test ratio

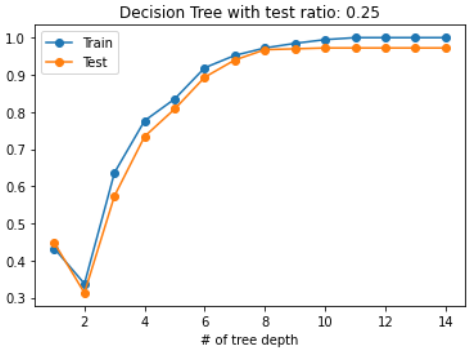

Fig.18: D. Tree overfitting at 0.25 test ratio

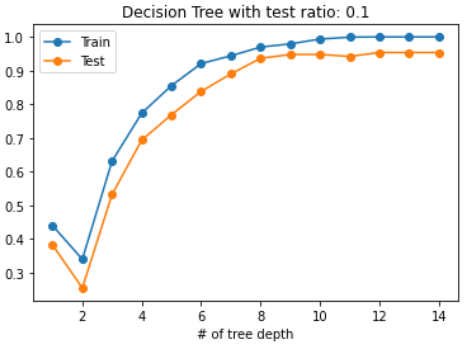

Fig.19: D. Tree overfitting at 0.1 test ratio

As we can see from Fig.16 - Fig.19, the best split model occurred at 0.4 splits, which did not overfit the model as much compared to other split ratio models. However, with a 0.4 split, it underfitting the model at 1 and 2 of tree depth but gradually overfitting as the number of tree depth increases. The same phenomenon applies to 0.5 and 0.25 test ratio splits. The most overfitting model occurred at a 0.1 test split where regardless of the number of tree depth, it always overfitted. There are a number of solutions to resolve this problem, which will be detailed under the limitation section.

Model Analysis 3: Random Forest

After training the model, we get the following accuracy table:

Accuracy\ Test Ratio	0.2	0.4	0.5	0.25	0.1
	0.9249	0.9249	0.9249	0.9249	0.9249

Fig.20: Accuracy table of the Random Forest model with respect to different test ratios

By looking at the accuracy table, all of the accuracies are the same, which is true by comparing to the experiments that we performed under the model section where the number of ntree in the forest does not affect the performance of the model. However, all of the models (regardless of the test ratio splits) achieved above 0.92 accuracies, which automatically implies that the model is overfitting. We will detail the issue of overfitting and how to deal with it in the Random Forest Model under the limitation section.

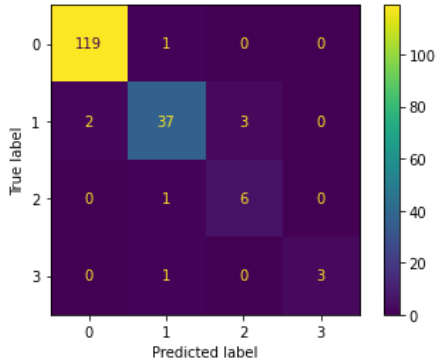

Fig.21: confusion matrix at 0.2 test ratio

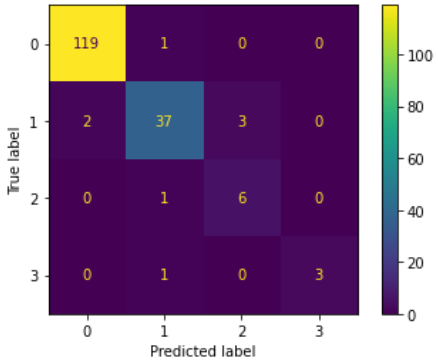

Fig.22: confusion matrix at 0.4 test ratio

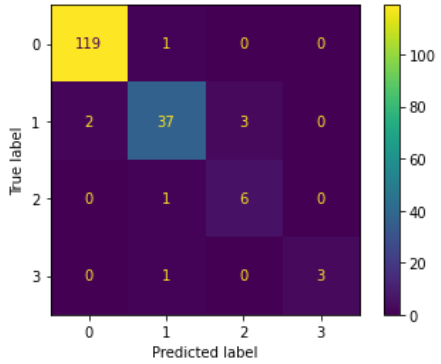

Fig.23: confusion matrix at 0.5 test ratio

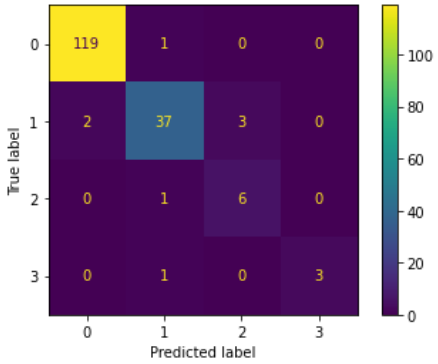

Fig.24: confusion matrix at 0.25 test ratio

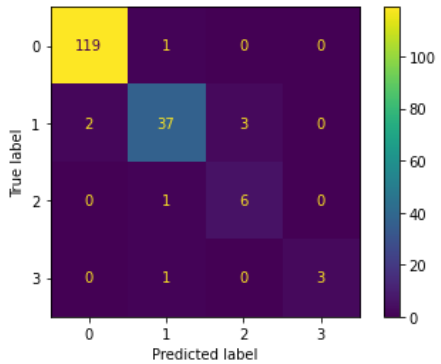

Fig.25: confusion matrix at 0.1 test ratio

With the help of Fig.21 - Fig.25, we can further derive the per class classification accuracy with respect to the different test ratios.

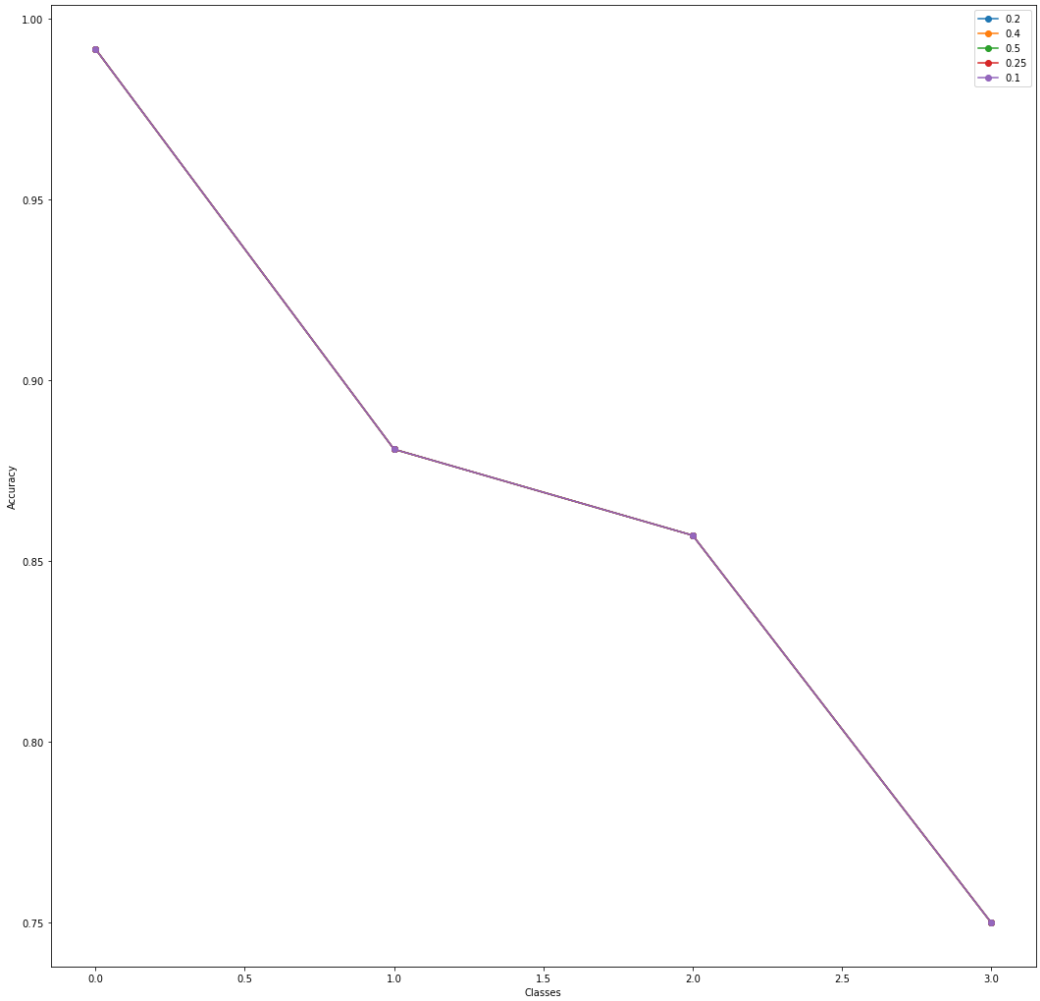

Fig.26: Random Forest Metrics: per class classification accuracy

As we can observe from Fig.26, all the per-class classification accuracies are the same, which is true since all the test ratio models have the same accuracies. Based on Fig.26, the Random Forest classifies well “good” class label compared to the Naïve Bayes and Decision Tree models. However, the Random Forest classifies poorly on the “vgood” class label data compared to the other two models.

Since we observed there are some possibilities of overfitting in the Random Forest model, we would like to find out how much the model overfits.

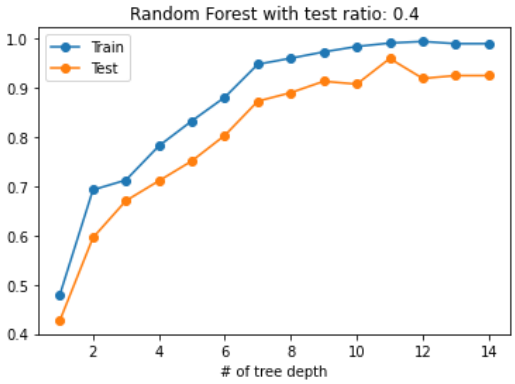

Fig.27: R. Forest overfitting at 0.4 test ratio

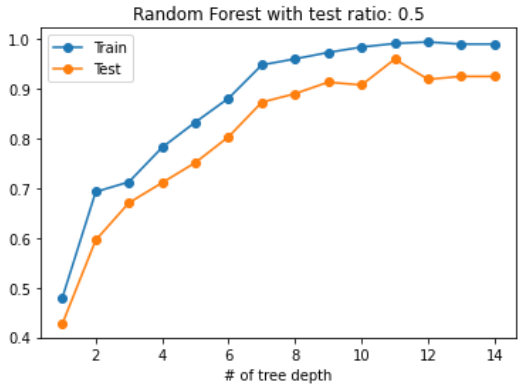

Fig.28: R. Forest overfitting at 0.5 test ratio

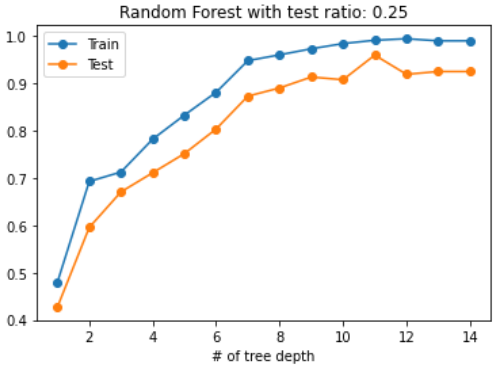

Fig.29: R. Forest overfitting at 0.25 test ratio

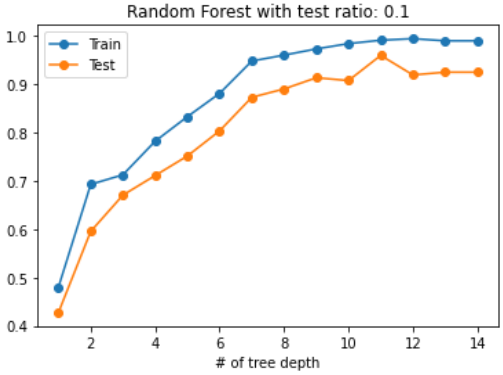

Fig.30: R. Forest overfitting at 0.1 test ratio

From Fig.27 - Fig.30, the Random Forest model is overfitting regardless of the number of tree depths. As the training size and the number of tree depth increases, the overfitting metrics gradually increase as well. Once again, we will discuss further the overfitting of the model in the limitation section.

Limitations

Throughout the project, there are a number of limitations:

1) The dataset is imbalanced based on Fig.1 where most of the observations are labelled as "unacc" while only a few of the observations are labelled "good" and "vgood" classes. There are two possible solutions to address this issue. Firstly, we could perform data sampling. Under the sample, we could perform the following two (either one):
    a) Upsampling on the class label of "acc", "good", and "vgood" since there are a few observations under these three labels.
    b) Downsampling on the "unacc" label since most of the observations fall under this class label.

    However, due to the limit of time, we will not perform sampling on the project.

2) The Decision Tree and the Random Forest models are overfitting regardless of the test ratio size in this project. There are two solutions to this problem:
    a) The first option is to have more training data for the model. However, since the dataset is imbalanced, the ideal way would be to have data sampling such as upsampling or downsampling. In that way, we would, possibly, resolve the overfitting and the imbalanced dataset at the same time.
    b) The second option is to perform hyperparameters tuning on the Decision Tree and Random Forest model. With different trials of different combinations of model parameters in order to find an optimal parameter for the model. However, due to the limitations of time in this project, we did not perform this approach.

Closing Remarks

In a nutshell, the Naïve Bayes classifier is most sensitive to the training sample size, followed by the Decision Tree, and the Random Forest classifier. However, since the scale of the dataset is low, Decision Tree and Random Forest models are easily overfitting. Therefore, without sampling or hyperparameter tuning, the most optimal model would be the Naïve Bayes classifier for this dataset in this project.

Source Code: https://github.com/hchen98/DTSC710-ML