กิจกรรมที่ 4 : Models (K-NN, DecisionTree, Random Forest, MLP)

4.1 : สร้าง Classification Model เพื่อประมาณระดับคุณภาพเมล็ดกาแฟ ด้วย (K-NN, DecisionTree, Random Forest, MLP)

- Library ที่ใช้

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

- อ่านข้อมูลจากไฟล์ "Coffee-modified.csv"
- ใช้ข้อมูลที่เตรียม Data Preparation เช่นเดียวกับ Lab#3
- เตรียมข้อมูล train 70% test 30% (train_test_split())
- K-NN (K-Nearest Neighbor) for Bean Grade Classification Model
  - เตรียมโมเดลพารามิเตอร์ k = [1,3,5,7,9,11,13,15,17,19,25,35] # เลือกทดสอบอย่างน้อย 5 ค่า
  - Initial Model: KNeighborsClassifier()
  - ทำการ Train Model ด้วยพารามิเตอร์ k ที่เลือก พร้อมค่า accuracy_score ของแต่ละพารามิเตอร์ เพื่อทำการ plot bar()
  - ทำการทดสอบโมเดล จาก Best Parameter ที่ได้จากการ train model
  - คำนวนค่า Model Performance พร้อมแสดงผล: Confusion Matrix, Classification Report
- Decision Tree
  - เตรียมโมเดลพารามิเตอร์ ASM_func = ['entropy', 'gini'] ทำทั้ง 2 ฟังก์ชัน, max_depth = [3,4,5,6,10,None] เลือกอย่างน้อย 3 ค่า
  - Initial Model: DecisionTreeClassifier()
  - ทำการ Train Model ด้วยพารามิเตอร์ ASM_func และ max_depth ที่กำหนด พร้อมแสดงค่า accuracy_score ของแต่ละพารามิเตอร์
  - ทำการทดสอบโมเดล จาก Best Parameter สำหรับ entropy และ gini และแสดง Decision Tree ทั้ง 2 โดยใช้ฟังก์ชัน plot_tree()
  - คำนวนค่า Model Performance พร้อมแสดงผล: Confusion Matrix, Classification Report
- Random Forest
  - เตรียมโมเดลพารามิเตอร์ ASM_func = ['entropy', 'gini'], n_estimators = [10, 20, 50, 80, 100] เลือกอย่างน้อย 2 ค่า

- Initial Model: RandomForestClassifier()
- ทำการ Train Model ด้วยพารามิเตอร์ ASM_func และ max_depth ที่กำหนด พร้อมแสดงค่า accuracy_score ของแต่ละพารามิเตอร์
- ทำการทดสอบโมเดล จาก Best Parameter สำหรับ entropy และ gini และแสดง Decision Tree ทั้ง 2 โดยใช้ฟังก์ชัน plot_tree()
- คำนวนค่า Model Performance พร้อมแสดงผล: Confusion Matrix, Classification Report

- Multi-Layer Perceptron (MLP)
  - เตรียมโมเดลพารามิเตอร์ hidden_layer_sizes=(10,10,)
  - Initial Model: MLPClassifier()
  - ทำการ Train Model ด้วยพารามิเตอร์ hidden_layer_sizes ที่กำหนด พร้อมแสดงค่า accuracy_score ของแต่ละพารามิเตอร์
  - คำนวนค่า Model Performance พร้อมแสดงผล: Confusion Matrix, Classification Report
- ทำ Hyperparameter Tuning เพื่อได้ Best parameters
  - โดยกำหนด Search parameters สำหรับ GridSearchCV parameters ดังนี้

No	Model	Parameters
1	K-NN	n_neighhbors = `[1,3,5,7,9,11,13,15,17,19,25,35,45]`
2	Decision Tree	criterion = `['entropy', 'gini']` max_depth = `[4,5,6]` max_features = `['sqrt', 'log2',None]` min_samples_leaf = `[1,2,4]`
3	Random Forest	criterion = `['entropy', 'gini']` max_depth = `[4,5,6]` max_features = `['sqrt', 'log2',None]` min_samples_leaf = `[1,2,4]` n_estimators = `[10,30,50,100]`
4	MLP	hidden_layer_sizes = `[(2,2,), (20,20,), (50,50,)]` n_iter = `[100,200,300]`

  - สร้างโมเดลและทำการ Train ด้วย Best Parameters ที่ได้จาก GridSearchCV()
  - คำนวนค่า Model Performance: Confusion Matrix
  - แสดงผลการคำนวน Model Performance ของแต่ละ Model ในรูปแบบของรูปภาพ

- ตอบคำถามท้ายการทดลอง

4.2 :สร้าง MLP สำหรับ Regression Model เพื่อประมาณค่า Simple Sine Wave

- Library ที่ใช้

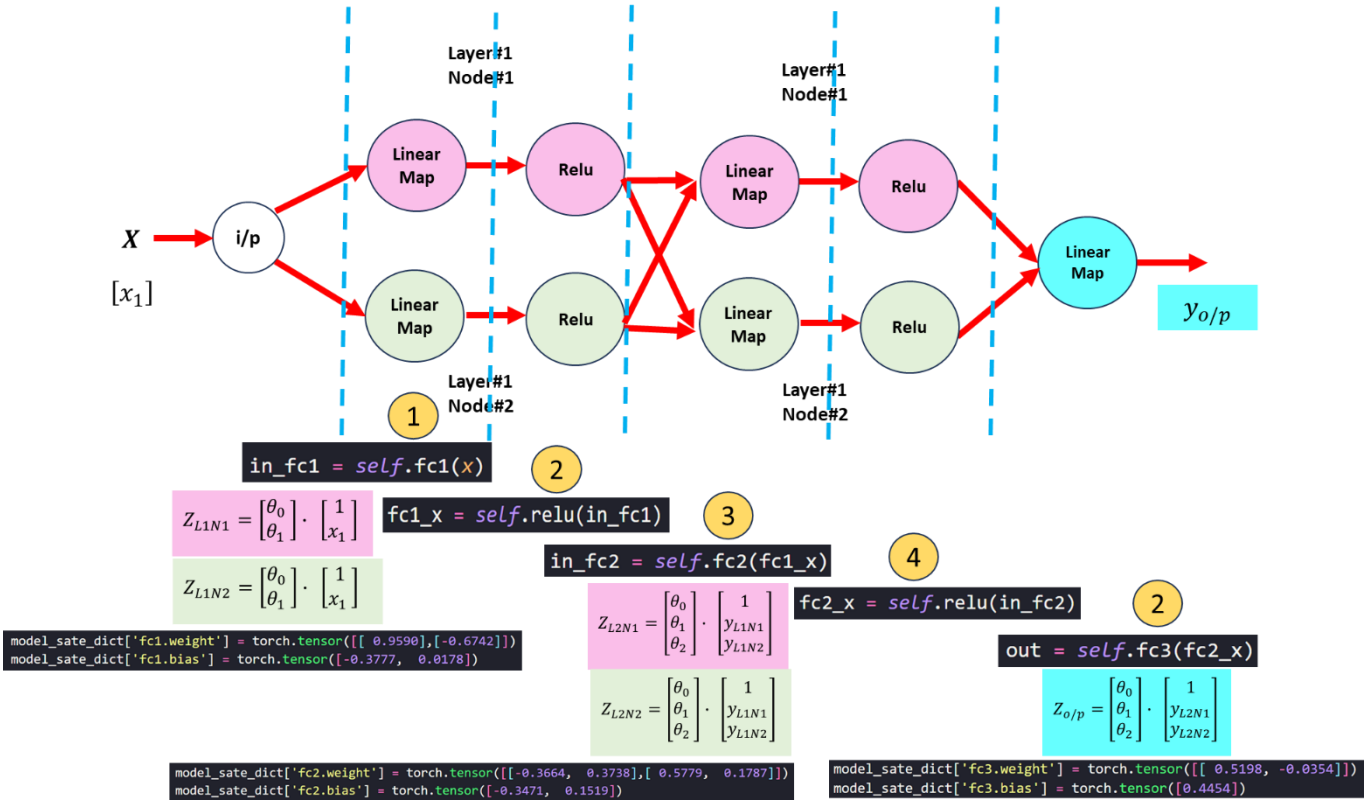
```
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
from collections import OrderedDict
```

- สร้างข้อมูล Sine wave สำหรับสอนโมเดล พร้อมแสดงกราฟ โดยกำหนดให้

$$y = sin(x)$$

Where $-\pi \leq x < \pi$ อย่างน้อย 9 มุม

- เก็บข้อมูล x, y เป็น Dataset Array: dataset = [[x, y]]
- แสดงกราฟความสัมพันธ์ x, y
- สร้างคลาสเพื่อเป็นตัวแทน MLP model ด้านล่าง



โดยกำหนดให้

- Inherite จาก class nn.Module
- สร้าง Constructor เพื่อ initial Super Class: MLP model
- กำหนด Linear Kernel Mapping Node ที่จะใช้สำหรับ Fully connected layer 1-3 (fc1 – fc3) เป็น nn.Linear() โดย fc1, fc2 เป็น Hidden Layer และ fc3 เป็น output (o/p) layer
- กำหนด Activation Node เป็น Relu()

```python
class MLP(nn.Module):
    def __init__(self, input_size=1, hidden_size=2, output_size=1):
        """
        Multi-Layer Perceptron (MLP) class.

        Args:
            input_size (int): The size of the input layer.
            hidden_size (int): The size of the hidden layer.
            output_size (int): The size of the output layer.
        """
        super(MLP, self).__init__()

        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = None
        self.fc3 = None
        self.relu = None
```

– สร้าง Forward Function เพื่อสร้างเส้นทางการ process ข้อมูล ไปยัง Node ประมวลผลต่างๆ ในแต่ละ Layer ตามโครงสร้างที่กำหนด

```python
def forward(self, x):
    """
    Forward pass of the MLP.

    Args:
        x (torch.Tensor): The input tensor.

    in_fc1 = self.fc1(x)
    fc1_x = self.relu(in_fc1)
    in_fc2 = self.fc2(fc1_x)
    fc2_x = self.relu(in_fc2)
    out = self.fc3(fc2_x)
    return in_fc1, fc1_x, in_fc2, fc2_x, out
```

– สร้าง object instance ของ Class MLP และกำหนดพารามิเตอร์ดังนี้

Loss Function: MSE (Mean Square Error)

Optimizer: SGD() (Stochastic Gradient Descent)

Learning Rate: 0.08

```python
mlp = MLP()
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(mlp.parameters(), lr=0.008)
print(mlp)
```

- กำหนดค่า initial model state จาก dict ที่กำหนด
  - กำหนด **model_sate_dict** เพื่อเก็บ weight parameters ตามลำดับ fc1 – fc3

    model_sate_dict = OrderedDict()
  - กำหนด Dictionary ของ Initial weights tensor
    - Linear Kernel Mapping Weights (fc1)

$$Z_{L1N1} = \begin{bmatrix} \theta_0 = -0.3777 \\ \theta_1 = 0.9590 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \end{bmatrix} \qquad Z_{L1N2} = \begin{bmatrix} \theta_0 = 0.0178 \\ \theta_1 = -0.6742 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \end{bmatrix}$$

fc1.weight = $[\theta_{1\,L1N1} \quad \theta_{1\,L1N2}] = [\ [0.9590], \quad [-0.6742]\ ]$

fc1.bias = $[\theta_{0\,L1N1} \quad \theta_{0\,L1N2}] = [-0.3777, \quad 0.0178]$

```
model_sate_dict['fc1.weight'] = torch.tensor([[ 0.9590],[-0.6742]])
model_sate_dict['fc1.bias'] = torch.tensor([-0.3777,  0.0178])
```

- Linear Kernel Mapping Weights (fc2)

$$Z_{L2N1} = \begin{bmatrix} \theta_0 = -0.3471 \\ \theta_1 = -0.3664 \\ \theta_2 = 0.3738 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \end{bmatrix} \qquad Z_{L2N2} = \begin{bmatrix} \theta_0 = 0.1519 \\ \theta_1 = 0.5779 \\ \theta_2 = 0.1787 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \end{bmatrix}$$

- Linear Kernel Mapping Weights (fc3)

$$Z_{o/p} = \begin{bmatrix} \theta_0 = 0.4454 \\ \theta_1 = 0.5198 \\ \theta_2 = -0.0354 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \end{bmatrix}$$

- กำหนดค่า initial weight ให้กับโมเดล

  mlp.load_state_dict(model_sate_dict)

- คำนวน MLP prediction ของ dataset ด้วย initial weights ที่โหลดเข้า MLP ในข้อก่อนหน้า

  # Convert dataset -> tensor

  dataset_tensor = torch.tensor(dataset).float()

  # Convert tensor[x] -> 2D array ของ $\in R^{rows\ x\ 1}$

  # MLP prediction ด้วย mlp()

  in_fc1, fc1_x, in_fc2, fc2_x, out = mlp(dataset_tensor[:,0].unsqueeze(1))

- แสดง subplot ของโมเดลในแต่ละ node จาก initial weights ที่กำหนด ดังนี้
  - แสดงกราฟ Model **Layer-1, Node-1** เพื่อเปรียบเทียบเทียบ ผลลัพธ์หลัง Linear Kernel Mapping (fc1) กับ หลัง Activation Function
  - แสดงกราฟ Model **Layer-1, Node-2** เพื่อเปรียบเทียบเทียบ ผลลัพธ์หลัง Linear Kernel Mapping (fc1) กับ หลัง Activation Function
  - แสดงกราฟ Model **Layer-2, Node-1** เพื่อเปรียบเทียบเทียบ ผลลัพธ์หลัง Linear Kernel Mapping (fc2) กับ หลัง Activation Function
  - แสดงกราฟ Model **Layer-2, Node-2** เพื่อเปรียบเทียบเทียบ ผลลัพธ์หลัง Linear Kernel Mapping (fc2) กับ หลัง Activation Function
  - แสดงกราฟ Model **Output (o/p)** เพื่อเปรียบเทียบเทียบ ผลลัพธ์หลัง Linear Kernel Mapping (fc3) กับ หลัง Activation Function

- สอน MLP model
  - กำหนดจำนวนรอบในการเทรน (Epoch)
  - กำหนด $log\_interval = \Delta epoch$ ที่ต้องการ พิมพ์ ค่า Loss
  - Loop สอนโมเดลตามจำนวนรอบ epoch ที่กำหนด
    - MLP prediction

      in_fc1, fc1_x, in_fc2, fc2_x, out = mlp(dataset_tensor[:,0].unsqueeze(1))
    - คำนวน Loss ของผลลัพธ์ out จาก O/P Node เทียบกับ $y_{real} = y$

      loss = criterion(out, y)
    - ทำการคำนวน Loss จาก Backward Propagation เพื่อให้ optimizer นำไปใช้ในการปรับ weight ในรอบการสอนนั้น

      Optimizer.zero_grad()

      Loss.backward()

      Optimizer.setp()
    - แสดงค่า loss ทุกๆ รอบ Log_interval และ เก็บผลลัพธ์ของทุก Node ทุก Layer เพื่อนำไปแสดงกราฟ หลังจากเทรนโมเดลเสร็จ

```
### START CODE HERE ###
num_epochs = None
log_interval = None
frame = []

for epoch in range(num_epochs):

    in_fc1, fc1_x, in_fc2, fc2_x, out = mlp(None)

    output_dict['in_fc1'].append(in_fc1.detach().numpy())
    output_dict['fc1_x']
    output_dict['in_fc2']
    output_dict['fc2_x']
    output_dict['out']

    loss = criterion(out, None)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch+1) % log_interval == 0:
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item()}")
        frame.append(output_dict)
```

- แสดง weights ที่ได้หลังการเทรน

  print(mlp.state_dict())
- แสดงกราฟผลลัพธ์ $\hat{y}$ จาก MLP เทียบกับ $y_{real} = \sin(x)$
- Save ภาพ gif animation ของ subplot ของผลลัพธ์ กราฟที่ได้ ทุก Node ทุก Layer ในแต่ละรอบ การเทรนโมเดล

```
### START CODE HERE ###
from IPython.display import HTML
import matplotlib.animation as animation
fig, axs = plt.subplots(2, 3, figsize=(15, 5))
ims = []

for f in frame:


    ims.append(im)

ani = animation.ArtistAnimation(fig, ims, interval=300, blit=True)
ani_js = ani.to_jshtml()

HTML(ani_js)
```

- ตอบคำถามท้ายการทดลอง

การส่งงาน

1. ให้ Staff ตรวจในห้อง (อย่างน้อยข้อ 4.1)
2. ส่งเอกสารในฟอร์มส่ง Lab (https://forms.gle/pWUx2vHrZq29Axnx8)

   2.1 source code

   2.2 เอกสาร (pdf) อธิบายการทำงานของ source code (ถ้าไม่ได้อธิบายในห้อง)

   2.3 การตั้งชื่อไฟล์ “Lab#4_ชื่อกลุ่ม_รหัสสมาชิก#1_รหัสสมาชิก#2.pdf”
3. กำหนดส่ง ในฟอร์ม ก่อนวันทำแลปครั้งต่อไป