Faculty of Information Technology
University of Science, VNU HCMC

AI
Lab 02
Report

Prepared by
Thien Huynh Duc - 21127693
August 21, 2023

Table of Contents

1. Project Introduction

1.1. Information

Project

Decision Tree on Nursery Database

Student

Full name: Huynh Duc Thien
ID: 21127693
Contact: hdthien21@clc.fitus.edu.vn
Course: CSC14003 - Artificial Intelligence
Class: 21CLC07

Mentor

Teacher, Nguyen Ngoc Thao
Teacher, Le Ngoc Thanh
Teacher, Nguyen Tran Duy Minh

1.2. Decision Tree

Decision trees are powerful machine learning models that excel in both classification and regression tasks. They offer an intuitive and interpretable approach to decision-making by representing decisions and their potential consequences in a tree structure. In this project, we explore the application of decision trees using **scikit-learn**, a popular machine learning library, in the context of the Nursery Database.

The Nursery Database is a real-world dataset originally developed as a hierarchical decision model for ranking applications to nursery schools. It captures various attributes and criteria used in evaluating the suitability of nursery schools for children. These attributes include parents' occupation, nursery quality, class form, number of children, housing conditions, family financial status, social factors, and children's health.

By leveraging decision tree models on the Nursery Database, we aim to gain insights into the decision-making process and predict the rankings of nursery school applications. Decision trees enable us to uncover patterns and relationships between the attributes, thereby understanding the factors that contribute to the ranking outcomes within the hierarchical structure.

Using scikit-learn's DecisionTreeClassifier, we will build decision tree models that replicate the decision model of the Nursery Database. Additionally, an evaluation of the models using performance metrics such as accuracy, precision, recall, and F1-score to assess their predictive power will be made.

2. Completion levels evaluation

No.	Work	Completion (%)
1	Preparing the data sets	100
2	Building the decision tree classifiers	100
3	Evaluating the decision tree classifiers	
	● Classification report and confusion matrix	100
	● Comments	100
4	The depth and accuracy of a decision tree	
	● Trees, tables, and charts	100
	● Comments	100

3. Preparing the data sets

3.1. Preparing subsets

The provided block of code utilizes the scikit-learn function 'train_test_split' to divide the dataset into two distinct sets: a training set and a test set. This function takes the input variables 'features' and 'labels' that have been extracted from the original data frame. It utilizes the parameters 'train_size' and 'test_size' to determine the proportion of data to allocate to the training set and test set, respectively.

To preserve the distribution of classes within the data, the 'stratify' parameter ensures that the proportion of each class is maintained in both the training and test sets. Setting the 'shuffle' parameter to True enables the data to be randomly shuffled before the split takes place. Additionally, the 'random_state' parameter is employed to set a specific random seed, ensuring the reproducibility of the split.

The resulting splits generated by the function are stored within a list called 'train_test_sets', containing the tuple of the training set and test set.

```python
train_test_sets = []

for prop in proportions:
    features_train, features_test, labels_train, labels_test = train_test_split(features.copy(),
                                                                                 labels.copy(),
                                                                                 train_size=prop[0],
                                                                                 test_size=prop[1],
                                                                                 shuffle=True,
                                                                                 stratify=labels,
                                                                                 random_state=0)
    train_test_sets.append((features_train, labels_train, features_test, labels_test))
```

Figure 1: prepare subsets code (detail)

3.2. Visualize the distribution of classes in all data sets

The code segment presented visualizes the distribution of data among different classes in the dataset, with a specific emphasis on the training and test sets derived through the 'train_test_split' operation. It employs bar charts to compare the counts of each class in the original data, training data, and test data, considering different proportions of the split. This visualization enables a comparative analysis of class distributions and provides insights into the impact of the split operation on the dataset.

```python
data_counts = labels.value_counts().sort_index()
train_counts = [labels_train.value_counts().reindex(data_counts.index, fill_value=0)
                for (_, labels_train, _, _) in train_test_sets]
test_counts = [labels_test.value_counts().reindex(data_counts.index, fill_value=0)
               for (_, _, _, labels_test) in train_test_sets]

for i, proportion in enumerate(proportions):
    plt.figure(figsize=(12, 6))
    plt.title(f"Data Distribution (Train: {proportion[0]}, Test: {proportion[1]})")

    x = np.arange(len(data_counts))
    width = 0.25
    plt.bar(x - width, data_counts, width, label='Original Data')

    x_train = np.arange(len(train_counts[i]))
    plt.bar(x_train, train_counts[i], width, label='Training Data')

    x_test = np.arange(len(test_counts[i]))
    plt.bar(x_test + width, test_counts[i], width, label='Test Data')

    plt.xlabel("Class")
    plt.ylabel("Count")
    plt.xticks(np.arange(len(data_counts)), data_counts.index, rotation=45)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

Figure 2: visualization code (detail)

After conducting the visualization, the result outputs:

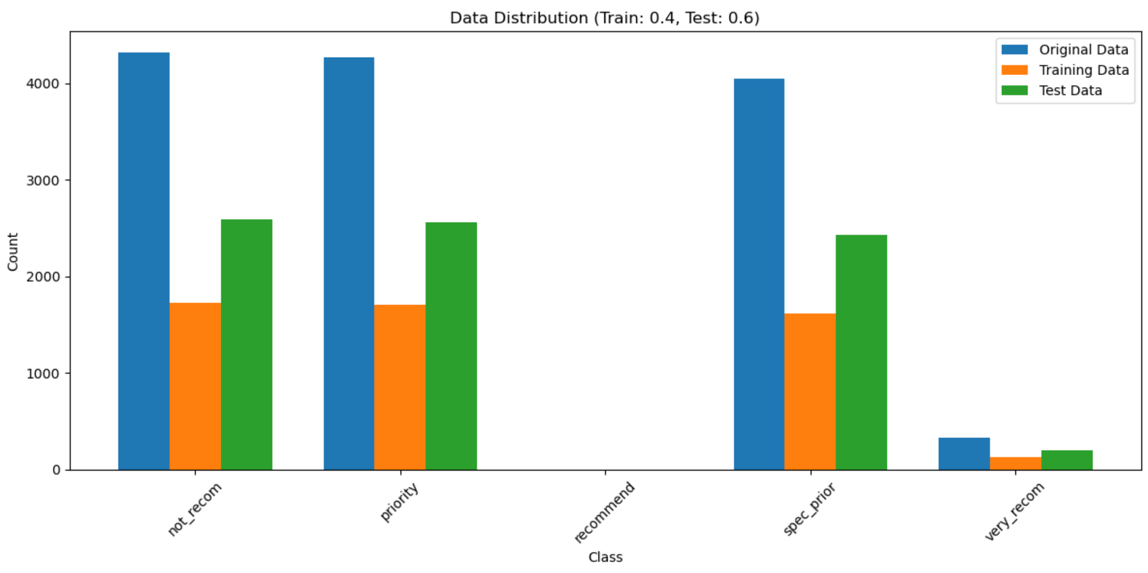


Figure 3: data distribution 40/60 (detail)

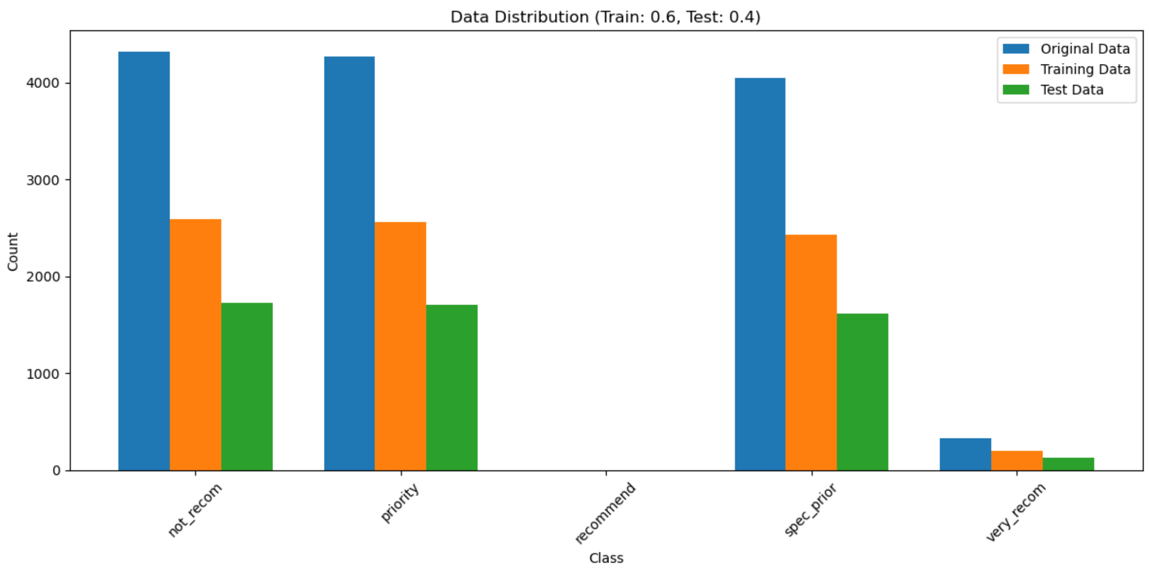


Figure 4: data distribution 60/40 (detail)

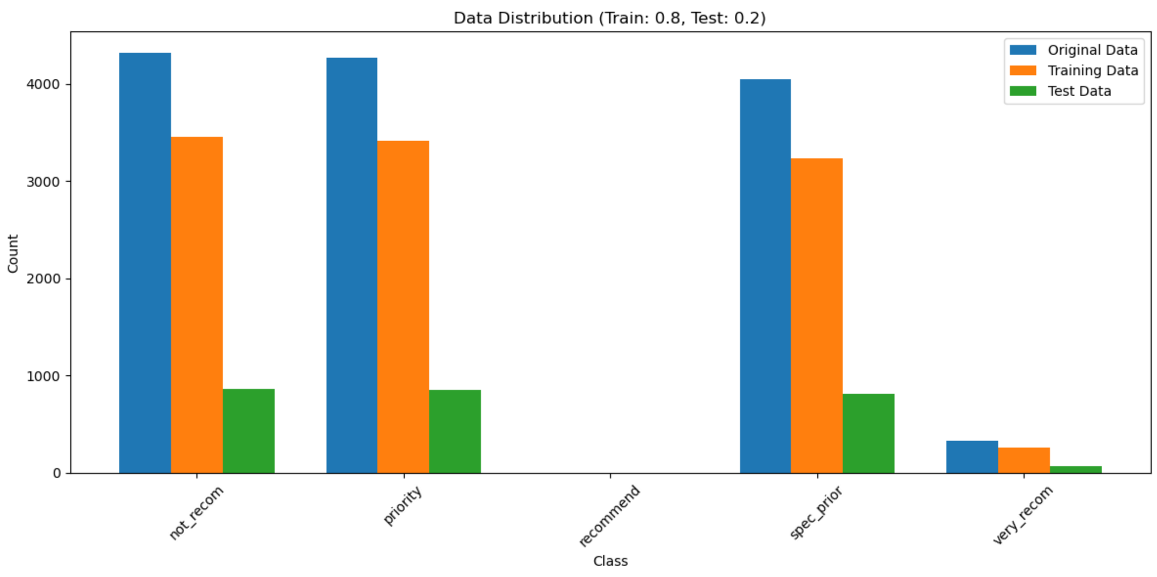


Figure 5: data distribution 80/20 ([detail](#))

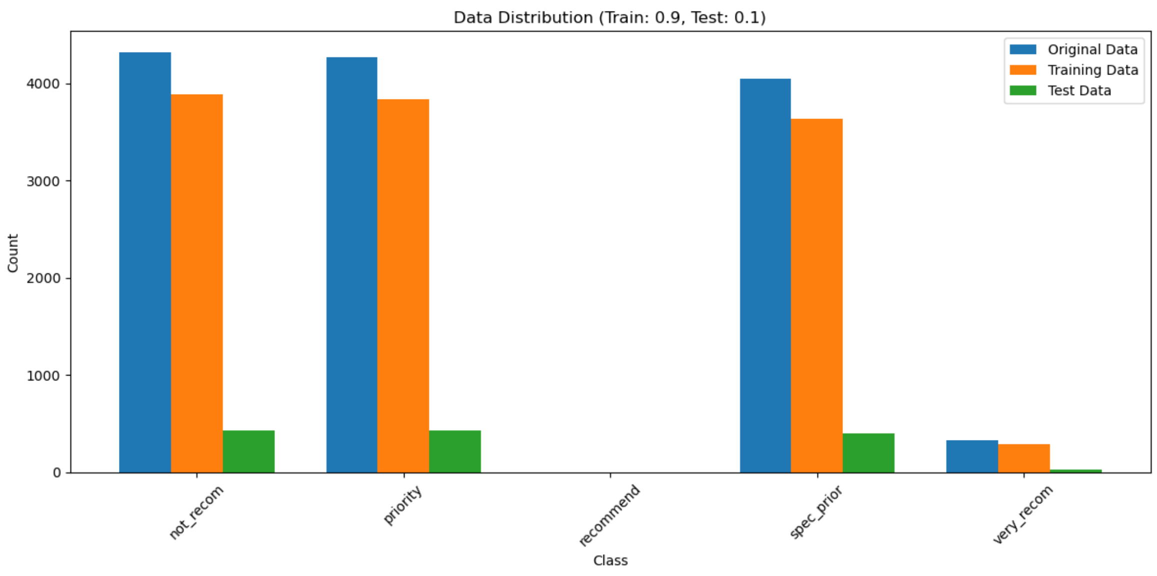


Figure 6: data distribution 90/10 ([detail](#))

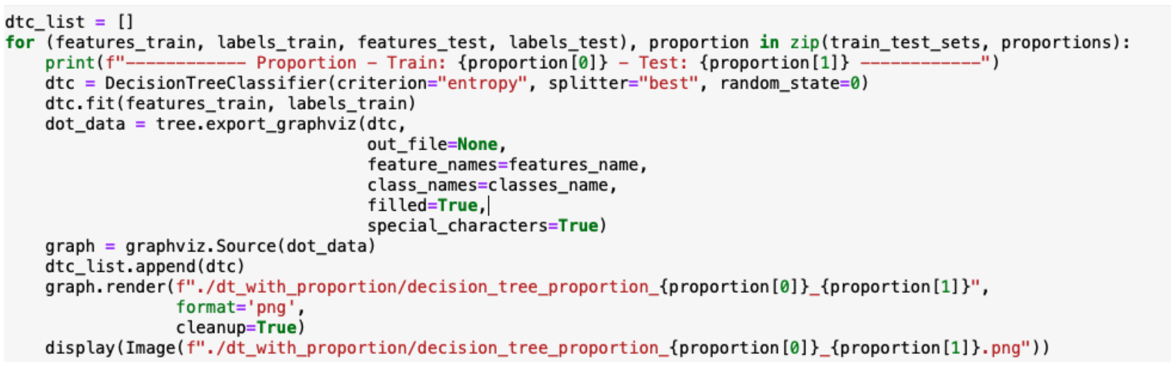
4. Building the decision tree classifiers

The below code segment applies decision tree modeling to multiple sets of training and test data, using varying proportions for data division. It creates a 'dtc_list' to store the resulting decision tree classifiers.

For each pair of training and test sets, the code constructs a decision tree classifier using the DecisionTreeClassifier class from scikit-learn. The classifier is configured with the criterion 'entropy' and the splitter 'best' to ensure optimal splitting decisions. The 'random_state' parameter is set to 0 to enable result reproducibility.

Once the classifier is fitted to the training data, the code generates a graphical representation of the decision tree using the 'export_graphviz' function from the tree module. This visualization captures the decision tree's structure, incorporating feature and class names. The resulting graph is rendered using graphviz and saved as a PNG file.

Lastly, the code employs the Image class to display the generated decision tree image, facilitating immediate visualization within environments such as Jupyter Notebook.



```python
dtc_list = []
for (features_train, labels_train, features_test, labels_test), proportion in zip(train_test_sets, proportions):
    print(f"----------- Proportion - Train: {proportion[0]} - Test: {proportion[1]} -----------")
    dtc = DecisionTreeClassifier(criterion="entropy", splitter="best", random_state=0)
    dtc.fit(features_train, labels_train)
    dot_data = tree.export_graphviz(dtc,
                                    out_file=None,
                                    feature_names=features_name,
                                    class_names=classes_name,
                                    filled=True,
                                    special_characters=True)
    graph = graphviz.Source(dot_data)
    dtc_list.append(dtc)
    graph.render(f"./dt_with_proportion/decision_tree_proportion_{proportion[0]}_{proportion[1]}",
                 format='png',
                 cleanup=True)
    display(Image(f"./dt_with_proportion/decision_tree_proportion_{proportion[0]}_{proportion[1]}.png"))
```

Figure 7: building classifiers code (detail)

The result outputs:

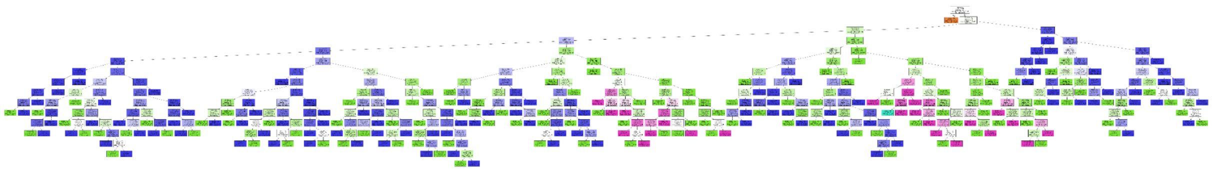


Figure 8: decision tree with data set 40/60 (detail)

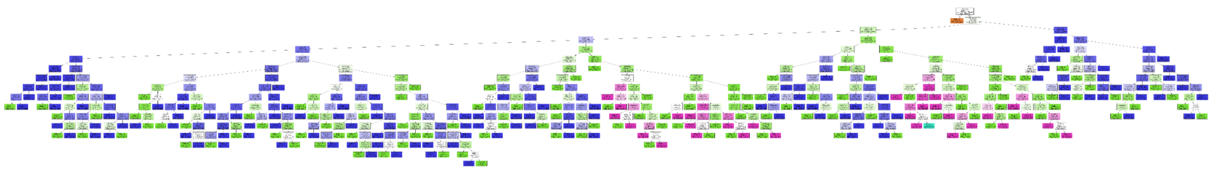


Figure 9: decision tree with data set 60/40 (detail)

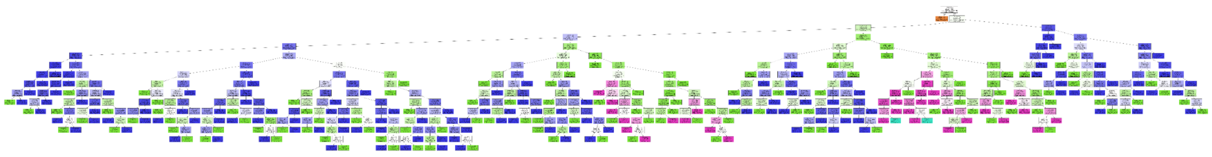


Figure 10: decision tree with data set 80/20 (detail)

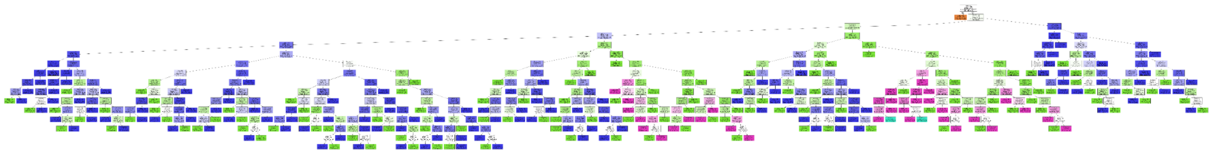


Figure 11: decision tree with data set 90/10 (detail)

5. Evaluating the decision tree classifiers

5.1. Classification report and confusion matrix

The classification report and confusion matrix play crucial roles in assessing the effectiveness of a classification model.

The confusion matrix is a tabular representation that summarizes the model's predictions on a test dataset, highlighting the number of correct and incorrect predictions. It consists of four values: true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). TP and TN indicate the correctly classified positive and negative samples, respectively. On the other hand, FP and FN represent the misclassified positive and negative samples. By analyzing the confusion matrix, we can calculate performance metrics like accuracy, precision, recall, and F1 score.

The classification report provides a concise overview of these performance metrics for each class within the dataset. It includes metrics such as precision, recall, F1 score, and support. Precision assesses the proportion of accurate positive predictions, while recall measures the proportion of actual positives correctly identified by the model. The F1 score is a balanced combination of precision and recall, taking both metrics into account. Support indicates the number of samples in each class.

Together, classification report and confusion matrix offer a comprehensive evaluation of the model's performance. They enable us to pinpoint areas that require improvement and make informed decisions regarding adjustments to the model's parameters or features.

The result outputs:

```
Classification Report - Proportion: (0.4, 0.6)
              precision    recall  f1-score   support

   not_recom       1.00      1.00      1.00      2592
    priority       0.97      0.98      0.98      2560
   recommend       0.00      0.00      0.00         1
  spec_prior       0.98      0.98      0.98      2426
  very_recom       0.95      0.91      0.93       197

    accuracy                           0.98      7776
   macro avg       0.78      0.77      0.78      7776
weighted avg       0.98      0.98      0.98      7776

Confusion Matrix:
[[2592    0    0    0    0]
 [   0 2508    0   44    8]
 [   0    0    0    0    1]
 [   0   54    0 2372    0]
 [   0   14    3    0  180]]
```

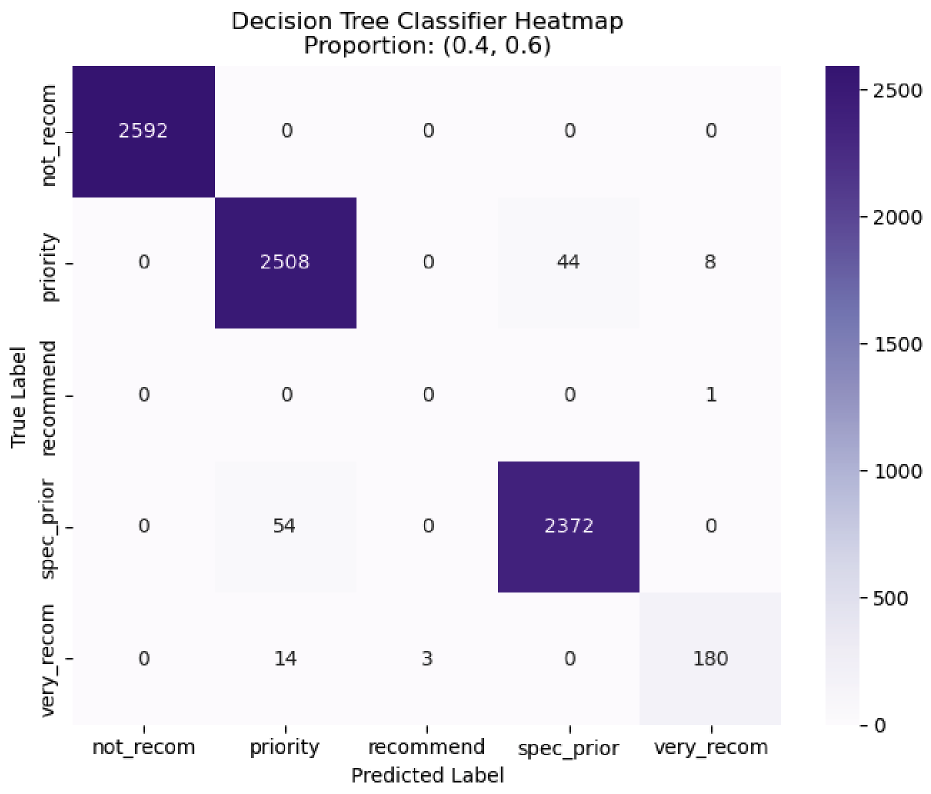


Figure 12: classification report and confusion matrix with data set 40/60 (detail)

```
Classification Report – Proportion: (0.6, 0.4)
              precision    recall  f1-score   support

   not_recom       1.00      1.00      1.00      1728
    priority       0.98      0.99      0.99      1706
   recommend       1.00      0.00      0.00         1
  spec_prior       0.99      0.99      0.99      1618
  very_recom       0.98      0.98      0.98       131

    accuracy                           0.99      5184
   macro avg       0.99      0.79      0.79      5184
weighted avg       0.99      0.99      0.99      5184

Confusion Matrix:
[[1728    0    0    0    0]
 [   0 1683    0   21    2]
 [   0    0    0    0    1]
 [   0   24    0 1594    0]
 [   0    3    0    0  128]]
```

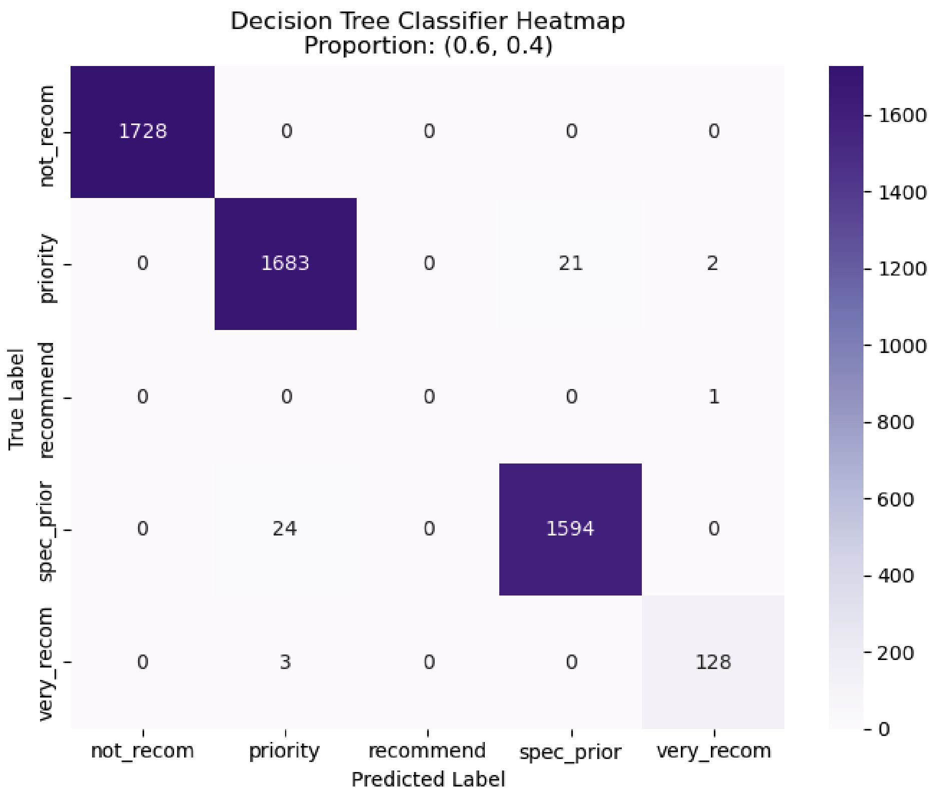


Figure 13: classification report and confusion matrix with data set 60/40 (detail)

```
Classification Report - Proportion: (0.8, 0.2)
              precision    recall  f1-score   support

   not_recom       1.00      1.00      1.00       864
    priority       0.99      1.00      0.99       853
   recommend       0.00      1.00      0.00         0
  spec_prior       1.00      1.00      1.00       809
  very_recom       1.00      0.95      0.98        66

    accuracy                           1.00      2592
   macro avg       0.80      0.99      0.79      2592
weighted avg       1.00      1.00      1.00      2592

Confusion Matrix:
[[864   0   0   0   0]
 [  0 849   0   4   0]
 [  0   0   0   0   0]
 [  0   4   0 805   0]
 [  0   2   1   0  63]]
```

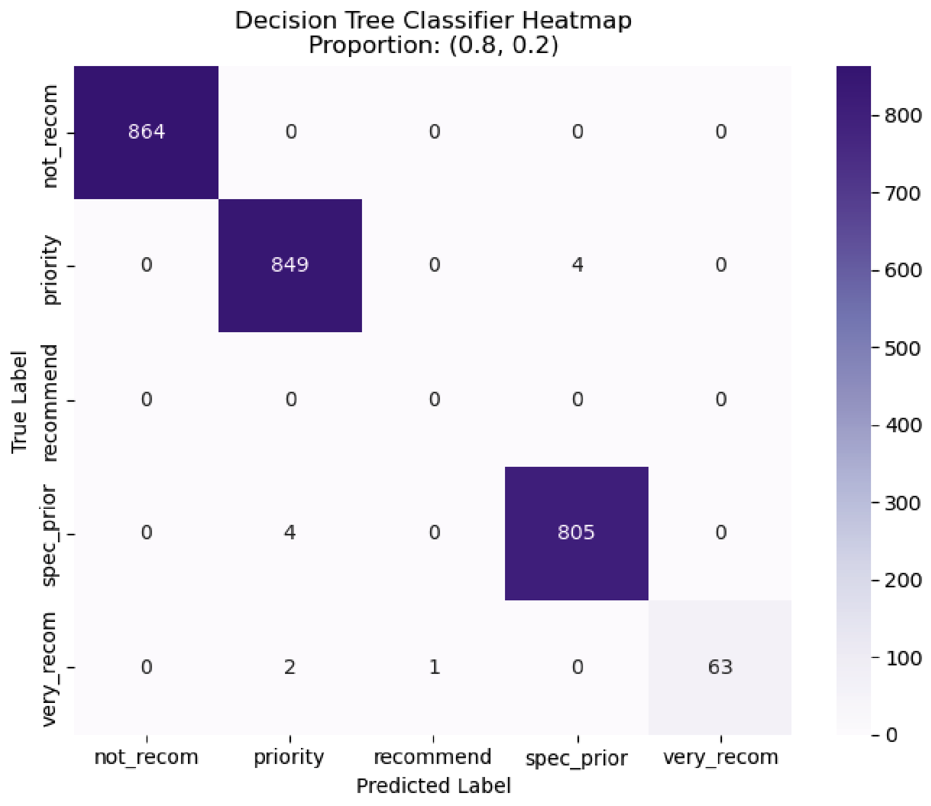


Figure 14: classification report and confusion matrix with data set 80/20 (detail)

```
Classification Report - Proportion: (0.9, 0.1)
               precision    recall  f1-score   support

   not_recom       1.00      1.00      1.00       432
    priority       1.00      1.00      1.00       427
   recommend       0.00      1.00      0.00         0
  spec_prior       1.00      1.00      1.00       404
  very_recom       1.00      0.97      0.98        33

    accuracy                           1.00      1296
   macro avg       0.80      0.99      0.79      1296
weighted avg       1.00      1.00      1.00      1296

Confusion Matrix:
[[432   0   0   0   0]
 [  0 425   0   2   0]
 [  0   0   0   0   0]
 [  0   2   0 402   0]
 [  0   0   1   0  32]]
```

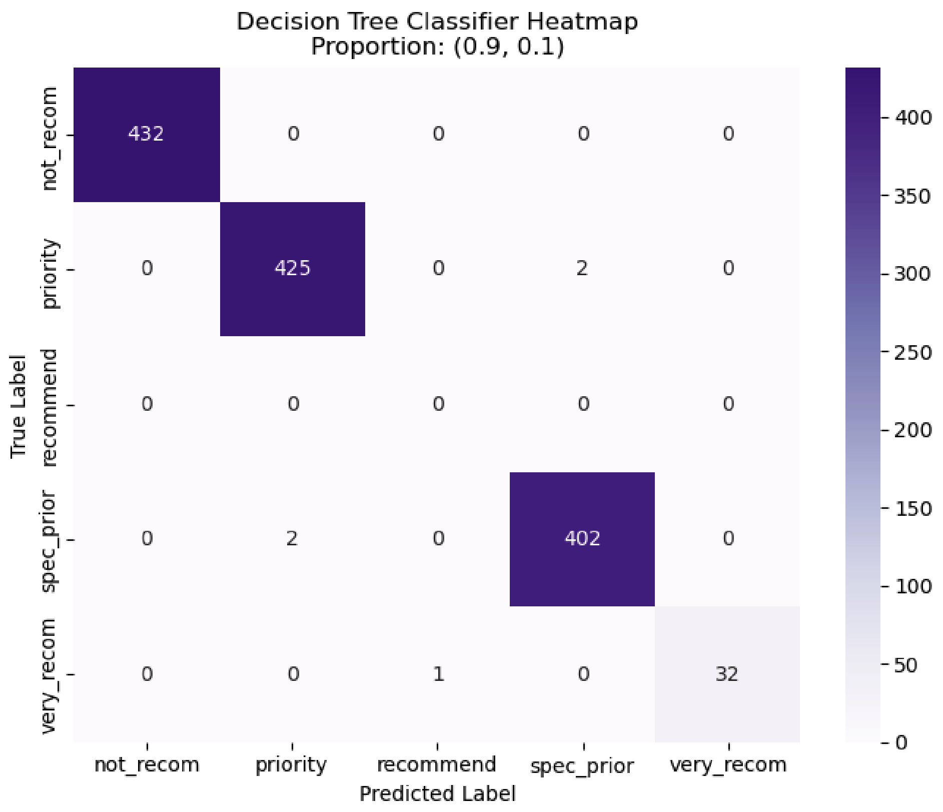


Figure 15: classification report and confusion matrix with data set 90/10 (detail)

5.2. Comments

- Proportion 40/60:
  - Classification Report: The model achieves high accuracy and performs well for most classes, except for the "recommend" class which has precision, recall, and F1-score values of 0 due to no instances being correctly classified as "recommend".
  - Confusion Matrix: Most instances are correctly classified for most classes, but there are some misclassifications between the "priority" and "spec_prior" classes.

- Proportion 60/40:
  - Classification Report: The model achieves high accuracy and performs well for most classes, except for the "recommend" class which has precision, recall, and F1-score values of 0 due to no instances being correctly classified as "recommend".
  - Confusion Matrix: Most instances are correctly classified for most classes, but there are some misclassifications between the "priority" and "spec_prior" classes.
- Proportion 80/20:
  - Classification Report: The model achieves perfect accuracy and performs well for most classes, except for the "recommend" class which has precision and F1-score values of 0 due to no instances being correctly classified as "recommend".
  - Confusion Matrix: Most instances are correctly classified for most classes, but there are a few misclassifications between the "priority" and "spec_prior" classes.
- Proportion 90/10:
  - Classification Report: The model achieves perfect accuracy and performs well for most classes, except for the "recommend" class which has precision, recall, and F1-score values of 0 due to no instances being correctly classified as "recommend".
  - Confusion Matrix: Most instances are correctly classified for most classes, but there are some misclassifications between the "priority" and "spec_prior" classes.

In summary, the model consistently struggles to predict the "recommend" class due to the lack of training examples. It performs well for other classes but occasionally confuses the "priority" and "spec_prior" classes. The overall accuracy is high, but improvements are needed to accurately classify the underrepresented "recommend" class.

6. The depth and accuracy of a decision tree

6.1. Trees, table, and chart

- Table:

Max depth	Accuracy
2	0.7692901234567902
3	0.8082561728395061

4	0.8572530864197531
5	0.8792438271604939
6	0.8892746913580247
7	0.9216820987654321
None	0.9949845679012346

- Chart:

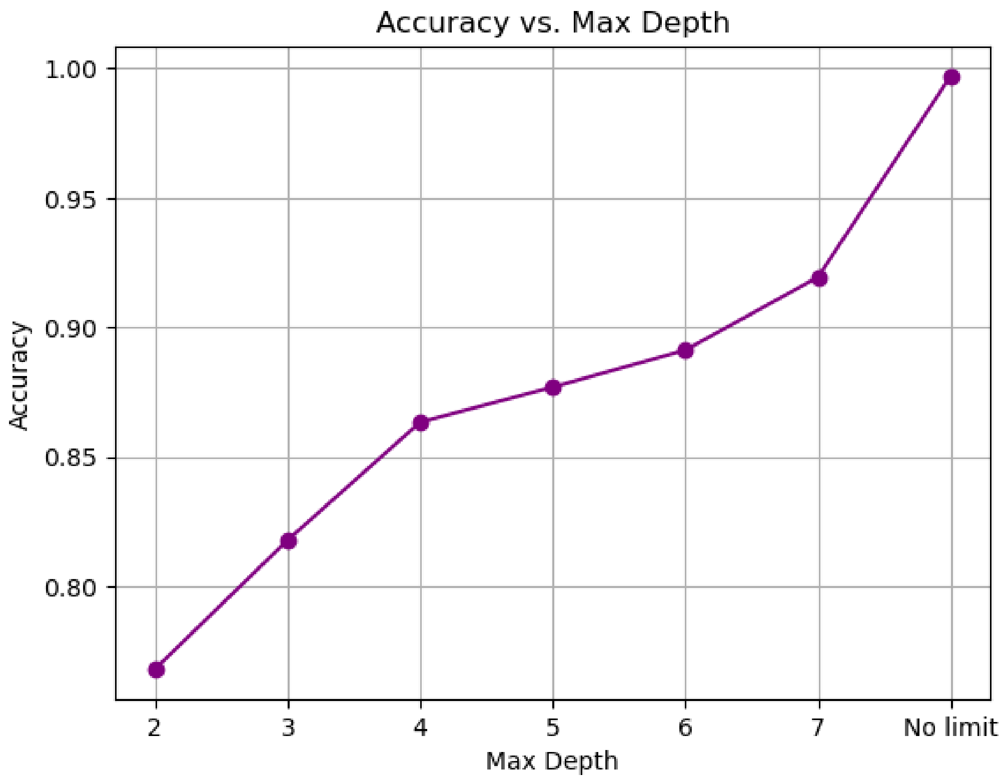


Figure 16: max depth and corresponding accuracy (detail)

- Trees:

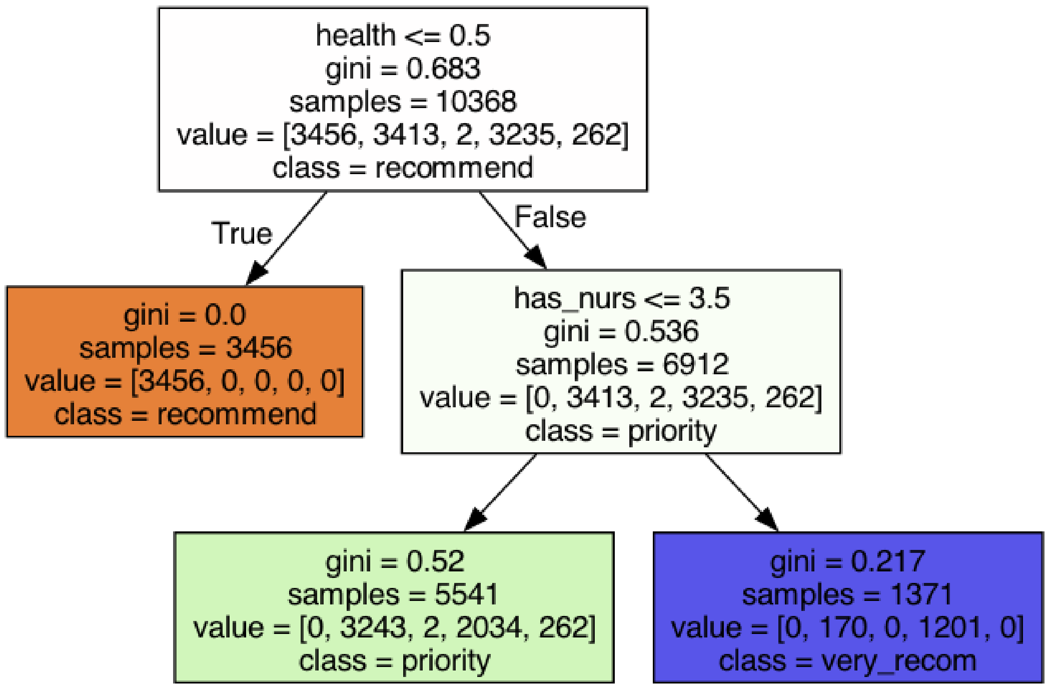


Figure 17: Decision tree with max depth 2 (detail)

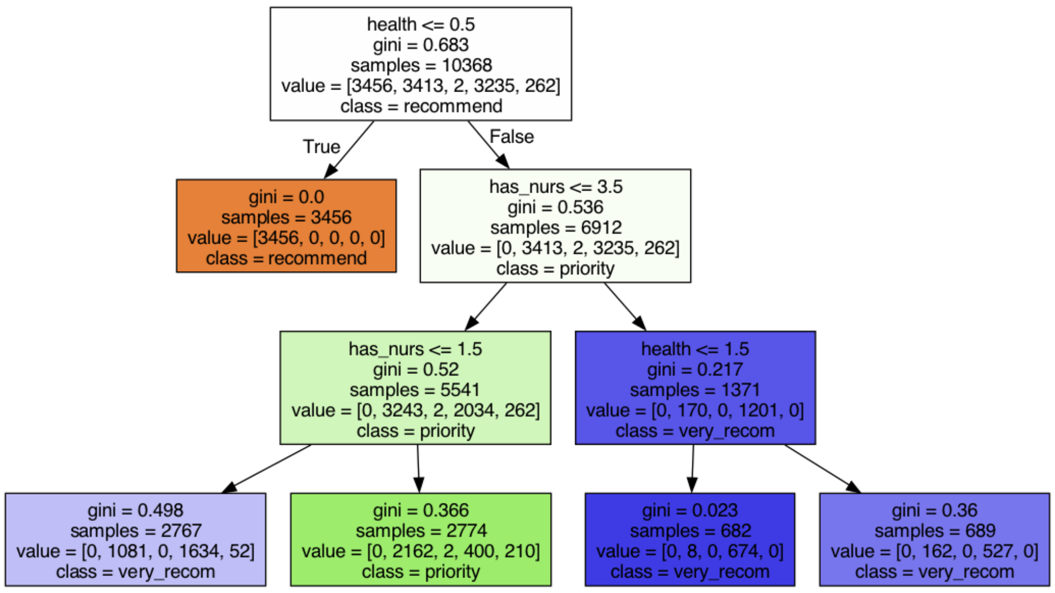


Figure 17: Decision tree with max depth 3 (detail)

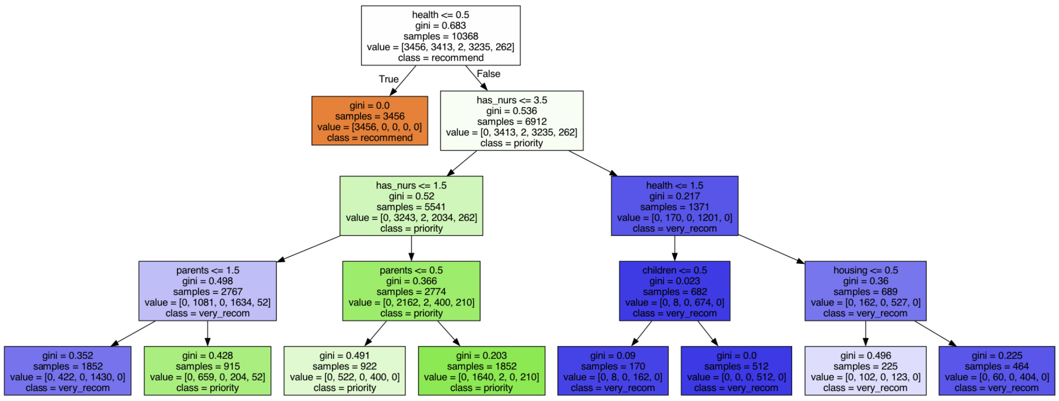


Figure 17: Decision tree with max depth 4 (detail)

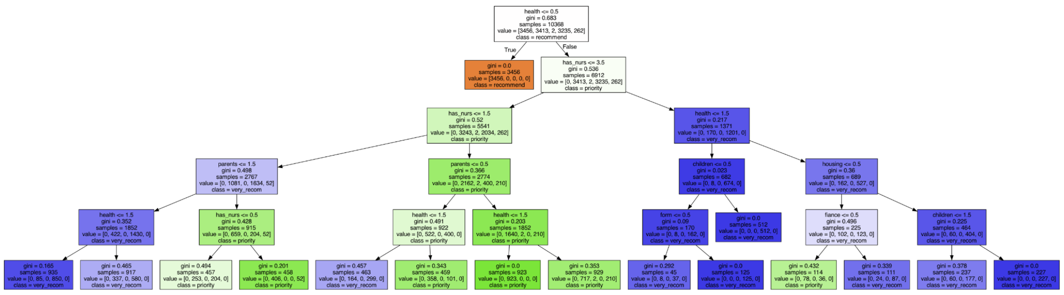


Figure 17: Decision tree with max depth 5 (detail)

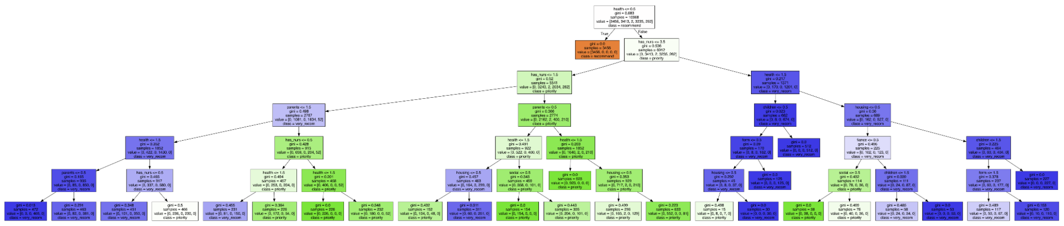


Figure 17: Decision tree with max depth 6 (detail)

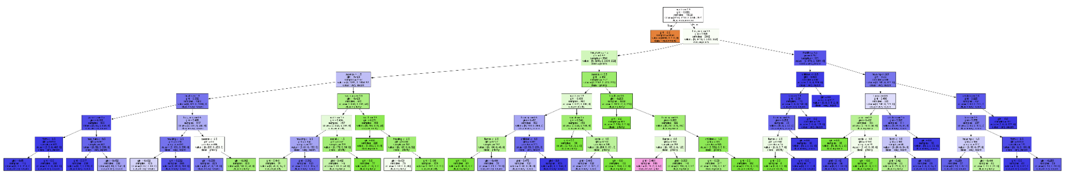


Figure 17: Decision tree with max depth 7 (detail)

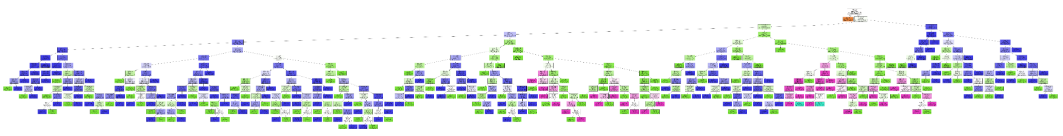


Figure 17: Decision tree with max depth None (detail)

6.2. Comments

The table and graph illustrate that the decision tree classifier's accuracy is affected by the max_depth parameter. When max_depth is unrestricted (None), the accuracy achieves an impressive value of 0.9949, surpassing all other tested values. This indicates that a decision tree without depth limitations performs exceptionally well on the dataset.

Moreover, increasing the max_depth from 2 to 7 consistently enhances accuracy, with values gradually improving from 0.7692 to 0.9217. This implies that deeper decision trees can capture more intricate patterns and relationships, resulting in improved predictions.

Additionally, the chart demonstrates a steady increase in accuracy as max_depth increases, accompanied by reduced fluctuations as it approaches 1. This suggests that selecting an appropriate max_depth level can yield relatively high accuracy, particularly in situations with limited resources.

These findings highlight the effectiveness of deeper decision trees in this specific scenario. However, it is crucial to consider dataset characteristics and the specific problem when determining the optimal max_depth value. While higher max_depth values can improve accuracy, there is a risk of overfitting if the model becomes overly complex.

Therefore, striking a balance between model complexity and performance is essential by selecting an appropriate max_depth value that avoids both underfitting and overfitting. Experimentation and evaluation on an independent test set can help identify the optimal max_depth value for this decision tree classifier.

Reference

[1] Graphiz: link

[2] Scikit-learn: link

[3] Decision Tree Classifiers tutorial: link

–The end–