UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

Lab 02: Decision Tree with scikit-learn

Subject: Introduction to Artificial Intelligence

Class: 21CLC03

Students: Trần Nguyên Huân

MSSV: 21127050

Instructors: Nguyễn Ngọc Thảo, Lê Ngọc Thành,
Nguyễn Trần Duy Minh, Nguyễn Hải Đăng

HO CHI MINH CITY – 2023

Contents

I. Evaluating the level of accomplishment

No.	Criteria	Completed
1	Preparing the data sets	✓
2	Evaluating the decision tree classifiers	✓
3	Evaluating the decision tree classifiers	
	Classification report and confusion matrix	✓
	Comments	✓
4	The depth and accuracy of a decision tree	
	Trees, tables, and charts	✓
	Comments	✓

II. Preparing the datasets

By using Excel, we can manually merge the two files and create a CSV file:

1. Open both the files, pokerhand-training-true.data and poker-hand-testing.data, in Excel.
2. Copy the entire data from the poker-hand-testing.data file and paste it at the end of the poker-hand-training-true.data file.
3. Save the merged data file as a CSV file by selecting "Save As" and choosing "CSV" as the file format.
4. Name the merged CSV file as "poker-hand-data.csv" and save it in your desired location.

➔ With these steps, you will have a single CSV file containing the merged data from both files.

```python
# Load the merged data into a pandas DataFrame
data = pd.read_csv('poker-hand-data.csv', header=None)

# Split the DataFrame into features and labels
features = data.iloc[:, :-1]
labels = data.iloc[:, -1]

# Shuffle the data randomly
data = data.sample(frac=1)

# Create 16 subsets with different training/test proportions
proportions = [(0.4, 0.6), (0.6, 0.4), (0.8, 0.2), (0.9, 0.1)]
subsets = []
for train_prop, test_prop in proportions:
    feature_train, feature_test, label_train, label_test = train_test_split(features, labels, train_size=train_prop,
                                                                            test_size=test_prop, stratify=labels)
    subsets.append((feature_train, label_train, feature_test, label_test))
```

✓ 2.1s Python

In this step, this code above loads the **"poker-hand-data.csv"** dataset into a Pandas DataFrame called **"data"**. The dataset consists of features (first 10 columns) and labels (last column), where each row represents a poker hand, and the label represents the type of the hand.

Then, the dataset is shuffled randomly using the **sample()** function in Pandas.

The code creates 16 different training/test subsets using the **train_test_split()** function from scikit-learn with different proportions of training and test data specified in the proportions list. For each proportion, the function splits the data into training and test sets and stratifies the data based on the labels. The resulting subsets are stored in a list called subsets, where each element is a tuple containing the feature and label data for both the training and test sets.

Finally, visualizing the distributions of classes in all data sets (the original set, training set, and test set) of all proportions.

Here is example of original data (Image II-1), training set and test set of proportion 40/60 (represent proportion: 0.4,0.6 in column chart)

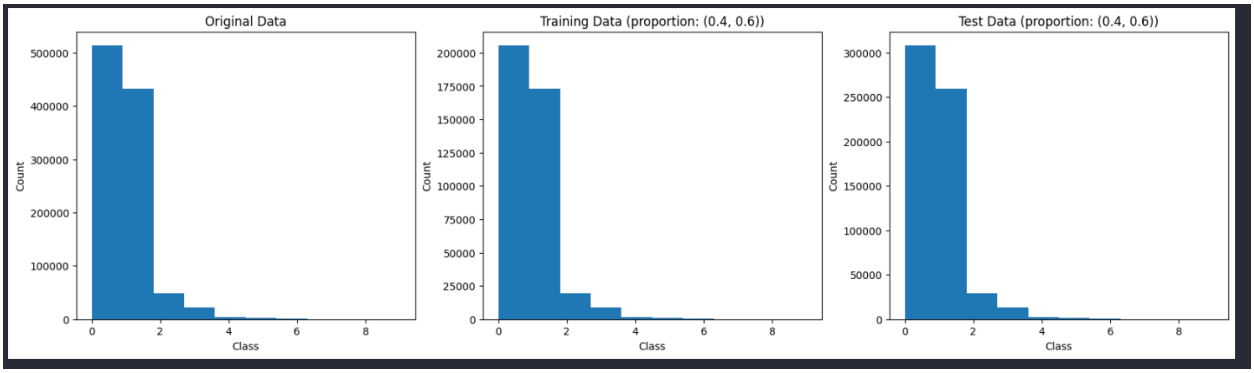


Image II-1. The distribution of class of proportion 40/60

III. Building the decision tree classifiers

The code is using the **DecisionTreeClassifier** from the **scikit-learn** library to train a decision tree model on the poker hand dataset.

```
X=data.drop(['Poker Hand'], axis=1)
Y=data['Poker Hand']
```
✓ 0.0s Python

To visualize the resulting decision tree classifiers tree, The dataset is first split into two parts, **X** containing the features (all columns except the last column containing the target variable), and **Y** containing the target variable.

```python
for train_prop, test_prop in proportions:
    feature_train, feature_test, label_train, label_test = train_test_split(features, labels, train_size=train_prop,
                                                                            test_size=test_prop, stratify=labels)
    clf = DecisionTreeClassifier(criterion='entropy', max_depth=7)
    clf = clf.fit(feature_train, label_train)
    dot_data = export_graphviz(clf, out_file=None,
                               feature_names=X.columns, class_names=X.columns,
                               filled=True, rounded=True,
                               special_characters=True)
    print(f"Decision Tree Classifier with proportion {int(train_prop*100)}/{int(test_prop*100)}")
    graph = graphviz.Source(dot_data)
    display(graph)
```

✓ 15.2s Python

The script then loops over a set of proportions (specified earlier in the code) for the training and test sets, and for each set of proportions, it further splits the data into training and test sets using **train_test_split** function from scikit-learn library. The function randomly splits the dataset into a training set and a test set based on the specified proportions, with stratification on the target variable to ensure that each class is represented equally in both the training and test sets.

Next, the **DecisionTreeClassifier** is instantiated with the entropy criterion and a depth of 7. The model is then fit to the training data using **clf.fit(feature_train, label_train).**

After the model is trained, the **export_graphviz** function from the **graphviz** library is used to create a graph representation of the decision tree model. The resulting graph is then displayed using the **display(graph)** function.

* The Decision Tree Classifiers for each proportion was represented directly in ipynb file.

IV. Evaluating the decision tree classifiers

1. Overview about Classification report and confusion matrix

Classification report and confusion matrix are commonly used tools to evaluate the performance of a classification model, such as a decision tree classifier.

The **confusion matrix** provides a tabular summary of the predictions made by the classifier, compared to the actual labels. It contains four values: true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). The rows correspond to the actual labels, while the columns correspond to the predicted labels. This matrix helps us to identify the types of errors that the classifier is making.

The **classification report** is a summary of the precision, recall, F1-score, and support for each class in the classification problem. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. Recall is the ratio of correctly predicted positive observations to the

total actual positive observations. The F1-score is a weighted average of precision and recall, where F1-score reaches its best value at 1 and worst at 0. Support is the number of samples of the true response that lies in that class.

2. Classification report and confusion matrix in my program

```
class_labels = ['Nothing', 'One Pair', 'Two Pairs', 'Three of a Kind', 'Straight', 'Flush', 'Full House', 'Four of a Kind', 'Straight Flush', 'Royal
for train_size, test_size in proportions:
    # Split data into train and test sets
    feature_train, feature_test, label_train, label_test = train_test_split(features, labels, train_size=train_size,
                                                    test_size=test_size, stratify=labels, shuffle=True)

    # Fit decision tree classifier to training set
    clf = DecisionTreeClassifier(criterion='entropy')
    clf.fit(feature_train, label_train)

    # Predict test set labels
    label_pred = clf.predict(feature_test)

    # Generate classification report and confusion matrix
    print("Decision Tree Classifier report:")
    print(f"Train/Test Proportions: {train_size}/{test_size}")
    print(classification_report(label_test, label_pred, labels=range(10), target_names=class_labels))
    cfm = confusion_matrix(label_test, label_pred)
    plt.figure(figsize=(9, 9))
    sns.heatmap(cfm, annot=True, fmt='d', linewidths=.5)
    plt.title('Decision Tree Classifier confusion matrix')
    plt.xlabel('Predicted Label')
    plt.ylabel('Actual Label')
    plt.show()
```

This code trains a decision tree classifier on different subsets of the poker hand dataset and evaluates its performance on a test set using a confusion matrix and a classification report.

The loop iterates over the proportions list, which contains tuples specifying the proportion of data to be used for training and testing. For each iteration, the code uses the **train_test_split()** function to split the dataset into training and test sets according to the specified proportions. Then, it fits a decision tree classifier with entropy criterion to the training set using the **DecisionTreeClassifier()** function and predicts the labels of the test set using the **predict()** method.

The classification report and confusion matrix are generated using the **classification_report()** and **confusion_matrix()** functions from the scikit-learn library. The **classification_report()** function displays a summary of precision, recall, F1-score, and support for each class, while the **confusion_matrix()** function creates a matrix showing the counts of true and false predictions for each class.

Finally, the code displays the classification report and confusion matrix for each iteration using **print()** and **sns.heatmap()** functions. The **plt.figure()** function creates a new figure for each confusion matrix, and **sns.heatmap()** generates a heatmap of the confusion matrix with annotations

indicating the counts of true and false predictions. The **plt.title()**, **plt.xlabel()**, and **plt.ylabel()** functions set the title and labels of the confusion matrix plot.

2. Comments

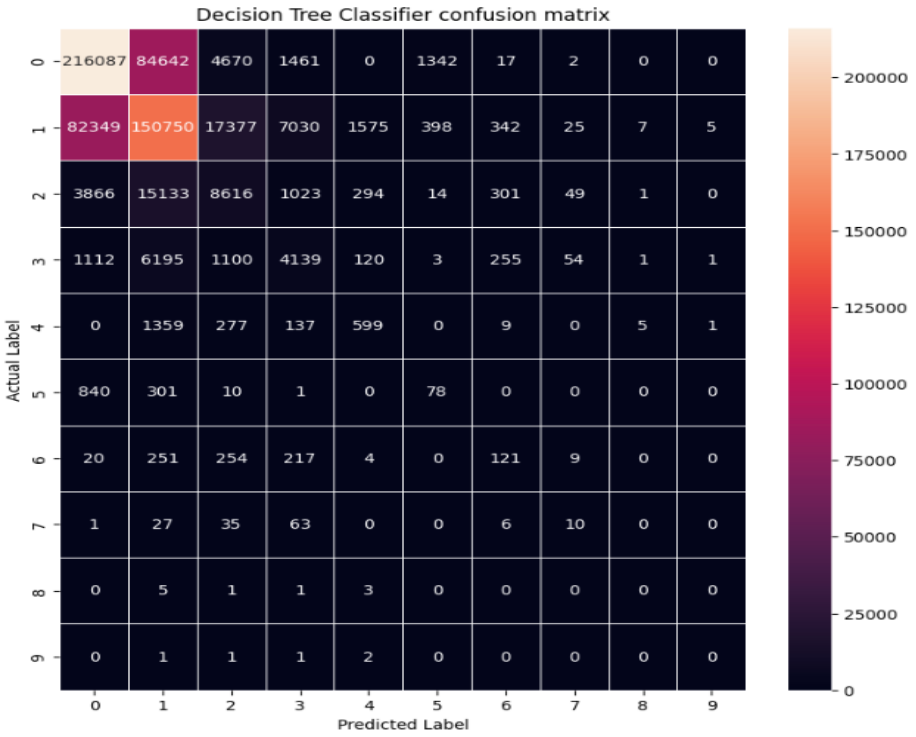
To measure the performance of a decision tree classifier, we can analyze the confusion matrix and classification report. A good classifier should have high values of precision, recall, and F1-score, and a high accuracy rate. A high precision value indicates that the model has a low false positive rate, while a high recall value indicates that the model has a low false negative rate. The F1-score provides a balance between precision and recall, and the accuracy rate indicates the proportion of correctly classified samples. We can also use the confusion matrix to calculate other metrics, such as the specificity and sensitivity of the model.

Here is the figure of classification report and confusion matrix measured for each decision tree classifiers in the corresponding test set.

- Decision Tree Classifiers With Proposion 40/60

Decision Tree Classifier report:
Proportion 40/60

	precision	recall	f1-score	support
Nothing	0.71	0.70	0.71	308221
One Pair	0.58	0.58	0.58	259858
Two Pairs	0.27	0.29	0.28	29297
Three of a Kind	0.29	0.32	0.31	12980
Straight	0.23	0.25	0.24	2387
Flush	0.04	0.06	0.05	1230
Full House	0.12	0.14	0.13	876
Four of a Kind	0.07	0.07	0.07	142
Straight Flush	0.00	0.00	0.00	10
Royal Flush	0.00	0.00	0.00	5
accuracy			0.62	615006
macro avg	0.23	0.24	0.24	615006
weighted avg	0.62	0.62	0.62	615006

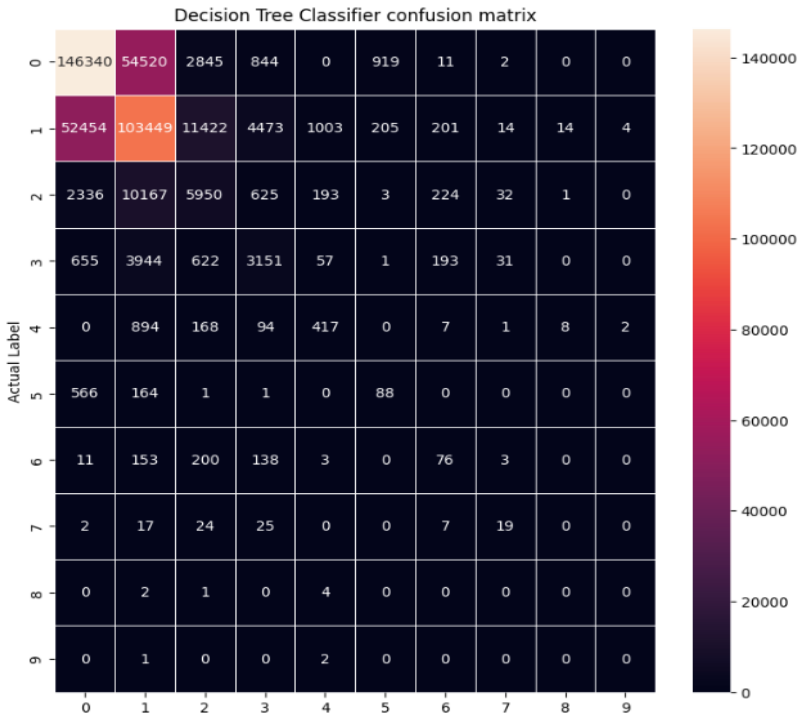


- Decision Tree Classifiers With Proposion 60/40

```
Decision Tree Classifier report:
Proportion 60/40
                precision    recall  f1-score   support

       Nothing       0.72      0.71      0.72    205481
      One Pair       0.60      0.60      0.60    173239
     Two Pairs       0.28      0.30      0.29     19531
Three of a Kind      0.34      0.36      0.35      8654
      Straight       0.25      0.26      0.26      1591
         Flush       0.07      0.11      0.09       820
    Full House       0.11      0.13      0.12       584
 Four of a Kind      0.19      0.20      0.19        94
 Straight Flush      0.00      0.00      0.00         7
    Royal Flush      0.00      0.00      0.00         3

      accuracy                           0.63    410004
     macro avg       0.25      0.27      0.26    410004
  weighted avg       0.64      0.63      0.63    410004
```

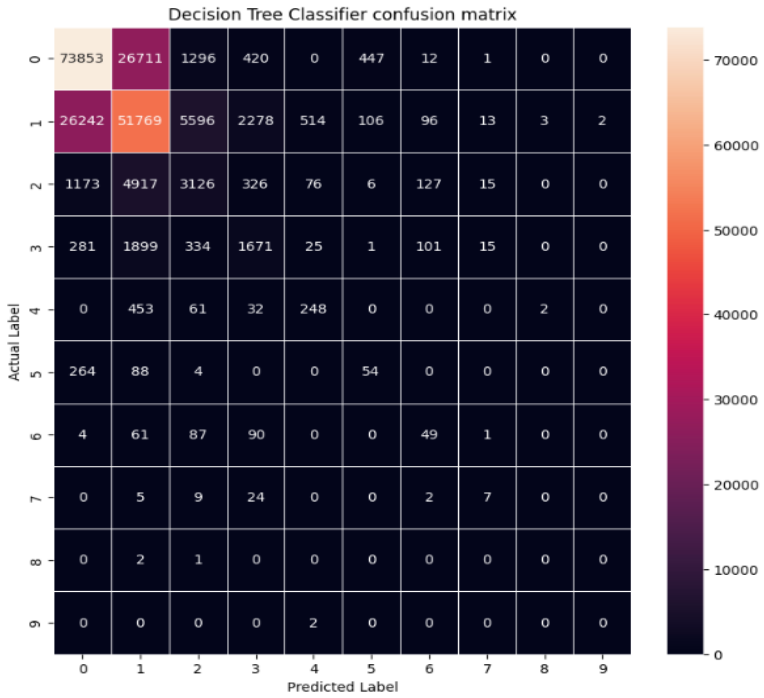


- Decision Tree Classifiers With Proposion 80/20

```
Decision Tree Classifier report:
Proportion 80/20
                precision    recall  f1-score   support

       Nothing       0.73      0.72      0.72    102740
      One Pair       0.60      0.60      0.60     86619
     Two Pairs       0.30      0.32      0.31      9766
Three of a Kind       0.35      0.39      0.36      4327
      Straight       0.29      0.31      0.30       796
         Flush       0.09      0.13      0.11       410
    Full House       0.13      0.17      0.14       292
 Four of a Kind       0.13      0.15      0.14        47
 Straight Flush       0.00      0.00      0.00         3
    Royal Flush       0.00      0.00      0.00         2

      accuracy                           0.64    205002
     macro avg       0.26      0.28      0.27    205002
  weighted avg       0.64      0.64      0.64    205002
```

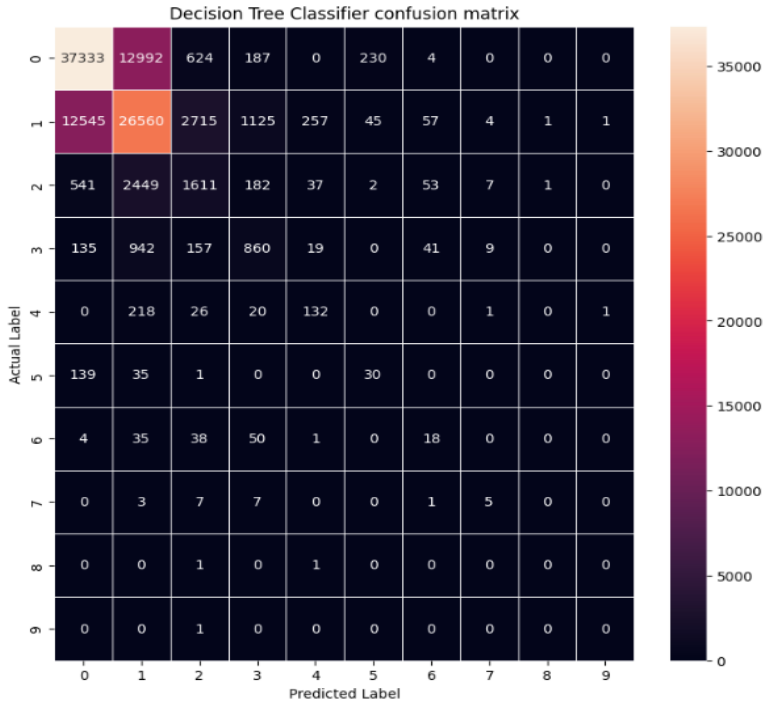


- Decision Tree Classifiers With Proposion 90/10

```
Decision Tree Classifier report:
Proportion 90/10
                precision    recall  f1-score   support

       Nothing       0.74      0.73      0.73     51370
      One Pair       0.61      0.61      0.61     43310
     Two Pairs       0.31      0.33      0.32      4883
Three of a Kind       0.35      0.40      0.37      2163
      Straight       0.30      0.33      0.31       398
         Flush       0.10      0.15      0.12       205
    Full House       0.10      0.12      0.11       146
 Four of a Kind       0.19      0.22      0.20        23
 Straight Flush       0.00      0.00      0.00         2
    Royal Flush       0.00      0.00      0.00         1

      accuracy                           0.65    102501
     macro avg       0.27      0.29      0.28    102501
  weighted avg       0.65      0.65      0.65    102501
```



➔ Looking at the confusion matrix and classfication_report of following Decision Tree Classiers the value of precision, recall, F1-score and a accuracy in desceding order of proportion is 90/10, 80/20, 60/40, 40/60. So the good level in order: 90/10 > 80/20 > 60/40 > 40/60.

❖ Overall, a good decision tree classifier should have a high accuracy rate and low values of false positives and false negatives. The classification report and confusion matrix provide valuable insights into the model's performance and can be used to fine-tune the model parameters for optimal performance.

V. The depth and accuracy of a decision tree

1. About this code of this specifications

```
from sklearn.metrics import accuracy_score
# Define list of max_depth values to try
max_depths = [None, 2, 3, 4, 5, 6, 7]
accuracy_scores = []


feature_train, feature_test, label_train, label_test = train_test_split(features, labels, train_size=0.8, test_size=0.2, stratify=labels)

# Loop over max_depth values
for max_depth in max_depths:
    # Create and fit decision tree classifier on training set
    clf = DecisionTreeClassifier(criterion='entropy', max_depth=max_depth)
    clf.fit(feature_train, label_train)

    # Make predictions on test set and calculate accuracy
    y_pred = clf.predict(feature_test)

    accuracy_scores.append(accuracy_score(label_test, y_pred))

    #print the accuracy of max_depth
    print("Accuracy of max_depth = {}: {}".format(max_depth, accuracy_score(label_test, y_pred)))
```

The code **from sklearn.metrics import accuracy_score** imports the function **accuracy_score** from the **metrics** module of the **sklearn** library.

The **accuracy_score** function is used to calculate the accuracy of a classification model. It takes two arguments: **y_true** and **y_pred**. **y_true** represents the true labels of the test data, while **y_pred** represents the predicted labels of the test data as predicted by the model.

The function returns a scalar value, which is the accuracy of the model. The accuracy is the ratio of the number of correctly predicted labels to the total number of labels in the test data.

For example, if the test data has 100 instances and the model correctly predicts 85 of them, the accuracy would be 85%.

2. Trees, table and chart

Trees and chart was represented directly in ipynb file.

Here is the table Report to the following table the accuracy_score (on the test set) of the decision tree classifier when changing the value of parameter max_depth.

max_depth	None	2	3	4	5	6	7
Accuracy	0.64022	0.50606	0.50838	0.52546	0.55706	0.55706	0.55736

3. Comments

- The statistics show the accuracy scores for decision tree classifiers with different values of max_depth parameter. The max_depth parameter controls the maximum depth of the decision tree, which can affect the model's ability to capture complex relationships in the data versus overfitting.

- From the statistics, we can observe that as the max_depth increases from 2 to 5, the accuracy also increases, indicating that a deeper tree is able to capture more complex relationships in the data. However, for max_depth values greater than 5, the accuracy levels off or even slightly decreases, which could be a sign of overfitting.

- Overall, the results suggest that a max_depth value of around 5 might be a good choice for this particular dataset, balancing the ability to capture complex relationships without overfitting. However, other factors such as computational complexity and interpretability should also be taken into consideration when choosing the optimal max_depth value for a decision tree classifier.

V. References

- (1) Confusion matrix: https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/#:~:text=A%20Confusion%20matrix%20is%20an,by%20the%20machine%20learning%20model.

- (2) Analysis and classification of Mushrooms: https://www.kaggle.com/haimfeld87/analysis-and-classification-of-mushrooms

- (3) Scikit-learn decision trees: https://scikit-learn.org/stable/modules/tree.html

- (4) Poker Hand Data Set: https://archive.ics.uci.edu/ml/datasets/Poker+Hand