Artificial Intelligence and Machine Learning

(ใบงานที่ 3)

เสนอ

อาจารย์ รุจิพันธุ์ โกษารัตน์

จัดทำโดย

67543210031-0 นายธนภัทร นุกูล

คณะวิศวกรรมศาสตร์

สาขาวิชา วศ.บ.วิศวกรรมซอฟต์แวร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา เชียงใหม่

Ornamental Plant Images Dataset

ชื่อชุดข้อมูล: Ornamental Plant Images Dataset

แหล่งที่มา (Source): Kaggle (Contributor: Muhammad Irvan Arfirza)

ประเภทของปัญหา: การจำแนกประเภทรูปภาพ (Image Classification) แบบ Supervised Learning

วัตถุประสงค์: เพื่อใช้ฝึกฝนโมเดลปัญญาประดิษฐ์ให้สามารถแยกแยะสายพันธุ์ของไม้ประดับชนิดต่างๆ จากภาพถ่ายได้

Random Forest

การทำงานของโค้ด

1. **นำเข้าไลบรารี**

- random: สุ่มภาพตัวอย่างเพื่อแสดงผล
- kagglehub: ดาวน์โหลด dataset จาก Kaggle อัตโนมัติ
- os: จัดการไฟล์และค้นหา path ของโฟลเดอร์
- numpy: จัดการข้อมูลเชิงตัวเลขและอาเรย์ (Array)
- matplotlib: แสดงผลกราฟและรูปภาพ
- PIL.Image: เปิด, แปลงสี (Grayscale) และย่อขนาดภาพ
- sklearn.model_selection: แบ่งข้อมูลเป็นชุด Train และ Test
- **sklearn.ensemble: นำเข้าโมเดล RandomForestClassifier (ส่วนที่ต่างจาก Decision Tree)**
- metrics: วัดผลความแม่นยำ (Accuracy) และแสดง Report

2. **โหลด Dataset จาก Kagglehub**

- dataset_path = kagglehub.dataset_download(...)
- ดาวน์โหลดชุดข้อมูล "Decorative Plant Image Dataset" มาเก็บไว้ในเครื่อง

3. **กำหนดค่าพารามิเตอร์ (Config)**

- IMG_SIZE = (64, 64): กำหนดให้ทุกภาพถูกย่อเหลือขนาด 64x64 pixel
- MAX_IMAGES_PER_CLASS: กำหนดจำนวนรูปสูงสุดต่อพันธุ์ไม้ที่จะโหลด (เพื่อลดเวลาประมวลผล)

4. **ค้นหาโฟลเดอร์ที่มีรูปภาพ**

- def find_image_folder(start_path)
- ใช้ os.walk() ไล่ค้นหาจนกว่าจะเจอโฟลเดอร์ที่มีไฟล์ .jpg หรือ .png จริงๆ เพื่อแก้ปัญหา Path ซ้อนกัน

5. **โหลดและเตรียมข้อมูลภาพ (Preprocessing)**

- def load_data(root_path):
- วนลูปอ่านไฟล์รูปภาพทีละรูป
- **Convert L**: แปลงเป็นภาพขาว-ดำ (Grayscale)
- **Resize**: ย่อขนาดภาพเป็น 64x64
- **Flatten**: แปลงภาพ 2 มิติ ให้เป็นเส้นตรง (Array 1 มิติ) เพื่อเตรียมเข้าโมเดล
- images: เก็บข้อมูล pixel ของภาพ
- labels: เก็บค่า index ของพันธุ์ไม้ (0, 1, 2, ...)
- class_names: เก็บชื่อพันธุ์ไม้จริง

6. **แบ่งข้อมูลเป็น Training และ Testing Set**

- X_train, X_test, y_train, y_test = train_test_split(..., test_size=0.3)
- แบ่งข้อมูล 70% ให้โมเดลเรียนรู้ (Training)
- แบ่งข้อมูล 30% ไว้สอบวัดผล (Testing)

7. **สร้างและฝึกโมเดล Random Forest**

- **clf = RandomForestClassifier(n_estimators=100, ...)**
- สร้างโมเดลโดยจำลองต้นไม้ตัดสินใจ (Decision Tree) จำนวน 100 ต้น (n_estimators=100)
- clf.fit(X_train, y_train): สั่งให้ต้นไม้ทั้ง 100 ต้นเรียนรู้ข้อมูลพร้อมกัน
- ใช้หลักการ **Ensemble Learning (Voting)**: ให้ต้นไม้ทุกต้นช่วยกันโหวตคำตอบ

8. **ประเมินประสิทธิภาพและบันทึกผล**

- y_pred = clf.predict(X_test): ให้โมเดลลองทำนายภาพในชุด Test
- accuracy_score: คำนวณความแม่นยำรวมเป็นเปอร์เซ็นต์
- classification_report: แสดงค่า Precision, Recall, F1-score ของแต่ละพันธุ์ไม้
- บันทึกผลลัพธ์ลงไฟล์ Text (.txt) เพื่อนำไปทำรายงาน

9. **แสดงผลลัพธ์การทำนาย (Visualization)**

- สุ่มภาพตัวอย่างจาก Test set มา 5 รูป
- plt.imshow: แสดงรูปภาพที่สุ่มได้
- เปรียบเทียบ **ค่าจริง (True)** vs **ค่าที่ทำนาย (Pred)**
- **Color Coding**: แสดงข้อความสีเขียวถ้าทายถูก, สีแดงถ้าทายผิด
- บันทึกรูปผลลัพธ์เป็นไฟล์ภาพ (.png)
- plt.show(): แสดงผลขึ้นหน้าจอ

Code Python:

```
import random
import kagglehub
import os
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# ==========================================
# Configuration
# ==========================================
IMG_SIZE = (64, 64)
MAX_IMAGES_PER_CLASS = None  # None = Load all images

# ==========================================
# 1. Dataset Preparation
# ==========================================
print("--- Downloading Dataset ---")
try:
    dataset_path = kagglehub.dataset_download(
        "muhammadirvanarfirza/decorative-plant-image-dataset")
    print(f"Dataset Path: {dataset_path}")
except Exception as e:
    print(f"Error downloading: {e}")
    exit()


def find_image_folder(start_path):
    """Recursively find the directory containing image files."""
    for root, dirs, files in os.walk(start_path):
        if len(files) > 0:
            if any(f.lower().endswith(('.png', '.jpg', '.jpeg')) for f in
files):
                return os.path.dirname(root)
    return start_path


def load_data(root_path):
    """Load images, resize, flatten, and assign labels."""
    images = []
    labels = []
    class_names = []

    target_path = find_image_folder(root_path)
```

```
    if os.path.exists(target_path):
        folder_list = sorted(os.listdir(target_path))

        for folder in folder_list:
            folder_path = os.path.join(target_path, folder)

            if os.path.isdir(folder_path) and not folder.startswith('.'):
                files_inside = os.listdir(folder_path)
                image_files = [f for f in files_inside if f.lower().endswith(
                    ('.jpg', '.png', '.jpeg'))]

                if len(image_files) > 0:
                    class_names.append(folder)
                    label_index = len(class_names) - 1
                    count = 0

                    for img_file in image_files:
                        if MAX_IMAGES_PER_CLASS is not None and count >=
MAX_IMAGES_PER_CLASS:
                            break

                        try:
                            img_path = os.path.join(folder_path, img_file)
                            with Image.open(img_path) as img:
                                # Convert to Grayscale
                                img = img.convert('L')
                                img = img.resize(IMG_SIZE)   # Resize
                                # Convert to NumPy Array
                                img_array = np.array(img)

                                images.append(img_array.flatten())
                                labels.append(label_index)
                                count += 1
                        except Exception:
                            pass

                    print(f"Loaded class '{folder}': {count} images")

    return np.array(images), np.array(labels), class_names


# Load and process data
print("\n--- Processing Images ---")
X, y, class_names = load_data(dataset_path)

if len(X) == 0:
    print("Error: No images found.")
    exit()
```

```python
print(f"Total images: {len(X)}")
print(f"Feature shape: {X.shape}")

# ========================================
# 2. Model Training (Random Forest)
# ========================================
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)

print("\n--- Training Random Forest Model ---")
clf = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
clf.fit(X_train, y_train)
print("Training Complete!")

# ========================================
# 3. Evaluation & Visualization
# ========================================
print("\n--- Generating Report ---")

y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=class_names)

# Save report to text file
result_text = f"""
========================================
Training Report: Random Forest Classifier
========================================
Image Size: {IMG_SIZE}
Total Images: {len(X)}
Training Set: {len(X_train)}
Testing Set: {len(X_test)}
Accuracy: {acc * 100:.2f}%
----------------------------------------
Classification Report:
{report}
"""

with open("rf_report_results.txt", "w", encoding="utf-8") as f:
    f.write(result_text)
print("Report saved to 'rf_report_results.txt'")

# Visualize predictions
try:
    num_samples = 5
    sample_count = min(num_samples, len(X_test))
    indices = random.sample(range(len(X_test)), sample_count)
```

```
    plt.figure(figsize=(15, 4))
    for i, idx in enumerate(indices):
        ax = plt.subplot(1, num_samples, i + 1)

        sample_img = X_test[idx].reshape(IMG_SIZE)
        pred_cls = class_names[clf.predict([X_test[idx]])[0]]
        true_cls = class_names[y_test[idx]]

        color = 'green' if pred_cls == true_cls else 'red'

        plt.imshow(sample_img, cmap='gray')
        plt.title(f"Actual: {true_cls}\nPred: {pred_cls}",
                  color=color, fontsize=10)
        plt.axis('off')

    plt.suptitle(
        f"Random Forest Predictions (Acc: {acc*100:.1f}%)", fontsize=14)
    plt.tight_layout()
    plt.savefig("rf_prediction_samples.png", dpi=300)
    print("Prediction samples saved to 'rf_prediction_samples.png'")
    plt.show()

except Exception as e:
    print(f"Error plotting samples: {e}")
```

Result:


```
========================================
Training Report: Random Forest Model (Full Dataset)
========================================
Model: Random Forest Classifier (100 Trees)
Image Size: (64, 64)
Max Images per Class: All (Unlimited)
Total Images: 11581
Training Set: 8106
Testing Set: 3475
----------------------------------------
Accuracy: 80.00%
----------------------------------------
Classification Report:
```

	precision	recall	f1-score	support
Anggrek Pot-Potted Orchid	0.97	1.00	0.98	118
Anggrek Potong-Cut Orchid	0.86	0.86	0.86	134
Anthurium Bunga-Flamingo Lily Flower	0.81	0.92	0.86	108
Anthurium Daun-Anthurium	0.81	0.79	0.80	131
Balanceng-Dieffenbacia	0.84	0.74	0.79	119
Bromelia-Bromelia	0.88	0.94	0.91	127
Bugenvil-Bugenvil	0.60	0.67	0.63	109
Dracaena-Dracaena	1.00	0.87	0.93	132
Euphorbia-Euphorbia	0.95	1.00	0.97	125
Hanjuang-Cordyline	0.59	0.36	0.45	121
Herbras-Gerbera	0.74	0.93	0.82	131
Kamboja Jepang-Adenium	0.93	0.92	0.92	113
Keladi Hias-Caladium	0.74	0.84	0.78	128
Krisan-Chrysantemum	0.85	1.00	0.92	110
Mawar-Rose	0.62	0.57	0.59	111
Melati-Jasmine	0.86	0.98	0.91	121
Monstera-Monstera	0.99	0.93	0.96	129
Pakis-Leather Leaf Fern	0.79	0.82	0.81	97
Palem-Palm	0.74	0.91	0.82	109
Pedang-pedangan-Sanseviera	0.89	0.81	0.85	129
Philodendron-Philodendron	0.76	0.82	0.79	122
Pisang-Pisangan-Heliconia	0.75	0.81	0.78	112
Puring-Croton	0.81	0.86	0.84	117
Sedap Malam-Tuberose	0.92	0.94	0.93	129
Soka-Ixora	0.98	1.00	0.99	120
Sri Rejeki-Aglaonema	0.89	0.71	0.79	119
Tulip	0.39	0.32	0.35	112
daisy	0.43	0.38	0.40	113
dandelion	0.52	0.42	0.46	129
accuracy			0.80	3475
macro avg	0.79	0.80	0.79	3475
weighted avg	0.79	0.80	0.79	3475